

ANALYTICS OF SMALL DATA

- *A MODE OF THINKING*

Shuai Huang & Houtao Deng

Contents

PREFACE.....	9
CHAPTER 1: INTRODUCTION	12
Overview of a Data Analytics Pipeline	12
Structure of the Chapters	13
Topics in a Nutshell	14
CHAPTER 2: ABSTRACTION	16
II. Regression Models	18
III. Tree Models	42
CHAPTER 3: RECOGNITION	59
II. Logistic Regression Model.....	60
III. A Product Ranking Problem by Pairwise Comparison.....	73
CHAPTER 4: COMPUTATION	78
II. How Bootstrap Works	79
III. Random Forests	93
CHAPTER 5: PERFORMANCE.....	112

II. Cross-Validation	115
III. Out-of-bag error in Random Forest.....	132
CHAPTER 6: DIAGNOSIS.....	140
II. Residual Analysis in Regression	141
III. Diagnosis in Random Forests.....	150
IV. Clustering.....	159
CHAPTER 7: BALANCE.....	170
II. Support Vector Machine	170
III. Ensemble Learning	196
CHAPTER 8: SCALABILITY	215
II. LASSO	216
III. Principal Component Analysis.....	234
IV. Variable Selection by Random Forests.....	244
CHAPTER 9: CRAFTSMANSHIP	264
II. Kernel Regression Model	264
III. Conditional Variance Regression Model	272
IV. System Monitoring as a Decision Tree Model.....	281
CHAPTER 10: SYNTHESIS.....	300
II. InTrees	301
CONCLUSION	326

CHAPTER 1: INTRODUCTION

Overview of a Data Analytics Pipeline

A typical data analytics pipeline consists of several major pillars. In the example shown in Figure 1.1, it has four pillars: sensor and devices, data preprocessing and feature engineering, feature selection and dimension reduction, modeling and data analysis. While this is not the only way to present the diverse data pipelines in real-world, they more or less resemble this architecture.

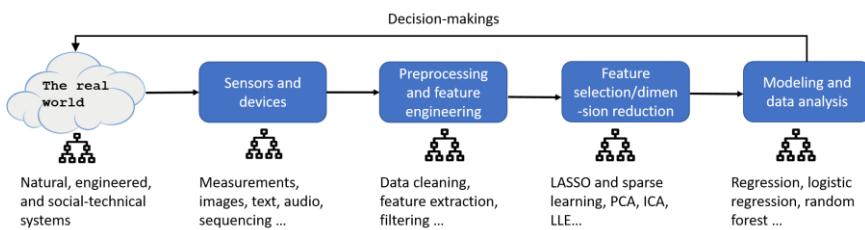


Figure 1.1: Overview of a data analytics pipeline

The pipeline starts with a real-world problem, for which we are not sure about the underlying system/mechanism, but we are able to characterize the system by defining some variables. Then, we could develop sensors and devices to acquire measurements of these variables. These measurements, we call as data, are objective evidences that we can use to explore the statistical principles or mechanistic laws regulating the system behaviors. But, before analyzing the data and building models using the data, in practice, the data preprocessing and feature engineering are important. For example, some signals acquired by sensors are not interpretable or not easily compatible with human sense, such as the signal acquired by MRI scanning machines in the Fourier space. Data preprocessing also refers to removal of outliers or imputation of missing data, detection and removal of redundant features, to name a few. After the preprocessing, we may conduct feature selection and dimension reduction to distill or condense signals in the data and reduce noise. Finally, we are ready to conduct modeling and data analysis on the prepared dataset to gain knowledge and build prediction models of the real-world system. Decision-makings such as prediction, intervention, and control policies can be derived based on the fitted models to optimize and control the real-world system.

This book focuses on the last two pillars of this pipeline, the modeling, data analysis, feature selection, and dimension reduction methods. But it is helpful to keep in mind of the big picture of a data analytics pipeline. Because in practice, what works is the whole pipeline.

Structure of the Chapters

The structures of the Chapters follow the same manner.

- Each chapter will introduce two or three techniques. In most cases, one technique is about regression model while another one is about tree model.
- For each technique, we will highlight the intuition and rationale behind it.

- Then, we articulate the intuition, use math to formulate the learning problem, and present the full version of the analytic formulation. But, it is always important to remember its intuitive underpinning.
- Then, we use R to implement the technique on both simulated and real-world dataset, present the analysis process (together with R code), show the dynamics in the analysis process, and comment on the results.
- Some remarks are also made to enhance understanding of the techniques, reveal their different natures by other perspectives, reveal their limitations, and mention existing remedies to overcome these limitations.

Topics in a Nutshell

Data models – regression based techniques:

- Chapter 2: Linear regression, least-square estimation, hypothesis testing, why normal distribution, its connection with experimental design, R-squared.
- Chapter 3: Logistic regression, generalized least square estimation, iterative reweighted least square (IRLS) algorithm, approximated hypothesis testing, Ranking as a linear regression
- Chapter 4: Bootstrap, data resampling, nonparametric hypothesis testing, nonparametric confidence interval estimation
- Chapter 5: Overfitting and underfitting, limitation of R-squared, training dataset and testing dataset, random sampling, K-fold cross validation, the confusion matrix, false positive and false negative, and Receiver Operating Characteristics (ROC) curve
- Chapter 6: Residual analysis, normal Q-Q plot, Cook's distance, leverage, multicollinearity, subset selection, heterogeneity, clustering, gaussian mixture model (GMM), and the Expectation-Maximization (EM) algorithm
- Chapter 7: Support Vector Machine (SVM), generalize data versus memorize data, maximum margin, support vectors, model complexity and regularization, primal-dual formulation, quadratic

programming, KKT condition, kernel trick, kernel machines, SVM as a neural network model

- Chapter 8: LASSO, sparse learning, L1-norm and L2-norm regularization, Ridge regression, feature selection, shooting algorithm, Principal Component Analysis (PCA), eigenvalue decomposition, scree plot
- Chapter 9: Kernel regression as generalization of linear regression model, kernel functions, local smoother regression model, k-nearest regression model, conditional variance regression model, heteroscedasticity, weighted least square estimation, model extension and stacking

Algorithmic models – tree based techniques:

- Chapter 2: Decision tree, entropy gain, node splitting, pre- and post-pruning, empirical error, generalization error, pessimistic error by binomial approximation
- Chapter 4: Random forest, Gini index, weak classifiers, probabilistic mechanism why random forest works
- Chapter 5: Out-of-bag (OOB) error in random forest
- Chapter 6: Importance score, partial dependency plot, residual analysis
- Chapter 7: Ensemble learning, Adaboost, sampling with (or without) replacement
- Chapter 8: Importance score in random forest, regularized random forests (RRF), guided regularized random forests (GRRF)
- Chapter 9: System monitoring reformulated as classification, real-time contrasts method (RTC), design of monitoring statistics, sliding window, anomaly detection, false alarm
- Chapter 10: Integration of tree models and regression models in inTrees, random forest as a rule generator, rule extraction, pruning, selection, and summarization, confidence and support of rules, variable interactions, rule-based prediction

In this book, we will use lower-case letters, e.g., x , to represent scalars, bold-face lower-case letters, e.g., \mathbf{v} , to represent vectors, and bold-face upper-case letters, e.g., \mathbf{W} , to represent matrices.

CHAPTER 2: ABSTRACTION

REGRESSION and TREE MODELS

I. Overview

Chapter one is about “Abstraction”. It concerns how we model and formulate a problem using *specific mathematical models*. Abstraction is powerful. With identification of a few main entities (usually called as variables or features) from the problem, and characterization of their relationships, we can free ourselves from the application context and focus on the study of these interconnected entities as a pure mathematical system. Consequences can be analytically (rather than speculatively) established within this abstracted framework, while phenomenon in the context could be identified as special instances of this abstracted model.

Generally, there are two main types of cultures for statistical modeling. Prof. Leo Breiman made these two cultures explicit as he articulated in his seminar paper¹. One is the “data modeling” culture, while another one is the “algorithmic modeling” culture. In this book, we will focus on two

¹ Leo Breiman, *Statistical Modeling: The Two Cultures*. *Statistical Science*, 2001.

models that are representative of each culture: the linear regression models (data modeling) and decision tree models (algorithmic modeling). Linear regression is a great example about statistics-driven considerations in modeling, while decision tree is a great example about computational- and nonparametric-driven considerations in modeling.

Many real-world problems usually present themselves in the form as a mystery, as highlighted as a blackbox in Figure 2.1. In these problems, there is usually an output variable (denoted as y) we care about and want to predict; meanwhile, to help us better understand the uncertainty of the output variable, we have other variables which we call as predictors (denoted as x_1, x_2, \dots, x_p). We know that there are relationships between the predictors and the output, but these relationships are unknown, due to our lack of understanding of the system. It is not always plausible or economically feasible to develop a Newtonian style characterization of the system using differential equations.

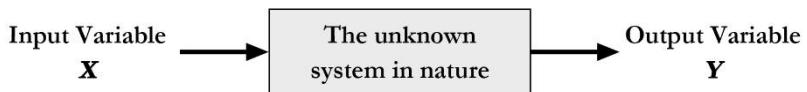


Figure 2.1: The blackbox nature of many data science problems

A common criterion for evaluating the success of any model, no matter what type of culture it belongs to, is the prediction performance on the output variable given the input variables. It is fair to say that, almost all the models in both cultures could be summarized using a generic form:

$$y = f(\mathbf{x}) + \epsilon,$$

where $f(\mathbf{x})$ reflects the deterministic part of y that can be determined by knowing \mathbf{x} , and ϵ reflects the uncertain part of y that could not be determined by \mathbf{x} alone. In some texts, $f(\mathbf{x})$ is also called the model of the mean structure, i.e., since given any value of \mathbf{x} we can predict y in the sense of an average; ϵ is usually called as the error term, noise term, or residual

term. Thus, $f(\mathbf{x})$ is a function of \mathbf{x} while ϵ is usually a distribution such as Gaussian distribution with mean as zero.

With this understanding, we could summarize the different principles of both cultures in designing their belonging models:

Table 2.1: Comparison between two cultures of models

	$f(x)$	ϵ	“Cosmology”
Data Modeling	Explicit form (e.g., linear regression)	Statistical distribution (e.g., Gaussian)	Imply <i>Cause and effect</i> ; articulate uncertainty
Algorithmic Modeling	Implicit form (e.g., tree model)	Rarely modeled as structured uncertainty; only acknowledged as meaningless noise	Look for accurate surrogate for prediction; to <i>fit</i> the data rather than to <i>explain</i> the data

II. Regression Models

II.1 Rationale and Formulation

Let's consider a simple regression model, where there is only one predictor x to predict the outcome y . Linear regression model assumes a linear form of $f(x)$, e.g.,

$$f(x) = \beta_0 + \beta_1 x,$$

and a distribution form for ϵ , e.g.,

$$\epsilon \sim N(0, \sigma_\epsilon^2).$$

With this model, for any given value of x , we could predict the value of y as $\beta_0 + \beta_1 x$. Apparently, a few assumptions have been made:

- There is linear relationship between x and y . And this linear relationship remains the same for all the values of x . This is often referred as a global relationship between x and y . Sometimes this assumption of global relationship is too strong, e.g., as shown in

the Figure 2.2 below, in many drug research works, it is found that the dose (x) is related to the effect of the drug (y) in a varying manner that depends on the value of x . But, still, from Figure 2.2 we can also see that the linear line captures an essential component in the relationship between x and y , providing a good statistical approximation.

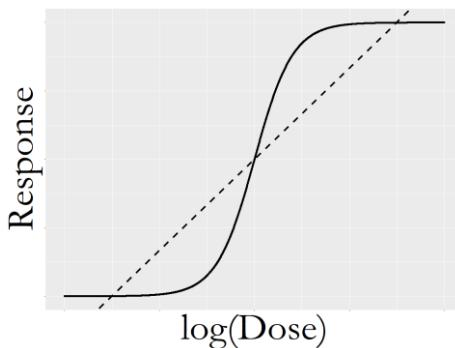


Figure 2.2: Complex relationship between dose (x) and drug response (y), while the linear line also provides a good statistical approximation

- The model suggests a fundamental unpredictability of y . That is to say, if y is generated by a combination of the signal (the $f(x)$) and the noise (ϵ), we could never predict the noise part. This has at least two implications. First, we can quantify the predictability of a dataset, by taking the ratio of $\frac{\sigma_y^2 - \sigma_\epsilon^2}{\sigma_y^2}$. Here, σ_y^2 is the overall variance of the output regardless of any predictor information. This ratio is named as **R-squared**, that ranges from 0 (zero predictability) to 1 (perfect predictability). Second, the *significance* of x in predicting y , and the *accuracy* of x in predicting y , are two different concepts. A predictor x could be inadequate in predicting y , e.g., the R-squared could be as low as 0.1, but it still could be

statistically significant. This happens a lot in social science research and education research projects.

- The noise is usually modeled as gaussian distribution, but this assumption could be relaxed. Violation of the gaussian assumption for ε could be a concern in many applications, but not as severe as other violations such as outliers in the dataset. Of course, this assertion is empirical, only mentioned here to guide practices, and should not be taken as a strict rule.

II. 2 Theory/Method

Parameter Estimation: The regression parameters could be estimated by the least-square estimation method. A training dataset is collected to estimate the unknown parameters in the model. The basic idea is, the best parameters should fit the training data as much as possible. This is illustrated in Figure 2.3, where two principles to fit a linear regression model are shown. The vertical offsets shown in the right of Figure 2.3 is the most popular approach though. Comparing with the perpendicular offsets shown in the left of Figure 2.3, the vertical offset leads to tractability in analytic forms, which is thus more preferred.

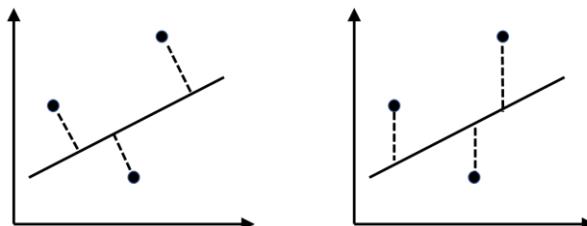


Figure 2.3: Two principles to fit a linear regression model: (left) perpendicular offsets; (right) vertical offsets.

Actually, the principle of minimizing vertical offsets leads to the least-squares estimation of linear regression models. We can exercise the least squares estimation using the simple regression model. The objective to determine the optimal line (or equivalently we can say to determine the optimal regression parameters), based on the principle suggested in the right one in Figure 2.3, is the sum of the squared of the vertical derivations of the observed data points from the line. Suppose that we have collected N data points, denoted as, (x_n, y_n) for $n = 1, 2, \dots, N$.

Then, the sum of the squared of the vertical derivations of the observed data points from the line is:

$$l(\beta_0, \beta_1) = \sum_{n=1}^N [y_n - (\beta_0 + \beta_1 x_n)]^2.$$

To estimate β_0 and β_1 is to minimize this least-square loss function $l(\beta_0, \beta_1)$. Thus, we could take derivatives of $l(\beta_0, \beta_1)$ regarding the two parameters and set them to be zero, to derive the estimation equations:

$$\begin{aligned}\frac{\partial l(\beta_0, \beta_1)}{\partial \beta_0} &= -2 \sum_{n=1}^N [y_n - (\beta_0 + \beta_1 x_n)] = 0, \\ \frac{\partial l(\beta_0, \beta_1)}{\partial \beta_1} &= -2 \sum_{n=1}^N x_n [y_n - (\beta_0 + \beta_1 x_n)] = 0.\end{aligned}$$

Putting these into a succinct way, we can derive

$$\begin{bmatrix} N & \sum_{n=1}^N x_n \\ \sum_{n=1}^N x_n & \sum_{n=1}^N x_n^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \sum_{n=1}^N y_n \\ \sum_{n=1}^N x_n y_n \end{bmatrix}.$$

Thus, we can solve these two equations and derive the estimator of β_0 and β_1 as

$$\begin{aligned}\beta_0 &= \frac{(\sum_{n=1}^N y_n)(\sum_{n=1}^N x_n^2) - (\sum_{n=1}^N x_n)(\sum_{n=1}^N x_n y_n)}{n \sum_{n=1}^N x_n^2 - (\sum_{n=1}^N x_n)^2}, \\ \beta_1 &= \frac{\sum_{n=1}^N x_n y_n - N \bar{x} \bar{y}}{\sum_{n=1}^N x_n^2 - N \bar{x}^2}.\end{aligned}$$

While the above mathematical expression seems to be complex, there is another angle to take a look at it. Notice that the sample correlation between x and y is:

$$\text{cov}(x, y) = \frac{\sum_{n=1}^N (x_n - \bar{x})(y_n - \bar{y})}{N-1} = \frac{\sum_{n=1}^N x_n y_n - N \bar{x} \bar{y}}{N-1},$$

Also, the sample variance is defined as

$$var(x) = \frac{\sum_{n=1}^N x_n^2 - N\bar{x}^2}{N-1}.$$

We can rewrite the estimators of β_0 and β_1 as

$$\begin{aligned}\beta_0 &= y - \beta_1 x, \\ \beta_1 &= \frac{cov(x,y)}{var(x)}.\end{aligned}$$

A simple example: Let's practice the estimation method using a simple example. The dataset is shown in Table 2.2:

Table 2.2: An exemplary dataset

X	1	3	3	5	5	5	6	8	9
Y	2	3	5	4	6	5	7	8	

The R-code to verify your calculation:

```
## Simple example of regression with one predictor
data = data.frame(rbind(c(1,2),c(3,3),c(3,5),c(5,4),c(5,6),c(6,5),
c(8,7),c(9,8)))
colnames(data) = c("Y", "X")
str(data)

lm.YX <- lm(Y ~ X, data = data)
summary(lm.YX)
```

Extension to multivariate regression model: While this is the case for a simple regression model, we can extend this experience to a more general case:

$$y = \beta_0 + \sum_{i=1}^p \beta_i x_i + \varepsilon.$$

To fit this multivariate linear regression model, we collect n data points, denoted as

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{21} & \cdots & x_{p1} \\ 1 & x_{12} & x_{22} & \cdots & x_{p2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1N} & x_{2N} & \cdots & x_{pN} \end{bmatrix},$$

where $\mathbf{y} \in R^{N \times 1}$ denotes for the n measurements of the response variable, and $\mathbf{X} \in R^{N \times (p+1)}$ denotes for the design matrix that includes the N measurements of the p input variables.

Then, the regression model can be rewritten in its matrix form as:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

Here, $\boldsymbol{\beta} \in R^{(p+1) \times 1}$ denotes for the regression parameters and $\boldsymbol{\varepsilon} \in R^{N \times 1}$ denotes for the N residuals which are assumed to follow a normal distribution with mean as zero and variance as σ_{ε}^2 .

A detailed presentation of them is shown in below:

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}, \text{ and } \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{bmatrix}.$$

Then, to estimate $\boldsymbol{\beta}$, we can derive the optimization formulation in matrix form as:

$$\min_{\boldsymbol{\beta}} (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}),$$

To solve this optimization problem, we can take the gradient of the objective function and set it to be zero:

$$\frac{\partial (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0,$$

which gives rise to the equation:

$$\mathbf{X}^T (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) = 0.$$

This leads to the least square estimator of $\boldsymbol{\beta}$ as

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}.$$

A resemblance can be easily detected between $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$ with $\beta_1 = \frac{\text{cov}(x, y)}{\text{var}(x)}$ by noticing that $\mathbf{X}^T \mathbf{Y}$ (corresponds to $\text{cov}(x, y)$) reflects the correlation between predictors and output, and $\mathbf{X}^T \mathbf{X}$ (corresponds to $\text{var}(x)$) reflects the variability of the predictors.

Hypothesis testing of regression parameters: It is important to recognize that, since \mathbf{y} is a random vector and induce uncertainty, $\hat{\boldsymbol{\beta}}$ is a random vector as well. The mean of $\hat{\boldsymbol{\beta}}$ is $\boldsymbol{\beta}$, as

$$E(\hat{\boldsymbol{\beta}}) = E[(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}] = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T E[\mathbf{y}] = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \boldsymbol{\beta}.$$

While the covariance matrix of $\hat{\boldsymbol{\beta}}$ can be readily derived as

$$\text{cov}(\hat{\boldsymbol{\beta}}) = \sigma_\varepsilon^2 (\mathbf{X}^T \mathbf{X})^{-1}.$$

This result lays the foundation for developing hypothesis testing of the regression parameters.

For example, as a typical hypothesis testing question, let's say, the null hypothesis is

$$H_0: \beta_i = 0.$$

By theory, it is known that $\hat{\beta}_i \sim N\left(\beta_i, \frac{\sigma_\varepsilon^2}{\mathbf{x}_i^T \mathbf{x}_i}\right)$. Thus, if our null hypothesis is true, then, $\hat{\beta}_i \sim N\left(0, \frac{\sigma_\varepsilon^2}{\mathbf{x}_i^T \mathbf{x}_i}\right)$. This gives us the theoretical ground to make conjecture of what our estimate $\hat{\beta}_i$ is “supposed to be”, i.e., as shown below, $\hat{\beta}_i$ is supposed to come from a normal distribution with mean as zero and variance as $\frac{\sigma_\varepsilon^2}{\mathbf{x}_i^T \mathbf{x}_i}$ (in a specific application, $\frac{\sigma_\varepsilon^2}{\mathbf{x}_i^T \mathbf{x}_i}$ can be calculated and take a specific value):

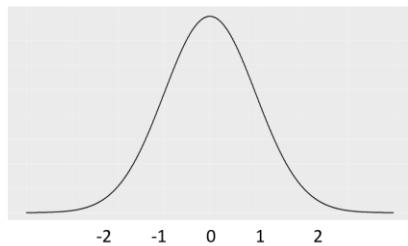


Figure 2.4: The distribution of $\hat{\beta}_i$

Based on this theory, we can see there is clearly a dominance of likelihood of what kind of $\hat{\beta}_i$ we can observe. We could define a range of $\hat{\beta}_i$ that we believe as plausible (i.e., if the null hypothesis is true, then it is normal to see this value of $\hat{\beta}_i$). Note that I use plausible in contrast with possible, since our theory tells us any value is always possible, but the possibility is not equally distributed among all the values as shown in the Figure 2.4. Also, our common sense tells us that some extreme values are always suspicious, pointing to rare chance. We may define a level of probability that represents our threshold of rare chance. We coin this threshold level as α .

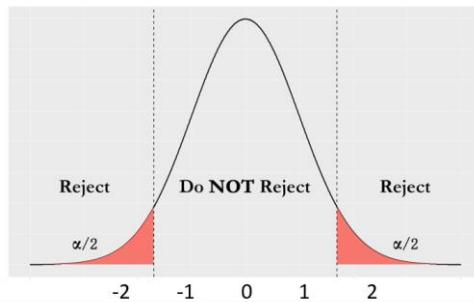


Figure 2.5: The framework of hypothesis testing

Now we have almost established the framework of hypothesis testing for regression parameters. With the threshold level α , we have *made a decision* that we will conclude that any value of $\hat{\beta}_i$ that falls into either side of the two extreme regions is unlikely – if the null hypothesis is true. Thus, if we see value in either side of the two extreme regions, we will reject the null hypothesis, since this indicates a strong conflict between theory (things suppose to be) and our empirical evidence (as what we observed on $\hat{\beta}_i$). This framework is shown in Figure 2.5.

Of course, we are conscious that we make a decision with a risk. We may be wrong, since even if the null hypothesis is true, there is still a small probability, α , that we may observe the $\hat{\beta}_i$ falls into either side of the two extreme regions. But we have accepted this risk. This risk is called the **Type 1 Error**.

II.3 R Lab

In this section, we illustrate step-by-step R codes to show how the linear regression model can be used in real-world data analysis. A distinct feature of this illustration lies on the “real-worldliness” of the data that embodies both statistical regularities (such that this analysis is enabled and called for) and realistic irregularities (such that we may recall the famous saying of Prof. George Box – “all models are wrong, some are useful”). Making informed decisions by drawing from rigorous theories, while at the same time, maintaining a critical attitude of theory, should both present simultaneously in practices of data analytics.

Here, our data is from a study of Alzheimer’s disease that collected demographics information and some genetic variables from hundreds of subjects. The goal of this dataset is to use these predictors to predict the score called Mini-Mental State Examination (**MMSCORE**) which is a clinical score (from 0-30) for determining Alzheimer’s disease, i.e., a **MMSCORE** of 20

to 24 suggests mild dementia, 13 to 20 suggests moderate dementia, and less than 12 indicates severe dementia.

First, let's load the data into the R workshop:

```
##### Example: Alzheimer's Disease
# filename
setwd("../analytics/data")
AD <- read.csv('AD_b1.csv', header = TRUE)
AD$ID = c(1:dim(AD)[1])
```

It is a nice habit to make detailed documentations of the variables with R using comments:

```
# Description of variables
# ID ID of the subjects
# Age Age
# PTGENDER Gender
# PTEDUCAT Years of education
# FDG Average FDG-PET
# AV45 Average AV45 SUVR
# HippoNV The normalized hippocampus volume
# e2_1 Apolipoprotein E4 polymorphism
# e4_1 Apolipoprotein E4 polymorphism
# rs3818361 CR1 gene rs3818361 polymorphism
# rs744373 BIN1 gene rs744373 polymorphism
# rs11136000 Clusterin CLU gene rs11136000 polymorphism
# rs610932 MS4A6A gene rs610932 polymorphism
# rs3851179 PICALM gene rs3851179 polymorphism
# rs3764650 ABCA7 gene rs3764650 polymorphism
# rs3865444 CD33 gene rs3865444 polymorphism
# MMSCORE Mini-mental state examination (outcome variable)
# TOTAL13 Alzheimer's Disease Assessment Scale (outcome variable)
```

After loading the data into the R workshop, we could use the `str()` function to give a sketchy overview of the data:

```
str(AD)

## 'data.frame':   517 obs. of  5 variables:
## $ MMSCORE : int  26 30 30 28 29 30 30 27 28 30 ...
## $ AGE     : num  71.7 77.7 72.8 69.6 70.9 65.1 79.6 73.6 60.7
## $ PTGENDER: int  2 1 2 1 1 2 2 2 1 2 ...
## $ PTEDUCAT: int  14 18 18 13 13 20 20 18 19 18 ...
## $ ID      : int  1 2 3 4 5 6 7 8 9 10 ...
```

First, let's build a regression model that only uses demographics variables. Demographics variables are usually the most accessible information of patients which we can use to build prediction models.

We can create a subset of the dataset as:

```
# subset of variables we want in our first model that only uses demographics predictors  
AD_demo <- subset(AD, select=c("MMSCORE", "AGE", "PTGENDER", "PTEDUCAT", "ID"))
```

Before building the model, by the spirit of **exploratory data analysis** (EDA)¹, we may draw the scatterplots to see how potentially the predictors can predict the outcome:

```
# ggplot: Plot the scatterplot of the data  
# install.packages("ggplot2")  
library(ggplot2)  
  
p <- ggplot(AD_demo, aes(x = PTEDUCAT, y = MMSCORE))  
p <- p + geom_point(size=2)  
# p <- p + geom_smooth(method = "auto")  
p <- p + labs(title="MMSE versus PTEDUCAT")  
print(p)
```

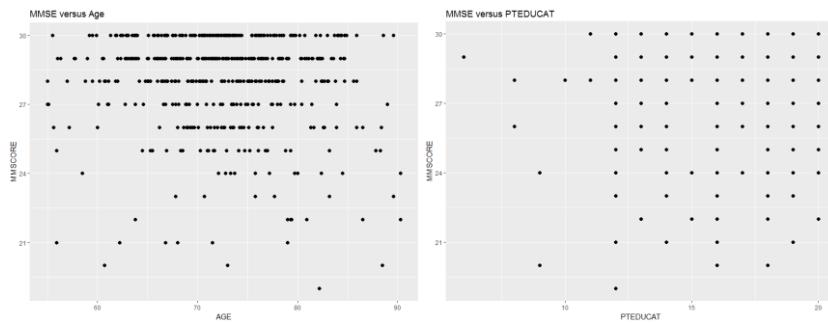


Figure 2.6: Scatterplots of (left) **MMSCORE** versus **AGE** and (right) **MMSE** versus **PTEDUCAT**

¹ John W. Tukey is a statistician whose career is known to be a strong advocate of EDA. See his book: Exploratory data analysis in 1977.

The scatterplots are shown in Figure 2.6. They show that there are weak relationships between the predictors with the **MMSCORE**, while still the relationship seems to be significant.

Then, we can use the **lm()** function to fit the regression model

```
# fit a simple linear regression model with only AGE
lm.AD_demo <- lm(MMSCORE ~ AGE, data = AD_demo)
# use summary() to get t-tests of parameters (slope, intercept)
summary(lm.AD_demo)

##
## Call:
## lm(formula = MMSCORE ~ AGE, data = AD_demo)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -8.7020 -0.9653  0.6948  1.6182  2.5447 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 30.44147   0.94564  32.191 <2e-16 ***
## AGE        -0.03333   0.01296  -2.572  0.0104 *  
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 2.11 on 515 degrees of freedom
## Multiple R-squared:  0.01268,    Adjusted R-squared:  0.01076 
## F-statistic: 6.614 on 1 and 515 DF,  p-value: 0.0104
```

Some important details could be read from the results shown above. First, it can be seen that the predictor, **AGE**, is significant with **p-value** as **0.0104** by the hypothesis testing procedure we delineated in Section II.2. It also seems that, the effect of this predictor, comparing with the noise, is rather weak, as the **R-squared** is only **0.01268**, suggesting that only 1.2% of the variability in **MMSCORE** could be explained by **AGE** alone.

To increase the **R-squared**, now let's include all the demographics variables into the model:

```
# fit the multiple Linear regression model with more than one predictor
lm.AD_demo2 <- lm(MMSCORE ~ AGE + PTGENDER + PTEDUCAT, data = AD_demo)
summary(lm.AD_demo2)
```

```

## 
## Call:
## lm(formula = MMSCORE ~ AGE + PTGENDER + PTEDUCAT, data = AD_de
## mo)
##
## Residuals:
##      Min     1Q Median     3Q    Max
## -8.4290 -0.9766  0.5796  1.4252  3.4539
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 27.70377  1.11131 24.929 < 2e-16 ***
## AGE         -0.02453  0.01282 -1.913  0.0563 .  
## PTGENDER    -0.43356  0.18740 -2.314  0.0211 *  
## PTEDUCAT    0.17120  0.03432  4.988 8.35e-07 *** 
## --- 
## Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.062 on 513 degrees of freedom
## Multiple R-squared:  0.0612, Adjusted R-squared:  0.05571 
## F-statistic: 11.15 on 3 and 513 DF,  p-value: 4.245e-07

```

From the results shown above we can see that, the predictor **AGE** is now on the boardline of significance with a p-value as **0.0563**. The other predictors, **PTGENDER** and **PTEDUCAT**, are significant. It also seems that the R-squared now increases from **0.01268** to **0.0612**, suggesting that 6.12% of the variability in **MMSCORE** could be explained by the three variables. The reason that the predictor **AGE** is now no longer significant is an interesting phenomena, but it is not unusual that a significant predictor becomes insignificant when other variables are included or excluded. This is because of the statistical dependence of the estimation of the predictors. As we have known that

$$\text{cov}(\hat{\beta}) = \sigma_{\varepsilon}^2 (\mathbf{X}^T \mathbf{X})^{-1}.$$

As long as $\mathbf{X}^T \mathbf{X}$ is not an identify matrix, the estimators of the regression parameters are dependent in a complicated and data-driven way. Due to this reason, we need to be very cautious about how to interpret the regression parameters as they are actually interrelated and also depend on the modeling process.

Having said that, regression model is still a useful approach to provide prediction power and insights about the data. Now let's build a full model with all the demographics, genetics, and imaging variables to predict **MMSCORE**.

```
# fit a full-scale model
AD_full <- AD[,c(1:16)]
lm.AD <- lm(MMSCORE ~ ., data = AD_full)

summary(lm.AD)

##
## Call:
## lm(formula = MMSCORE ~ ., data = AD_full)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -6.3201 -1.0265  0.2765  1.1977  4.1463 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 18.09118   1.69735 10.659 < 2e-16 ***
## AGE          0.01365   0.01197  1.140 0.254794    
## PTGENDER     -0.13146   0.16356 -0.804 0.421944    
## PTEDUCAT     0.16020   0.02947  5.436 8.53e-08 ***
## FDG           0.86143   0.13368  6.444 2.74e-10 ***
## AV45          -1.55526   0.42909 -3.625 0.000319 *** 
## HippoNV       7.27789   1.20610  6.034 3.11e-09 *** 
## e2_1          -0.03103   0.27459 -0.113 0.910068    
## e4_1          -0.18525   0.17651 -1.049 0.294456    
## rs3818361     0.18737   0.16373  1.144 0.253007    
## rs744373     -0.30165   0.15576 -1.937 0.053359 .  
## rs11136000    -0.03018   0.16423 -0.184 0.854257    
## rs610932      -0.34879   0.16208 -2.152 0.031872 *  
## rs3851179     0.05742   0.15675  0.366 0.714276    
## rs3764650     0.31522   0.19691  1.601 0.110049    
## rs3865444     -0.38589   0.15474 -2.494 0.012960 *  
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.744 on 501 degrees of freedom
## Multiple R-squared:  0.3443, Adjusted R-squared:  0.3246 
## F-statistic: 17.54 on 15 and 501 DF,  p-value: < 2.2e-16
```

From the results shown above we can see that, the **PTEDUCAT**, **FDG**, **AV45**, **HippoNV**, **rs610932**, and **rs3865444**, are significant. It also seems that the **R-**

squared now increases from **0.0612** to **0.3443**, suggesting that now 33.43% of the variability in **MMSCORE** could be explained by the variables.

We also notice that there are many variables showing insignificant p-values. Thus, we may conduct a **feature selection** procedure to delete the insignificant variables from the model. Roughly speaking, there are two approaches. One is called **stepwise backward** that begins with a full model, and sequentially remove variables. Another approach is called the **stepwise forward**, that begins with a one-predictor model and then sequentially adds variables into the model. Here, let's use the stepwise backward method for an example to show how it works.

The first variable to be removed from the model is the one that has the largest p-value (thus least significant).

```
# Do we need all the variables?
# remove e2_1, as it is Least significant
lm.AD.reduced <- lm.AD;
lm.AD.reduced <- update(lm.AD.reduced, ~ . - e2_1);
summary(lm.AD.reduced);

##
## Call:
## lm(formula = MMSCORE ~ AGE + PTGENDER + PTEDUCAT + FDG + AV45
+ 
##     HippoNV + e4_1 + rs3818361 + rs744373 + rs11136000 + rs610
932 +
##     rs3851179 + rs3764650 + rs3865444, data = AD_full)
##
## Residuals:
##      Min    1Q Median    3Q   Max
## -6.3189 -1.0216  0.2807  1.2016  4.1466
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 18.08148   1.69351 10.677 < 2e-16 ***
## AGE          0.01367   0.01196  1.143 0.253499    
## PTGENDER     -0.13191   0.16335 -0.808 0.419758    
## PTEDUCAT     0.16022   0.02944  5.442 8.25e-08 ***
## FDG           0.86185   0.13350  6.456 2.54e-10 ***
## AV45         -1.55316   0.42826 -3.627 0.000316 ***
## HippoNV       7.27258   1.20400  6.040 3.00e-09 ***
## e4_1          -0.18202   0.17401 -1.046 0.296053    
## rs3818361     0.18809   0.16345  1.151 0.250379    
## rs744373     -0.30116   0.15555 -1.936 0.053417 .
```

```

## rs11136000 -0.03037 0.16406 -0.185 0.853200
## rs610932 -0.34840 0.16188 -2.152 0.031854 *
## rs3851179 0.05936 0.15565 0.381 0.703078
## rs3764650 0.31553 0.19670 1.604 0.109322
## rs3865444 -0.38599 0.15459 -2.497 0.012848 *
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.742 on 502 degrees of freedom
## Multiple R-squared: 0.3443, Adjusted R-squared: 0.326
## F-statistic: 18.82 on 14 and 502 DF, p-value: < 2.2e-16

```

It can be seen that the R-squared is not affected. To formally draw a conclusion, we can compare the full model with this new model by `F-test` that is implemented in `anova()`:

```

anova(lm.AD.reduced, lm.AD)

## Analysis of Variance Table
##
## Model 1: MMSCORE ~ AGE + PTGENDER + PTEDUCAT + FDG + AV45 + Hi
## ppoNV +
##      e4_1 + rs3818361 + rs744373 + rs11136000 + rs610932 + rs38
## 51179 +
##      rs3764650 + rs3865444
## Model 2: MMSCORE ~ AGE + PTGENDER + PTEDUCAT + FDG + AV45 + Hi
## ppoNV +
##      e2_1 + e4_1 + rs3818361 + rs744373 + rs11136000 + rs610932
## +
##      rs3851179 + rs3764650 + rs3865444
##   Res.Df   RSS Df Sum of Sq    F Pr(>F)
## 1     502 1523.2
## 2     501 1523.1  1  0.038826 0.0128 0.9101

```

And we can see that it is statistically indistinguishable between the two models by the `F-test`, with `p-value` as `0.9101`.

We then move forward to delete the latest least significant predictor, `rs11136000`:

```

lm.AD.reduced <- update(lm.AD.reduced, ~ . - rs11136000);
summary(lm.AD.reduced);

##
## Call:
## lm(formula = MMSCORE ~ AGE + PTGENDER + PTEDUCAT + FDG + AV45
## +

```

```

##      HippoNV + e4_1 + rs3818361 + rs744373 + rs610932 + rs38511
79 +
##      rs3764650 + rs3865444, data = AD_full)
##
## Residuals:
##      Min      1Q Median      3Q      Max
## -6.3315 -1.0138  0.2713  1.1929  4.1375
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.05037   1.68353 10.722 < 2e-16 ***
## AGE          0.01360   0.01194  1.139 0.255316
## PTGENDER    -0.13228   0.16318 -0.811 0.417982
## PTEDUCAT     0.16035   0.02941  5.453 7.78e-08 ***
## FDG          0.86249   0.13333  6.469 2.34e-10 ***
## AV45         -1.54367   0.42477 -3.634 0.000308 ***
## HippoNV      7.26894   1.20268  6.044 2.93e-09 ***
## e4_1         -0.18292   0.17377 -1.053 0.293003
## rs3818361    0.19161   0.16218  1.181 0.237973
## rs744373     -0.30130   0.15540 -1.939 0.053077 .
## rs610932     -0.34802   0.16171 -2.152 0.031863 *
## rs3851179    0.06092   0.15527  0.392 0.694989
## rs3764650    0.31577   0.19651  1.607 0.108700
## rs3865444    -0.38681   0.15437 -2.506 0.012536 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.74 on 503 degrees of freedom
## Multiple R-squared:  0.3442, Adjusted R-squared:  0.3273
## F-statistic: 20.31 on 13 and 503 DF,  p-value: < 2.2e-16

```

Again, we can compare the full model with this new model by F-test
`anova(lm.AD.reduced, lm.AD)`

```

## Analysis of Variance Table
##
## Model 1: MMSCORE ~ AGE + PTGENDER + PTEDUCAT + FDG + AV45 + HippoNV +
##           e4_1 + rs3818361 + rs744373 + rs610932 + rs3851179 + rs3764650 +
##           rs3865444
## Model 2: MMSCORE ~ AGE + PTGENDER + PTEDUCAT + FDG + AV45 + HippoNV +
##           e2_1 + e4_1 + rs3818361 + rs744373 + rs11136000 + rs610932 +
##           rs3851179 + rs3764650 + rs3865444
## Res.Df   RSS Df Sum of Sq   F Pr(>F)

```

```
## 1    503 1523.2
## 2    501 1523.1  2   0.14282 0.0235 0.9768
```

We can repeat this process, until no more variable could be deleted.

While this approach is simple and gives us great visibility of the model selection process, it is a tedious process. Automation of this process could be achieved by the function **step()**, as shown in below:

```
# Automatic model selection
lm.AD.F <- step(lm.AD, direction="backward", test="F")
```

Then we can obtain the final selected model as:

```
## Step: AIC=581.47
## MMSCORE ~ PTEDUCAT + FDG + AV45 + HippoNV + rs744373 + rs61093
2 +
##      rs3764650 + rs3865444
##
##              Df Sum of Sq    RSS     AIC F value    Pr(>F)
## <none>                 1537.5 581.47
## - rs3764650  1     7.513 1545.0 581.99  2.4824  0.115750
## - rs744373  1    12.119 1549.6 583.53  4.0040  0.045924 *
## - rs610932  1    14.052 1551.6 584.17  4.6429  0.031652 *
## - rs3865444 1    21.371 1558.9 586.61  7.0612  0.008125 **
## - AV45      1    50.118 1587.6 596.05 16.5591 5.467e-05 ***
## - PTEDUCAT  1    82.478 1620.0 606.49 27.2507 2.610e-07 ***
## - HippoNV   1   118.599 1656.1 617.89 39.1854 8.206e-10 ***
## - FDG       1   143.852 1681.4 625.71 47.5288 1.614e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It can be seen that the predictors kept in the final model are all significant. Also, the R-squared is 0.3381 using the 8 predictors. This is not bad comparing with the R-squared as 0.3443 by using all the 15 predictors. Again, we can compare the full model with this final model by F-test, which shows that the it is statistically indistinguishable between the two models by the F-test, with p-value as 0.6913.

```
anova(lm.AD.F, lm.AD)

## Analysis of Variance Table
##
## Model 1: MMSCORE ~ PTEDUCAT + FDG + AV45 + HippoNV + rs744373
+ rs610932 +
```

```

##      rs3764650 + rs3865444
## Model 2: MMSCORE ~ AGE + PTGENDER + PTEDUCAT + FDG + AV45 + Hi
ppoNV +
##      e2_1 + e4_1 + rs3818361 + rs744373 + rs11136000 + rs610932
+
##      rs3851179 + rs3764650 + rs3865444
##   Res.Df   RSS Df Sum of Sq    F Pr(>F)
## 1     508 1537.5
## 2     501 1523.1  7    14.414 0.6773 0.6913

```

II.4 Remarks

The regression model is a very useful model, while at the same time, it is a model that demands a great amount of meticulous analysis and justification. It also a model that is frequently reported to be a troublemaker for confusion or misinterpretation, while a common misinterpretation is to treat the *statistical significance* of a predictor as a *causal significance* in the application context. Indeed, it is tempting to ignore the statistical nature of the regression model and treat it as a causal model given its asymmetric form (i.e., predictors are in one side of equation while outcome is in the other side). As we have seen in our example, some variables showing significance may disappear when other variables are added into the model, providing empirical evidences that the regression model is essentially a hypothesized model, whose statistical validity relies on its goodness-of-fit on the data.

A model fits the data well and passes the significance test only means that in data there is nothing significant against the model, but this fit process doesn't mean we found in data that this model is the only causal model that excludes the possibility of other models.

Related to this, as we briefly mentioned before, the design of experiment (DOE) is a discipline which tries to provide systematic data collection procedure to render the regression model as a causal inference model. How this could be done demands a lengthy discussion and illustration. Here, we briefly review its foundation to see why it has the connection with linear regression model.

It can be seen that, the uncertainty of $\hat{\beta}$ mainly comes from two sources, the variability from the data that is encoded in σ_ε^2 , and the structure of \mathbf{X} . The noise encoded in σ_ε^2 reflects essential uncertainty inherent in the system or the measurement device that generates the data. But \mathbf{X} is how we collect the data at what data points. Thus, many experimental design methods seek to optimize the structure of \mathbf{X} . We can try different cases of the matrix \mathbf{X} to see the implication of this result.

For example, let's consider the following \mathbf{X} :

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

It can be seen that, given this structure, the variance of $\hat{\beta}$ takes a very special structure:

$$\text{cov}(\hat{\beta}) = \sigma_\varepsilon^2 \mathbf{I}.$$

In other words, we can draw two main observations. First, the estimations of the regression parameters are independent, given that their correlations are zero. Second, the variances of the regression parameters are the same. From these two traits, this is an ideal structure for the data matrix \mathbf{X} that is commonly adopted for design of experiments when we have control over what data points we could collect. On the other hand, the data matrix \mathbf{X} most often takes an arbitrary structure that results in a general form for $\text{cov}(\hat{\beta})$. Thus, estimations of the regression parameters are often correlated with each other. Adding or deleting variables from the regression model will result in changes of the estimations of other parameters, calling for cautions from us to interpret the regression models properly.

Another interesting remark we'd like to point it out is that, in regression models, it is often the case that the interactions of the predictors could contribute extra prediction power. For example, to predict the yield of a chemical production process using the predictors, Temperature and

Catalyst concentration, it is likely that we need to include the main effects of both predictors, but also their interactions. But, generally speaking, it is challenging in practice to recognize that there are important interaction terms to be included in the model. Thus, we need both contextual knowledge and meticulous craftwork of analytics to play with the data and interrogate the data.

Here we provide an exemplary illustration of how to play with the data using EDA to discover interactions among variables. Thinking of the interaction between two variables, let's say, X_1 and X_2 , it essentially suggests that the relationship between X_1 (or X_2) and the outcome Y depends on what value the other variable X_2 (or X_1) takes. Thus, this gives us an insight that, as we can use scatterplot to visualize the relationship between any variable with the outcome, we could see how this relationship changes according to another variable.

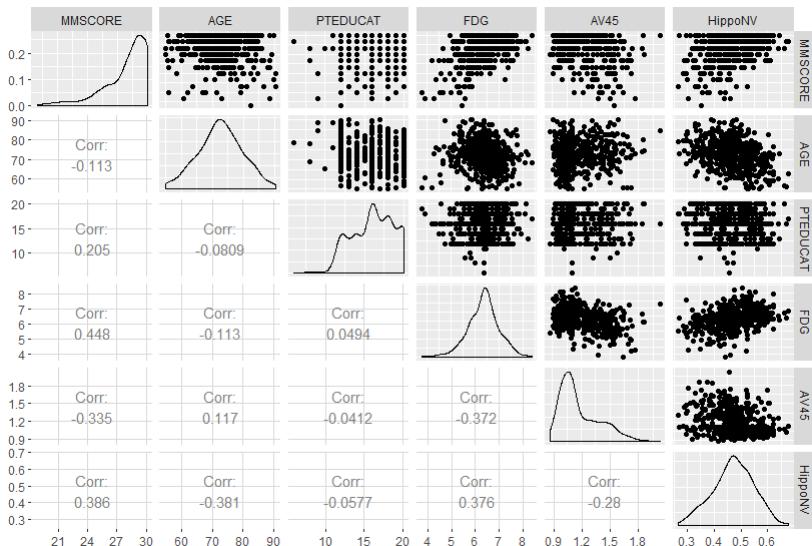


Figure 2.7: Scatterplots of the continuous predictors versus outcome variable

To implement this idea, first, let's use the AD dataset, and draw the scatterplot of the predictors as shown in Figure 2.7.

```
# Supplement the model with some visualization of the statistical patterns
# Scatterplot matrix to visualize the relationship between outcome variable with continuous predictors
library(ggplot2)
# install.packages("GGally")
library(GGally)

# draw the scatterplots and also empirical shapes of the distributions of the variables
p <- ggpairs(AD[,c(16,1,3,4,5,6)], upper = list(continuous = "points")
             , lower = list(continuous = "cor"))
)
print(p)
```

For the other predictors which are binary, we can use boxplot, which is shown in Figure 2.8.

```
# Boxplot to visualize the relationship between outcome variable with categorical predictors
library(ggplot2)
qplot(factor(PTGENDER), MMSCORE, data = AD,
      geom=c("boxplot"), fill = factor(PTGENDER))
```

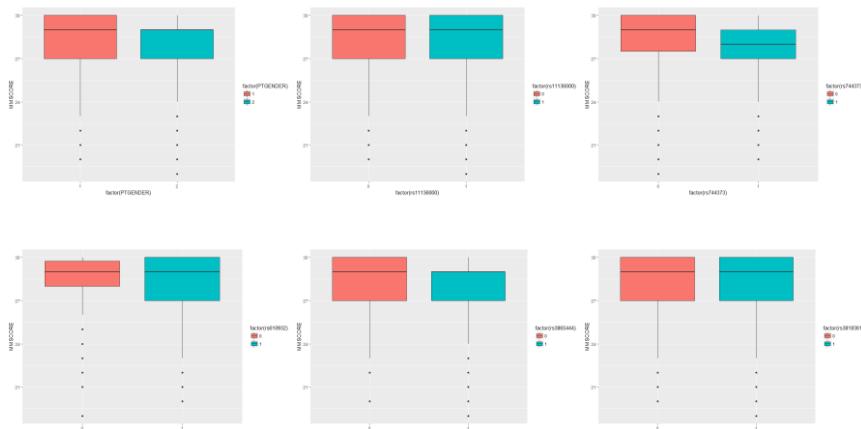


Figure 2.8: Boxplots of the binary predictors versus outcome variable

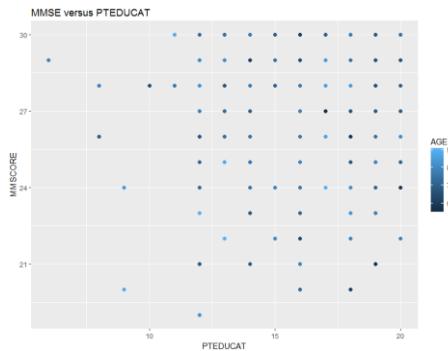


Figure 2.9: Scatterplots of PTEDUCAT versus MMSE

Now, let's pick up the scatterplot of `MMSCORE` versus `PTEDUCAT`, and see if the predictor, `AGE`, mediates the relationship between `MMSCORE` and `PTEDUCAT`. We then color the data points in the scatterplot while the color corresponds to the numerical scale of `AGE`. The following R codes generate Figures 2.9 and 2.10.

```
# How to detect interaction terms by exploratory data analysis (EDA)
require(ggplot2)
p <- ggplot(AD_demo, aes(x = PTEDUCAT, y = MMSCORE))
p <- p + geom_point(aes(colour=AGE), size=2)
# p <- p + geom_smooth(method = "auto")
p <- p + labs(title="MMSE versus PTEDUCAT")
print(p)
```

It looks like that the relationship between `MMSCORE` and `PTEDUCAT` indeed changes according to different levels of `AGE`. Thus, we draw the same scatterplot on two levels of `AGE`, $AGE < 60$ and $AGE > 80$.

```
p <- ggplot(AD_demo[which(AD_demo$AGE < 60),], aes(x = PTEDUCAT,
y = MMSCORE))
p <- p + geom_point(size=2)
p <- p + geom_smooth(method = lm)
p <- p + labs(title="MMSE versus PTEDUCAT when AGE < 60")
print(p)

p <- ggplot(AD_demo[which(AD_demo$AGE > 80),], aes(x = PTEDUCAT,
y = MMSCORE))
p <- p + geom_point(size=2)
p <- p + geom_smooth(method = lm)
```

```
p <- p + labs(title="MMSE versus PTEDUCAT when AGE > 80")
print(p)
```

Then, we can obtain Figure 2.10.

Obviously, an interesting phenomenon emerges and shows that the relationship between `MMSCORE` and `PTEDUCAT` changes dramatically according to different levels of `AGE`!

Thus, we further add this interaction term into the model that uses all the demographics variables:

```
# fit the multiple Linear regression model with an interaction term
# rm: AGE*PTEDUCAT
lm.AD_demo2 <- lm(MMSCORE ~ AGE + PTGENDER + PTEDUCAT + AGE*PTEDUCAT,
  data = AD_demo)
summary(lm.AD_demo2)
```

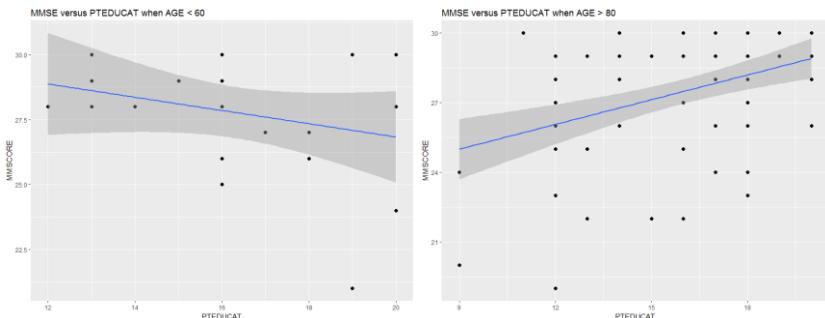


Figure 2.10: Scatterplots of `PTEDUCAT` versus `MMSCORE` when (left) `AGE < 60` or (right) `AGE > 80`

Then, we can see that this interaction term is significantly contributing extra prediction power on top of the existing predictors (`p-value` is `0.01534`)!

```
## 
## Call:
## lm(formula = MMSCORE ~ AGE + PTGENDER + PTEDUCAT + AGE * PTEDUCAT,
##     data = AD_demo)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1.0000 -0.4000 -0.1000  0.3000  1.0000
```

```

## -8.2571 -0.9204  0.5156  1.4219  4.2975
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept) 40.809411  5.500441  7.419 4.93e-13 ***
## AGE         -0.202043  0.074087 -2.727  0.00661 **
## PTGENDER    -0.470951  0.187143 -2.517  0.01216 *
## PTEDUCAT   -0.642352  0.336212 -1.911  0.05662 .
## AGE:PTEDUCAT 0.011083  0.004557  2.432  0.01534 *
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.052 on 512 degrees of freedom
## Multiple R-squared:  0.07193,    Adjusted R-squared:  0.06468
## F-statistic:  9.92 on 4 and 512 DF,  p-value: 9.748e-08

```

III. Tree Models

III.1 Rationale and Formulation

While the linear regression model is a typical data modeling method, the decision tree model represents a typical method in the category of algorithmic modeling as discussed in Table 1. The linear regression model formalizes the data generating mechanism, which emphasizes an understanding of the underlying system. In contrast, the decision tree mimics heuristics in human reasoning. An exemplary tree model is shown in Figure 2.11, which uses weather and day of week (Dow) (as two predictors) to predict whether to play basketball (as outcome variable).

As shown in Figure 2.11, a decision tree contains of one root node (highlighted as yellow), inner nodes (highlighted as yellow), and decision nodes (highlighted as green). It also has splitting rules that are specified alongside the arcs. To predict on a data point, i.e., \mathbf{x}_i , the root node is where to begin with. The data point will travel along the inner nodes according to the rules specified alongside the arcs. For example, considering the tree model in Figure 2.11. If $\mathbf{x}_i = \{\text{Weather} = \text{Sunny}, \text{Dow} = \text{Yes}\}$, then we can see that the data point will first go to inner node 1, and then, go to the left decision node and reach the decision that Play = Yes. If $\mathbf{x}_i = \{\text{Weather} \neq \text{Sunny}, \text{Dow} = \text{Yes}\}$, then we can see that the data point

will go to the decision node right from the root node, and reach the decision that Play = No.

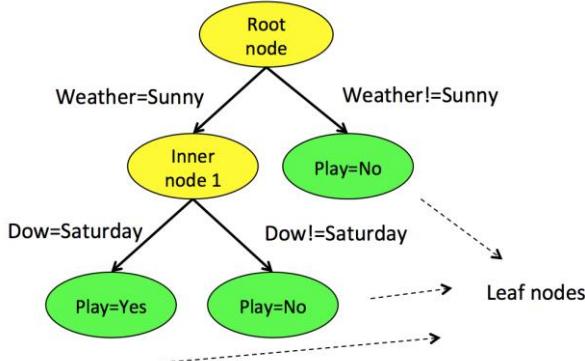


Figure 2.11: An exemplary decision tree model

Table 2.3: An exemplary dataset for building a decision tree

ID	Weather	Dow (day-of-week)	Play
1	Rainy	Saturday	No
2	Sunny	Saturday	Yes
3	Windy	Tuesday	No
4	Sunny	Saturday	Yes
5	Sunny	Monday	No
6	Windy	Saturday	No

III.2 Theory/Method

The decision tree shown in Figure 2.11 is created by domain knowledge. In what follows, we introduce how to create such a decision tree using data. As we can see from Figure 2.11, the key elements of a decision tree is the splitting rules that can lead us through the inner nodes to the final decision node to reach a decision. A splitting rule is defined by a variable and the set

of values it belongs to, e.g., Weather = Sunny. The variable used for splitting is referred as the splitting variable, and the corresponding value is referred as the splitting value.

Pretending that we don't have the domain knowledge to create the tree model in Figure 2.11, let's consider the dataset in Table 2.3 to build a decision tree. The dataset has 6 samples, while each sample was collected empirically by observing the decision of a basketball team.

To build a decision tree model, we now face the first question, which is, in the root node, which variable should we use to define the first splitting rule. Possible splitting rules are {Weather = Rainy, Weather = Sunny, Dow = Saturday, Dow = Monday, Dow = Tuesday}. If we use the splitting rule, Weather = Sunny, it will result in the decision tree as shown in the left figure of Figure 2.12. If we use the splitting rule, Dow = Saturday, it will result in the decision tree as shown in the right figure of Figure 2.22. Which one should we use?

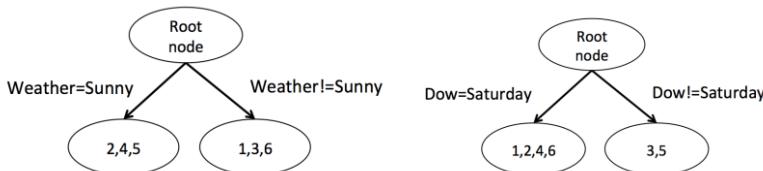


Figure 2.12: The decision tree models with Weather (left tree) or Dow (right tree) as the splitting variable in the root node

As we can see from Figure 2.12, once there is a set of splitting rule candidates, one of them needs to be selected at a node for splitting. To help us decide on which splitting rule is the best, the concept of **impurity** of data has been developed.

Impurity and information gain (IG): For classification problem (the outcome variable is categorical), the impurity of data points in a given node can be measured by **entropy**:

$$e = \sum_{i=1,\dots,K} -P_i \log_2 P_i.$$

where K represents the number of classes and P_i is the proportion of data points in the node that belong to the class i . The entropy e is defined as 0 when the data points in the node all belong to one single class.

It is easy to see that, a node that has a large impurity is not ready to be a decision node yet. Thus, if we want to further split the node and create two more child nodes under it, we look for the best splitting rules that can minimize the impurity of the children nodes. This reduction of impurity can then be measured by **information gain (IG)**, which is the difference of the entropy of the splitting node and the average entropy of the two children nodes weighted by their number of data points. The IG is defined as:

$$IG = e_s - \sum_{i=1,\dots,n} w_i * e_i.$$

Here, e_s is the entropy at the the splitting node, e_i is the entropy at the child node i , and w_i is the number of data points in the children node i divided by the number of data points at the splitting node¹.

To show how these concepts could be implemented, let's look at the decision tree model in the left figure of Figure 2.12. The entropy of the root node is calculated as

$$-\frac{4}{6} \log_2 \frac{4}{6} - \frac{2}{6} \log_2 \frac{2}{6} = 0.92.$$

The entropy of the left child node (“Weather = Sunny”) is

$$-\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3} = 0.92.$$

The entropy of the right child node (“Weather != Sunny”) is 0 since all the three data points (ID = 1,3,6) belong to the same class.

¹ For regression problems, the variance of the outcome variable can be used for measuring the impurity of a node, i.e.,

$$v = \sum_{n=1}^N (\bar{y} - y_n)^2,$$

where $y_{n=1,\dots,N}$ are the values of the outcome variable, and \bar{y} is the average of the outcome variable at the node. And the information gain can be calculated similarly to the classification problem.

The information gain for the splitting rule “Weather = Sunny” is then

$$IG = 0.92 - \frac{3}{6} * 0.92 - \frac{3}{6} * 0 = 0.46.$$

For the decision tree in the right figure of Figure 2.12, the entropy of the left child node (“Dow = Saturday”) is

$$-\frac{2}{4} \log_2 \frac{2}{4} - \frac{2}{4} \log_2 \frac{2}{4} = 1.$$

The entropy of the right child node (“Dow != Saturday”) is 0 since the two data points (ID = 3,5) belong to the same class.

Thus, the information gain for the splitting rule “Dow = Saturday” is then

$$IG = 0.92 - \frac{3}{6} * 1 - \frac{3}{6} * 0 = 0.42.$$

As the information gain of “Weather = Sunny” is higher, the splitting rule “Weather = Sunny” is preferred over its competitor “Dow = Saturday”.

Similarly, the information gain can be calculated for all other splitting rules. As “Weather = Sunny” has the maximum information gain, it is selected for splitting the root node. The left child node with data points (ID = 2,4,5) still has two classes, and can be further split by selecting the next best splitting rule. The right child node has only one class and becomes a decision node labeled with the decision “play = No”.

Note that, in our example, we only have categorical variables, while splitting rules could be easily defined. For continuously variables, the values of a variable are firstly ordered, and then, the average of each pair of consecutive values is used as the splitting value.

Greedy recursive approach to build a tree: Based on the concept of impurity and IG, we could develop a greedy strategy that recursively split the data points until there is no further IG, e.g., when there is only one data point, or only one class in the node. Most tree-building methds use this approach, with difference in the definitions of the impurity and IG.

However, this will inevitably lead to a very large decision tree with many inner nodes and unstable decision nodes (i.e., since the decisions assigned to these nodes are inferred empirically based on very few data points). Thus,

such a decision tree can be sensitive to noisy data, leading to worse accuracy and interpretability.

Tree pruning: To mitigate this problem, the pre-pruning or post-pruning methods can be used to control the complexity of a decision trees. Pre-pruning stops growing a tree when a pre-defined criterion is met. One can define the maximum depth of a tree, or minimum number of data points at each node. As these approaches are based on prior knowledge, they may not necessarily reflect the data characteristics. More data-dependent approaches can be used. For example, the minimum impurity gain threshold can be used to stop growing a tree when the impurity gain is below the threshold. Still, a small impurity gain at a node does not necessarily indicate equivalent small impurity gain from its children nodes. Therefore, pre-pruning can cause over-simplified and thus under-fitted tree models. In other words, it is too cautious.

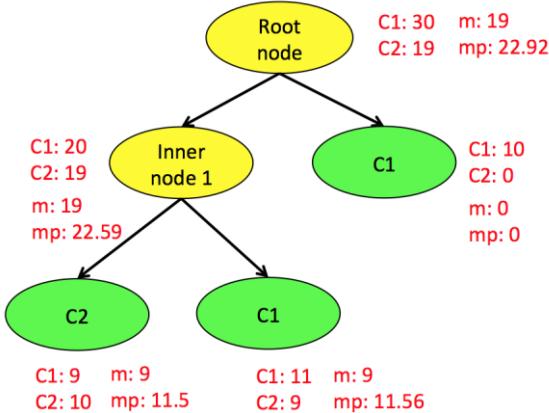


Figure 2.13: An example for tree pruning using pessimistic error

In contrast, post-pruning prunes a tree after it is fully-grown and has less risk of under-fit as a full-grown tree contains sufficient information captured from the data. Post-pruning starts from the bottom of the tree. If removing the sub-tree of an inner node does not increase the error, then

the sub-tree under the inner node can be pruned. Note that, the error here is not the error calculated based on the **training data** that is used to train the tree. This error calculated based on training data is called as **empirical error**. Rather, here, we should use the **generalization error**¹, that is, the error when applied to unseen data. Thus, in the famous decision tree algorithm, C4.5, the **pessimistic error estimation** approach is used.

We can derive the pessimistic error estimation using binomial approximation. Denote the empirical error rate on the training data as \hat{e} , which is only an estimation of the generalization error e . Since each data point can be either correctly or wrongly classified, we can view the probability of being correctly classified as a binomial distribution with probability e . With this insight, the normal distribution approximation can be applied here to derive the confidence interval of the generalization error e as:

$$\hat{e} - z_{\alpha/2} \sqrt{\frac{\hat{e}(1-\hat{e})}{n}} \leq e \leq \hat{e} + z_{\alpha/2} \sqrt{\frac{\hat{e}(1-\hat{e})}{n}}.$$

The upper bound of the interval, $\hat{e} + z_{\alpha/2} \sqrt{\frac{\hat{e}(1-\hat{e})}{n}}$, is used as the estimate of e . As this is an upper bound, it is named as pessimistic error estimation. Note that, the estimate of the error depends on three values, α , which is often set to be 0.25 so that $z_{\alpha/2}=1.15$; \hat{e} , which is the training error; and n , which is the number of data points at the node. Therefore, the estimated error is larger with a smaller n , accounting for the sample size as well.

Considering the tree in Figure 2.13 as an example. Each decision node is labeled with a class (i.e., either C1 or C2). Besides each node, we also highlight the distribution of the two classes, the misclassified instances (m), and the misclassified instances using the pessimistic error estimation (mp).

¹ We will return to this issue with more delicate discussion in Chapter 5.

Consider the inner node 1. We can see that, if we prune the subtree below inner node 1, we will label it with class C1, as 20 of the included data points are labeled as C1 while 19 are labeled as C2. And we can get that the total misclassified instances m is 19. Thus, $\hat{e} = \frac{19}{39} = 0.4871$. For the pessimistic error estimation, we can get that

$$\hat{e} + z_{\alpha/2} \sqrt{\frac{\hat{e}(1-\hat{e})}{n}} = 0.4871 + 1.15 \sqrt{\frac{0.4871(1-0.4871)}{39}} = 0.579.$$

Thus, we can get that $mp = 0.579 \times 39 = 22.59$.

Now, let's see if the splitting of this inner node into its two child nodes improves the pessimistic error. It can be seen that, with this subtree of the two child nodes, the total misclassified instances m is $9+9=18$. And by pessimistic error estimation, $mp = 11.5 + 11.56 = 23.06$ for the subtree. Therefore, based on the pessimistic error, we should prune the subtree.

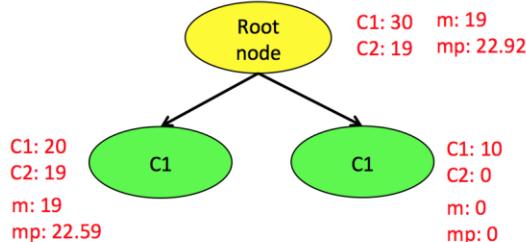


Figure 2.14: The pruned tree of Figure 2.13

Now the pruned tree is shown in Figure 2.14. Now consider whether to prune the children nodes of the root node. The pessimistic misclassified instances at the root node is 22.92, and the total pessimistic misclassified instances from its children nodes is $22.59+0=22.59$. Pruning the children node would lead to increased error, and thus, the children nodes are kept and the final tree consists of three nodes.

III.3 R Lab

Now let's use the AD dataset for illustrating how the decision tree model can be used. Here, we use `DX_b1` as the outcome variable that is binary ("0" denotes for normal subjects while "1" denotes for diseased subjects), and use other variables (except `ID`, `TOTAL13` and `MMSCORE`) to predict `DX_b1`.

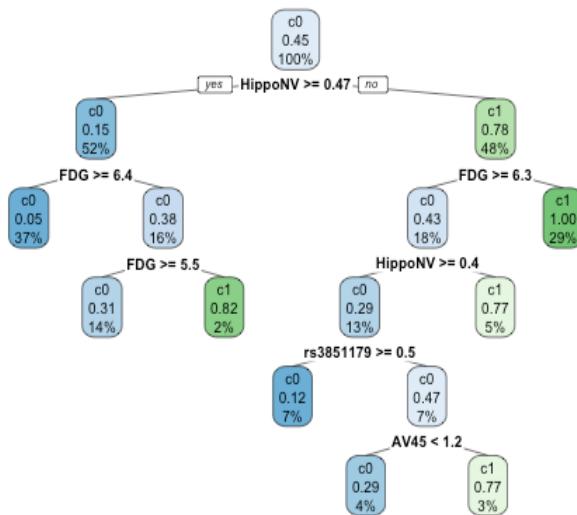


Figure 2.15: The unpruned decision tree model to predict `DX_b1` of the AD data

The R code in below loads the needed R packages and loads the data into the workspace.

```
library(rpart)
library(rpart.plot)
library(dplyr)
library(tidyr)
library(ggplot2)
library(partykit)
theme_set(theme_gray(base_size = 15))
```

```

path <- "../analytics/data/ AD_b1.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_b1")
data[, target_indx] <- as.factor(paste0("c", data[, target_indx]))
rm_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))
data <- data[, -rm_indx]

```

The `rpart()` function in the R package “`rpart`” can be used to build the decision tree using the data and plot the decision tree in Figure 2.15.

```

tree <- rpart(DX_b1 ~ ., data)
rpart.plot(tree, fallen.leaves = FALSE)

```

As an associated function with the `rpart` package, the importance score for each variable can be obtained from the tree. `HippoNV` has the largest importance score among all the variables.

```

print(tree$variable.importance)

##      HippoNV          FDG          AV45          AGE        PTGENDER
## 116.6665538  89.5608444  39.9595988  28.2195180  12.2040648
## 6.4708596
##    rs3851179      PTEDUCAT    rs3818361
##    4.2352941     1.1552265   0.8663915

```

The tree can be further pruned with the `prune` function whereas the parameter `cp` controls the model complexity. `cp` is the minimum relative error improved by splitting the node. A larger `cp` leads to a less-complex tree. First, let we try `cp = 0.05` which leads to Figure 2.16.

```

tree_0.05 <- prune(tree, cp = 0.05)
rpart.plot(tree_0.05, fallen.leaves = FALSE)

```

We can see that the tree is pruned. Then, we increase `cp` to `0.1` which leads to Figure 2.17.

```

tree_0.1 <- prune(tree, cp = 0.1)
rpart.plot(tree_0.1, fallen.leaves = FALSE)

```

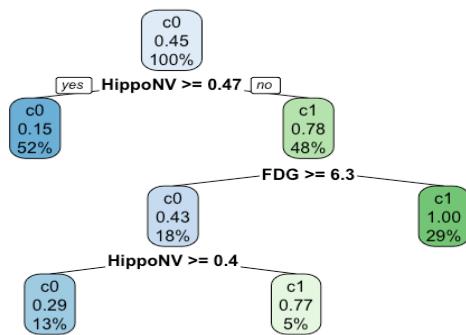


Figure 2.16: The pruned decision tree model to predict `DX_b1` of the AD data with `cp = 0.05`

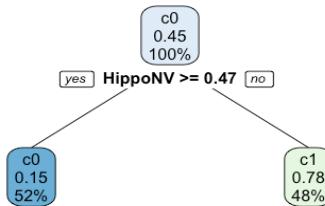


Figure 2.17: The pruned decision tree model to predict `DX_b1` of the AD data with `cp = 0.1`

We can see that, with `cp = 0.1`, the tree only has two nodes.

Now we have seen that the parameter `cp` could be used to control the model complexity in pruning. In practices, `cp` can be decided by minimizing the cross-validation error. The cross-validation will be introduced in detail in Chapter 5. Here, we take this opportunity to present it so we could get a sense of what it does. First, we split the data into halves, one half for training the tree model while another half for testing its accuracy. We then build a series of tree models using the training data with `cp` values ranging

from 0.2 to 0. For each built tree model, a training error and testing error can be calculated using the two datasets, respectively. For each tree, the number of decision nodes is recorded and used for measuring the complexity of the tree.

```
set.seed(1)
train.ix <- sample(nrow(data), floor(nrow(data)/2))
err.train.v <- NULL
err.test.v <- NULL
leaf.v <- NULL
for (i in seq(0.2, 0, by = -0.005)) {
  tree <- rpart(DX_b1 ~ ., data = data[train.ix, ], cp = i)
  pred.train <- predict(tree, data[train.ix, ], type = "class")
  pred.test <- predict(tree, data[-train.ix, ], type = "class")
  current.err.train <- length(which(pred.train != data[train.ix,
]$DX_b1))/length(pred.train)
  current.err.test <- length(which(pred.test != data[-train.ix,
]$DX_b1))/length(pred.test)
  err.train.v <- c(err.train.v, current.err.train)
  err.test.v <- c(err.test.v, current.err.test)
  leaf.v <- c(leaf.v, length(which(tree$frame$var == "<leaf>")))
}
err.mat <- as.data.frame(cbind(train_err = err.train.v, test_err
= err.test.v,
  leaf_num = leaf.v))
err.mat$leaf_num <- as.factor(err.mat$leaf_num)
err.mat <- unique(err.mat)
err.mat <- err.mat %>% gather(type, error, train_err, test_err)

data.plot <- err.mat %>% mutate(type = factor(type))
ggplot(data.plot, aes(x = leaf_num, y = error, shape = type, colo
r = type)) +
  geom_line() + geom_point(size = 3)
```

The training errors and testing errors of the trees at different number of decision nodes are plotted in Figure 2.18. It can be seen that, as the complexity of trees increases, the training errors continue to decrease, while the testing errors first decrease but increase at some point. This indicates that, there is an optimal tree size that can be identified by testing error but will be missed by training error. This is actually the danger of aggressively pursuing models that can achieve too-good-to-be-true performances on training data, a commonly known phenomenon as **overfitting**.

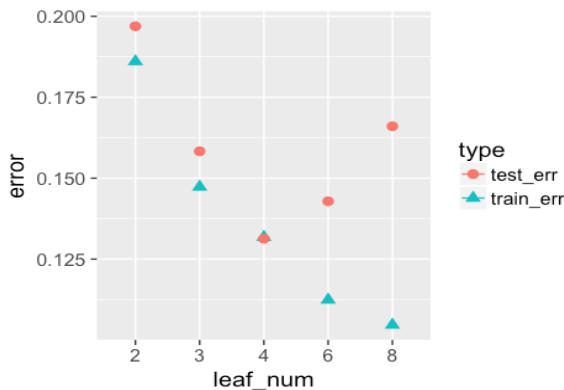


Figure 2.18: Training and testing errors versus complexity of the tree model (i.e., measured by the number of decision nodes in the tree)

III.4 Remarks

At each node of a decision tree, split is based on one single variable, and therefore, if the variable is continuous, the classification boundary for that split is perpendicular to the variable. And overall the classification boundary from a decision tree is always parallel or perpendicular to a continuous variable. This is illustrated in the following graph. The classification boundary consisting of splits by X_1 and X_2 is either parallel or perpendicular to one axis.

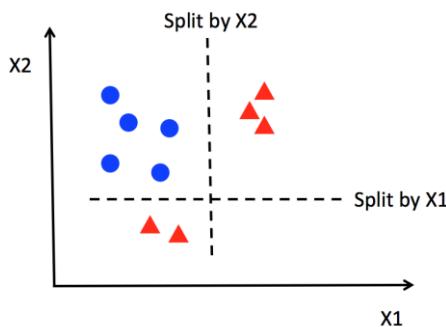


Figure 2.19: Decision boundary captured by tree models

This implies that, when applying a decision tree to a dataset with linear relationship between predictors and outcome variables, it may not be an optimal choice. In the following example, we simulate a data set and apply a decision tree and a logistics regression model (the counterpart of linear regression model for classification problem that will be introduced in Chapter 3) to the data, respectively. The training data, and the predicted classes for each data point from the logistic regression and decision models are shown in Figures 2.20, 2.21 and 2.22, respectively. It can be seen the classification boundary from the logistics regression model is linear, while the one from the decision tree is parallel to the axis. This limitation makes a decision tree not be able to fully capture the linear relationship in the data.

```

ndata <- 2000
X1 <- runif(ndata, min = 0, max = 1)
X2 <- runif(ndata, min = 0, max = 1)
data <- data.frame(X1,X2)
data <- data %>% mutate( X12 = 0.5 * (X1 - X2), Y = ifelse(X12>=0,
1,0) )
ix <- which( abs(data$X12) <= 0.05 )
data$Y[ix] <- ifelse(runif( length(ix)) < 0.5, 0, 1)
data <- data %>% select(-X12) %>% mutate( Y = as.factor(as.character(Y)) )
ggplot(data,aes(x=X1,y=X2,color=Y))+geom_point()

linear_model <- glm(Y ~ ., family = binomial(link = "logit"), data = data)
tree_model <- rpart( Y ~ ., data = data)
pred_linear <- predict(linear_model, data,type="response")
pred_tree <- predict(tree_model, data,type="prob")[,1]
data_pred <- data %>% mutate( pred_linear_class = ifelse( pred_linear <0.5,0,1) ) %>%
    mutate( pred_linear_class = as.factor(as.character(pred_linear_class)) ) %>%
    mutate( pred_tree_class = ifelse( pred_tree <0.5,0,1) )
%>%
    mutate( pred_tree_class = as.factor(as.character(pred_tree_class)) )
ggplot(data_pred,aes(x=X1,y=X2,color=pred_linear_class))+geom_point()

ggplot(data_pred,aes(x=X1,y=X2,color=pred_tree_class))+geom_point()

```

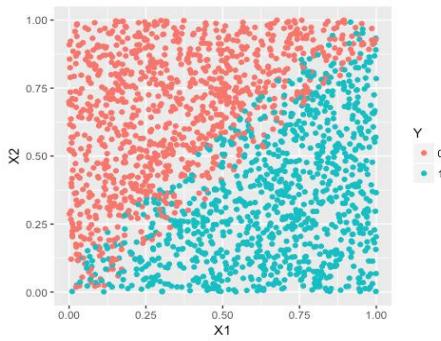


Figure 2.20: Scatterplot of the generated dataset

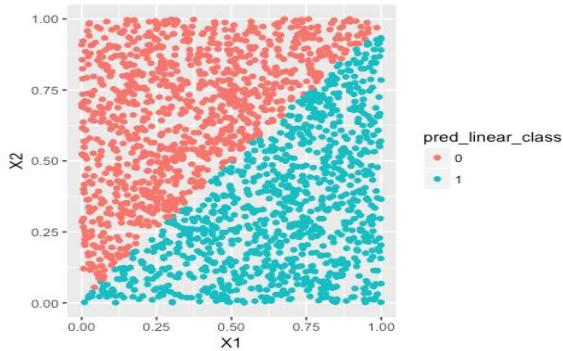


Figure 2.21: Decision boundary captured by logistic regression model

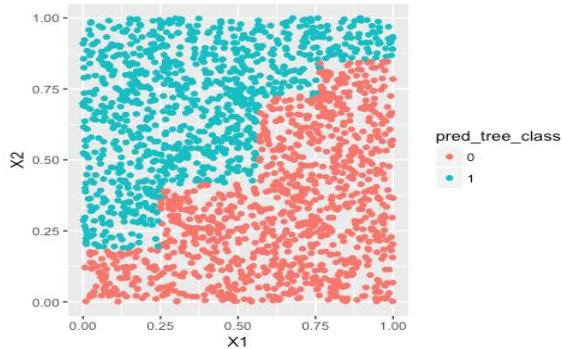


Figure 2.22: Decision boundary captured by the tree model

IV. Exercises

Data analysis

1. Repeat the analysis shown in the R lab of this chapter, but use **TOTAL13** as the outcome variable. Please build both the regression model and the decision tree model (for regression). Identify the final models you would select, evaluate the models, and compare the regression model with the tree model.
2. Find two datasets from the UCI data repository¹ or R datasets². Conduct a detailed regression analysis for both datasets using both regression model and the tree model (for regression), e.g., for regression model, you may want to conduct model selection, model comparison, testing of the significance of the regression parameters, evaluation of the R-squared and significance of the model. Also comment on the application of your model on the context of the dataset you have selected.
3. Pick up any dataset you have used, and randomly split the data into two halves. Use one half to build the tree model and the regression model. Test the models' prediction performances on the second half. Report what you have found, adjust your way of model building, and suggest a strategy to find the model you consider as the best.

Derivation

4. Consider the case that, in building linear regression models, there is a concern that some data points may be more important (or more trustable). Thus, it is not uncommon to assign a weight to each data point. Denote the weight for the i th data point as w_i . We still want to estimate the regression parameters in the least squares framework. Follow the process of the derivation of the least

¹ <http://archive.ics.uci.edu/ml/index.php>

² <https://vincentarelbundock.github.io/Rdatasets/datasets.html>

squares estimator and propose your new estimator of the regression parameters.

5. Build a decision tree model based on the following dataset. Don't use R. Use your pen and paper, and show the process.

Table 2.4: dataset for building a decision tree

ID	X1	X2	Y
1	0.22	0.38	No
2	0.58	0.32	Yes
3	0.57	0.28	Yes
4	0.41	0.43	Yes
5	0.6	0.29	No
6	0.12	0.32	Yes
7	0.25	0.32	Yes
8	0.32	0.38	No

Programming

6. Write your own R script to implement the least squares estimation of a regression model. Compare the output from your script with the output from `lm()`.
7. As a continuation of 1, also write your own R script to derive the p-value of your regression parameters. Compare the output from your script with the output from `lm()`.
8. Write your own R script to build a decision tree using the greedy recursive process mentioned in this chapter, using the information gain. By your script, the tree growth process stops when a given depth of the tree is reached.

CHAPTER 3: RECOGNITION

LOGISTIC REGRESSION AND

RANKING

I. Overview

Chapter 2 is about “**Recognition**”. This is a very important capability in real-world practices of analytics. It is a capability to recognize the same “abstracted” analytic problem embedded in seemly different real-world problems. This is not to say that all the real-world problems can be reduced to one single abstracted problem. A real-world problem usually contains multiple perspectives and layers, presenting itself as a combination or composition of multiple abstracted problems. Being able to recognize these abstracted problems holds the key to solve these real-world problems effectively. After all, to solve a real-world problem, at a certain point or a certain level, you have to bring the problem or part of the problem or a certain aspect of the problem into the territory of a classic analytic problem, because only in those classic territories we know we can solve problems for sure.

II. Logistic Regression Model

II.1 Rationale and Formulation

Linear regression model is introduced as a tool to predict a continuous response (or called outcome) variable y using a few input variables \mathbf{x} . In some applications, the response is a binary variable that denotes two classes. For example, in the AD dataset, we may wonder if we could use some variables to predict if the subject is a normal person or diseased.

We have learned about linear regression model to connect some input variables with the outcome variable. It is natural to wonder if and how the linear regression framework could still be useful here. Specifically, here, the input variables are still \mathbf{x} , but the outcome is not simply y . Rather, we may be more interested to predict probability $Pr(y = 1)$. Thus, we want to create probability by a function $p(\mathbf{x})$ such that $Pr(y = 1) = p(\mathbf{x})$. Following the mentality of linear regression, we somehow envision that the linear form, $\beta_0 + \sum_{i=1}^p \beta_i x_i$, should be used here as a constitutional component to define $p(\mathbf{x})$. It is hard to directly link $p(\mathbf{x}) = \beta_0 + \sum_{i=1}^p \beta_i x_i$ though, since $p(\mathbf{x})$ as a probability has to be in the range of $[0,1]$ but $\beta_0 + \sum_{i=1}^p \beta_i x_i$ has no limitation in the range. If we look closer into the idea of using a linear form to encode the predictive information in \mathbf{x} , we may realize that the linear form is very useful in ranking the possibilities rather than directly being eligible probabilities. In other words, a linear form is advantageous to make a comparison of two inputs, say, \mathbf{x}_i and \mathbf{x}_j , and evaluates which one leads to a higher probability of $Pr(y = 1)$. Thus, we don't have to let $p(\mathbf{x}) = \beta_0 + \sum_{i=1}^p \beta_i x_i$, but only need $p(\mathbf{x}) \propto \beta_0 + \sum_{i=1}^p \beta_i x_i$.

To fix this problem, statisticians soon found that it is better to link these two entities as:

$$\log \frac{p(\mathbf{x})}{1-p(\mathbf{x})} = \beta_0 + \sum_{i=1}^p \beta_i x_i.$$

This is the so-called logistic regression model. The name stems from the transformation of $p(\mathbf{x})$ used here, i.e., the $\log \frac{p(\mathbf{x})}{1-p(\mathbf{x})}$, which is the logistic

transformation that has been widely used in many areas such as physics and signal processing.

The logistic regression model can be further transformed:

$$p(\mathbf{x}) = \frac{1}{1+e^{-(\beta_0 + \sum_{i=1}^p \beta_i x_i)}}.$$

Based on this, we could predict $y = 1$ if $p(\mathbf{x}) \geq 0.5$, and $y = 0$ if $p(\mathbf{x}) < 0.5$.

As the thought process revealed above, logistic regression model is not the only eligible model for doing classification using linear form of the input variables. This is just one of the possibilities motivated by the linkage we can establish between $p(\mathbf{x})$ and $\beta_0 + \sum_{i=1}^p \beta_i x_i$ by $\log \frac{p(\mathbf{x})}{1-p(\mathbf{x})}$. Later we will learn more such models such as the Support Vector Machine (SVM) that still keep the linear form but follow different linkage functions. It is very important to know that this is a choice made by us, rather than a reality imposed on us nor a mathematical necessity that we have to accept.

II.2 Theory/Method

Now we show how to estimate the regression parameters in a logistic regression model.

The likelihood function is:

$$L(\boldsymbol{\beta}) = \prod_{n=1}^N p(\mathbf{x}_n)^{y_n} (1 - p(\mathbf{x}_n))^{1-y_n}.$$

We use the log-likelihood to turn products into sums:

$$l(\boldsymbol{\beta}) = \sum_{n=1}^N \{y_n \log p(\mathbf{x}_n) + (1 - y_n) \log(1 - p(\mathbf{x}_n))\}.$$

This could be further transformed into

$$l(\boldsymbol{\beta}) = \sum_{n=1}^N -\log \left(1 + e^{\beta_0 + \sum_{i=1}^p \beta_i x_{ni}} \right) - \sum_{n=1}^N y_n (\beta_0 + \sum_{i=1}^p \beta_i x_{ni}),$$

since

$$\sum_{n=1}^N \{y_n \log p(\mathbf{x}_n) + (1 - y_n) \log(1 - p(\mathbf{x}_n))\},$$

$$= \sum_{n=1}^N \log(1 - p(\mathbf{x}_n)) - \sum_{n=1}^N y_n \log \frac{p(\mathbf{x}_n)}{1-p(\mathbf{x}_n)},$$

$$= \sum_{n=1}^N -\log \left(1 + e^{\beta_0 + \sum_{i=1}^p \beta_i x_{ni}} \right) - \sum_{n=1}^N y_n (\beta_0 + \sum_{i=1}^p \beta_i x_{ni}).$$

The Newton-Raphson algorithm is commonly used to optimize the log-likelihood function of the logistic regression model to identify the optimal regression parameters. The Newton-Raphson algorithm is an iterative algorithm that seeks updates of the current solution using the following formula:

$$\boldsymbol{\beta}^{new} = \boldsymbol{\beta}^{old} - \left(\frac{\partial^2 l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} \right)^{-1} \frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}.$$

Here, $\boldsymbol{\beta}$ is the column vector form of the regression parameters.

We can show that

$$\begin{aligned} \frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \sum_{n=1}^N \mathbf{x}_n (y_n - p(\mathbf{x}_n)), \\ \frac{\partial^2 l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} &= - \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T p(\mathbf{x}_n) (1 - p(\mathbf{x}_n)). \end{aligned}$$

A certain structure can then be revealed if we rewrite it in matrix form:

$$\begin{aligned} \frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \mathbf{X}^T (\mathbf{y} - \mathbf{p}), \\ \frac{\partial^2 l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} &= -\mathbf{X}^T \mathbf{W} \mathbf{X}, \end{aligned}$$

where \mathbf{X} is the $N \times (p + 1)$ input matrix, \mathbf{y} is the $N \times 1$ column vector of y_i , \mathbf{p} is the $N \times 1$ column vector of $p(\mathbf{x}_n)$, and \mathbf{W} is a $N \times N$ diagonal matrix of weights with the n^{th} diagonal element as $p(\mathbf{x}_n)(1 - p(\mathbf{x}_n))$.

Then, plugging this into the updating formula of the Newton-Raphson algorithm, we can derive that

$$\begin{aligned} \boldsymbol{\beta}^{new} &= \boldsymbol{\beta}^{old} + (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} (\mathbf{y} - \mathbf{p}), \\ &= (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \left(\mathbf{X} \boldsymbol{\beta}^{old} + \mathbf{W}^{-1} (\mathbf{y} - \mathbf{p}) \right), \\ &= (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{z}, \end{aligned}$$

where $\mathbf{z} = \mathbf{X} \boldsymbol{\beta}^{old} + \mathbf{W}^{-1} (\mathbf{y} - \mathbf{p})$.

This resembles the generalized least squares (GLS) estimator of a regression model, where each data point (\mathbf{x}_n, y_n) is associated with a weight w_n to reduce the influence of potential outliers in fitting the regression model. This insight revealed by the Newton-Raphson algorithm suggests a new perspective to look at the logistic regression model. The

updating formula suggests that we are actually solving a weighted regression model as:

$$\boldsymbol{\beta}^{new} \leftarrow \arg \min_{\boldsymbol{\beta}} (\mathbf{z} - \mathbf{X}\boldsymbol{\beta})^T \mathbf{W} (\mathbf{z} - \mathbf{X}\boldsymbol{\beta}).$$

For this reason, this algorithm is also called the Iteratively Reweighted Least Square or IRLS algorithm. \mathbf{z} is referred as the adjusted response.

Putting all these together, a complete flow of the IRLS is shown in below:

1. Initialize $\boldsymbol{\beta}$.
2. Compute \mathbf{p} by its definition: $p(\mathbf{x}_n) = \frac{1}{1+e^{-(\beta_0 + \sum_{i=1}^p \beta_i x_{ni})}}$ for $n = 1, 2, \dots, N$.
3. Compute the diagonal matrix \mathbf{W} , while the n^{th} diagonal element as $p(\mathbf{x}_n)(1 - p(\mathbf{x}_n))$ for $n = 1, 2, \dots, N$.
4. Set \mathbf{z} as $= \mathbf{X}\boldsymbol{\beta} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$.
5. Set $\boldsymbol{\beta}$ as $(\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{z}$.
6. If the stopping criteria is met, stop; otherwise go back to step 2.

II.3 R Lab

Focusing on the AD dataset, now let's predict the diagnosis of the subjects either as normal or diseased. The variable, `DX_b1`, encodes the diagnosis information, i.e., “0” denotes normal while “1” denotes diseased.

```
#### Dataset of Alzheimer's Disease
#### Objective: prediction of diagnosis
# filename
AD <- read.csv('AD_b1.csv', header = TRUE)
str(AD)
```

First, let's examine a simple logistic regression model using the function `glm()` with only one predictor, `FDG`.

```
# Fit a Logistic regression model with FDG
logit.AD <- glm(DX_b1 ~ FDG, data = AD, family = "binomial")
summary(logit.AD)
```

```

## 
## Call:
## glm(formula = DX_b1 ~ FDG, family = "binomial", data = AD)
##
## Deviance Residuals:
##      Min      1Q  Median      3Q     Max
## -2.4686  -0.8166  -0.2758   0.7679   2.7812
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 18.3300   1.7676  10.37  <2e-16 ***
## FDG        -2.9370   0.2798 -10.50  <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: 711.27  on 516  degrees of freedom
## Residual deviance: 499.00  on 515  degrees of freedom
## AIC: 503
##
## Number of Fisher Scoring iterations: 5

```

It can be seen from the summary of the built model that the predictor **FDG** is significant, as the p-value is `<2e-16` that is far less than 0.05. Although it is not as straightforward as in linear regression model that we could derive the metric R-squared to examine how much proportion of the variability of the outcome variable could be explained away by the predictor, here, we could observe that, out of the total deviance of 711.27, $711.27 - 499.00 = 212.27$ could be explained by the predictor **FDG**.

We could then query what is the 95% CI of the regression parameter:

```

## CIs of the regression parameters using profiled log-likelihood
confint(logit.AD)

##                  2.5 %    97.5 %
## (Intercept) 15.033585 21.974091
## FDG        -3.513878 -2.415248

```

It is worthy of mentioning that, the `wald.test()` that builds on the Chi-squared test, is also another method that is often used in testing the significance of the regression parameters in logistic regression model. The

results by the `wald.test()` is usually in compliance with the approximated t-test results as shown in the `summary()` function as we have seen above.

```
## wald test for the regression coefficients
library(aod)

## Warning: package 'aod' was built under R version 3.3.3

wald.test(b = coef(logit.AD), Sigma = vcov(logit.AD), Terms = 2)
```

For instance, we can see that, by setting “`Terms = 2`”, the Chi-squared test used in the `wald.test()` shows significance of the predictor `FDG` in predicting `DX_b1`.

```
## Wald test:
## -----
## Chi-squared test:
## X2 = 110.2, df = 1, P(> X2) = 0.0
```

With a built model, it is of interest to see how it can be used to predict on a given dataset. Here, for presentational purpose, we randomly pick up 200 samples from the AD dataset to form the `AD.pred`, our imagined dataset to be predicted by the model.

```
# To predict on a given dataset
AD.pred <- AD[sample(1:dim(AD)[1], 200),]
# predict() uses all the temp values in dataset, including appended values
pred <- predict(logit.AD, AD.pred, type = "link", se.fit = TRUE)
AD.pred$fit <- pred$fit
AD.pred$se.fit <- pred$se.fit
```

We can readily convert these information into the 95% CIs of the predictions (the way these 95% CIs are derived are again, only in approximated sense).

```
# CI for fitted values
AD.pred <- within(AD.pred, {
  # added "fitted" to make predictions at appended temp values
  fitted = exp(fit) / (1 + exp(fit))
  fit.lower = exp(fit - 1.96 * se.fit) / (1 + exp(fit - 1.96 * se.fit))
  fit.upper = exp(fit + 1.96 * se.fit) / (1 + exp(fit + 1.96 * se.fit))
})
```

We can draw the following figure to visualize the prediction.

```
# visualize the prediction
library(ggplot2)
newData <- AD.pred[order(AD.pred$FDG),]
p <- ggplot(newData, aes(x = FDG, y = DX_b1))
# predicted curve and point-wise 95% CI
p <- p + geom_ribbon(aes(x = FDG, ymin = fit.lower, ymax = fit.upper),
per), alpha = 0.2)
p <- p + geom_line(aes(x = FDG, y = fitted), colour="red")
# fitted values
p <- p + geom_point(aes(y = fitted), size=2, colour="red")
# observed values
p <- p + geom_point(size = 2)
p <- p + ylab("Probability")
p <- p + labs(title = "Observed and predicted probability of disease")
print(p)
```

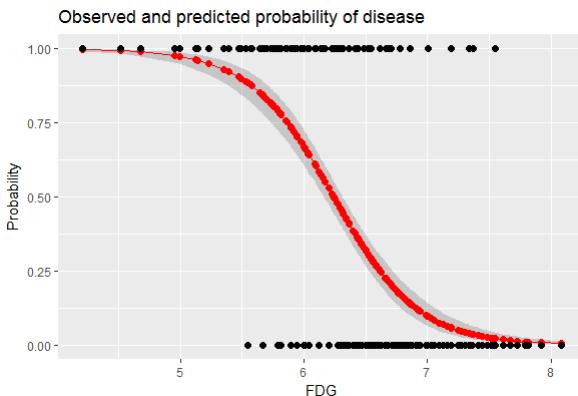


Figure 3.1: Predicted probabilities (with their 95% CIs) versus observed outcomes

The figure is shown in Figure 3.1. It can be seen that, the model prediction is significant and captures the relationship between `FDG` with `DX_b1` with a smooth logit curve, and the prediction confidences are fairly small (evidenced by the tight 95% CIs).

On the other hand, from Figure 3.1, we can also see that, while the single-predictor model is significant and does well on the two extreme ends of the probability range, in the middle part its prediction power is limited,

calling for more predictors to enhance its prediction power. Thus, in the next step, we decide to add more variables to the logistic regression model. Before we build the model, it is of interest to visualize the relationships between the predictors with the outcome variable. For example, in the following we show how the continuous variables could be presented in Boxplot form to see if the distribution of the continuous variable is different across the two classes of samples.

```
# install.packages("reshape2")
require(reshape2)

AD.long <- melt(AD[,c(1,2,4,5,6,7,19)], id.vars = c("ID", "DX_b1"))
# Plot the data using ggplot
require(ggplot2)
p <- ggplot(AD.long, aes(x = factor(DX_b1), y = value))
# boxplot, size=.75 to stand out behind CI
p <- p + geom_boxplot(size = 0.75, alpha = 0.5)
# points for observed data
p <- p + geom_point(position = position_jitter(w = 0.05, h = 0),
alpha = 0.1)
# diamond at mean for each group
p <- p + stat_summary(fun.y = mean, geom = "point", shape = 18, s
ize = 6,
alpha = 0.75, colour = "red")
# confidence limits based on normal distribution
p <- p + stat_summary(fun.data = "mean_cl_normal", geom = "errorb
ar",
width = .2, alpha = 0.8)
p <- p + facet_wrap(~ variable, scales = "free_y", ncol = 3)
p <- p + labs(title = "Boxplots of variables by diagnosis (0 - no
rormal; 1 - patient)")
print(p)
```

This code will generate Figure 3.2 which shows that some variables, such as `FDG` and `HippoNV`, could separate the two classes significantly. Some variables such as `AV45` and `AGE` have less prediction power, but still look promising. It is important to recognize that these figures only show marginal relationship among variables. Thus, while it is helpful, it is also important to keep in mind its limitations such as the inability to show synergistic effects among the variables.

Boxplots of variables by diagnosis (0 - normal; 1 - patient)

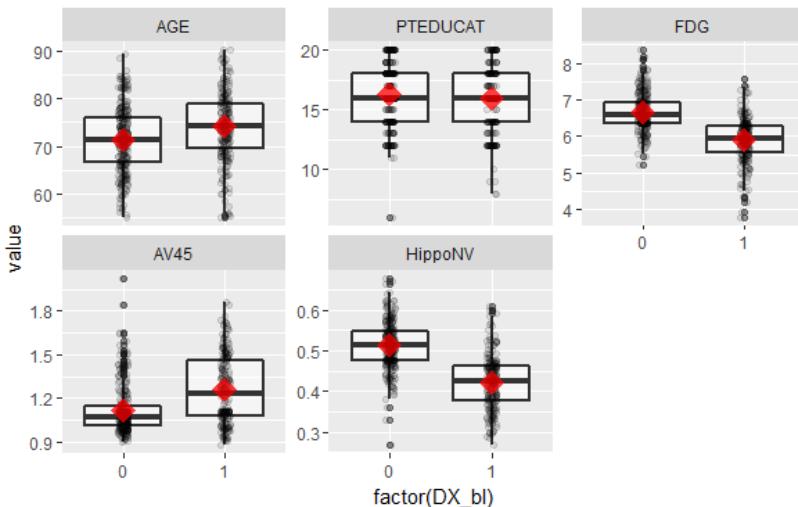


Figure 3.2: Boxplots of the continuous predictors in the two classes

Just like in the linear regression model, we could use the function **step()** to automatically select the best model given a set of variables.

```
# Automatic selection of the model
logit.AD.full <- glm(DX_bl ~ ., data = AD[,c(1:16)], family = "binomial")
logit.AD.final <- step(logit.AD.full, direction="both", trace = 0)
summary(logit.AD.final)

##
## Call:
## glm(formula = DX_bl ~ AGE + PTEDUCAT + FDG + AV45 + HippoNV +
##     rs3818361 + rs610932 + rs3851179, family = "binomial", dat
a = AD[,,
##     c(1:16)])
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.6385   -0.5022   -0.1146    0.3651    3.0694
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 31.17834   3.76132   8.289 < 2e-16 ***
## AGE         -0.03128   0.02134  -1.466   0.1427
```

```

## PTEDUCAT -0.12833 0.05087 -2.523 0.0116 *
## FDG -2.73262 0.33499 -8.157 3.43e-16 ***
## AV45 1.58053 0.72007 2.195 0.0282 *
## HippoNV -24.42793 2.74972 -8.884 < 2e-16 ***
## rs3818361 -0.43672 0.28703 -1.522 0.1281
## rs610932 0.47909 0.28390 1.688 0.0915 .
## rs3851179 -0.48461 0.27440 -1.766 0.0774 .
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 711.27 on 516 degrees of freedom
## Residual deviance: 344.47 on 508 degrees of freedom
## AIC: 362.47
##
## Number of Fisher Scoring iterations: 6

```

The final model selected by the backward-forward selection procedure implemented in the `step()` function is shown in below, which includes 8 predictors and one intercept. You may have noticed that some variables included in this model are actually not significant by the approximated t-test.

The model as a whole could be evaluated by the Chi-square test against the null hypothesis that there is a lack of fit. And it can be seen that the model shows no lack of fit as the p-value is 1.

```

# Test residual deviance for lack-of-fit (if > 0.10, little-to-no
# Lack-of-fit)
dev.p.val <- 1 - pchisq(logit.AD.final$deviance, logit.AD.final$d
f.residual)
dev.p.val

## [1] 1

```

Again, we can derive the 95% CIs of the regression coefficients:

```

# coefficients and 95% CI
cbind(coef = coef(logit.AD.final), confint(logit.AD.final))

```

And we can observe that:

```

##                 coef      2.5 %     97.5 %
## (Intercept) 31.17834039 24.15644635 38.94011124
## AGE         -0.03127754 -0.07367180  0.01022977
## PTEDUCAT   -0.12833007 -0.23028570 -0.03019323
## FDG        -2.73262447 -3.42455191 -2.10810353

```

```

## AV45          1.58052749   0.17699902   3.00488253
## HippoNV      -24.42793042  -30.12412328  -19.31065470
## rs3818361    -0.43672386   -1.00682550   0.12136335
## rs610932     0.47909019   -0.07263237   1.04320233
## rs3851179    -0.48460576   -1.02757559   0.05088704

```

Regarding the regression coefficients of logistic regression model, it is also of interest to convert them into odds ratios for another interpretation. This could be done by directly calling upon the definition of the odds ratio, as done in the codes shown in below:

```

## odds ratios and 95% CI
exp(cbind(OR = coef(logit.AD.final), confint(logit.AD.final)))

```

Then, we can derive the odds ratios and their 95% CIs.

```

##                               OR      2.5 %      97.5 %
## (Intercept) 3.472012e+13 3.097500e+10 8.155967e+16
## AGE          9.692065e-01 9.289765e-01 1.010282e+00
## PTEDUCAT    8.795630e-01 7.943066e-01 9.702580e-01
## FDG          6.504835e-02 3.256387e-02 1.214681e-01
## AV45         4.857517e+00 1.193630e+00 2.018384e+01
## HippoNV      2.460847e-11 8.265316e-14 4.106664e-09
## rs3818361    6.461498e-01 3.653770e-01 1.129035e+00
## rs610932     1.614605e+00 9.299426e-01 2.838292e+00
## rs3851179    6.159400e-01 3.578735e-01 1.052204e+00

```

Besides these significant tests and presentations of the model parameters, we can also look at the predictions of the model on the samples. We can use the function, **fitted()**, to derive the probabilities of diseased given by the model for the samples. Then, we could visualize the predictions using the boxplots:

```

# visualize the correlation
tempData = cbind(Yhat,AD$DX_b1)
require(ggplot2)
qplot(factor(AD$DX_b1), Yhat, data = AD,
      geom=c("boxplot"), fill = factor(AD$DX_b1),title="Prediction versus Observed")

```

The result is shown in Figure 3.3, which indicates that the model can separate the two classes significantly.

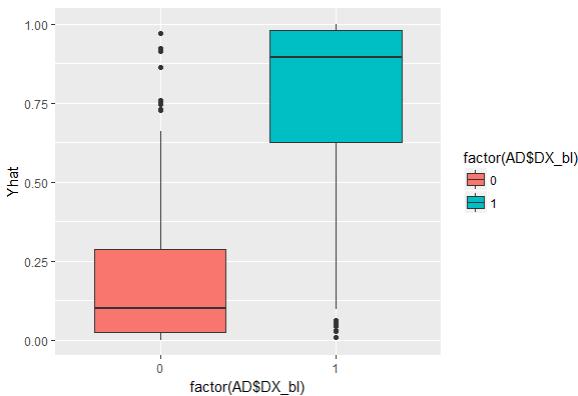


Figure 3.3: Boxplots of the predicted probabilities of diseased in the two classes

II.4 Remarks

Another perspective to look into the origin of logistic regression model is motivated by an empirical observation. Here, let's use the AD dataset, and pick up the variables, `HippoNV` and `DX_b1`, to see empirically what is the shape the relationship between the two takes. Specifically, here, we categorize the continuous variable `HippoNV` into distinct levels, and observe what is the prevalence of AD incidences within each level. The following R code serves this data processing purpose.

```
# Create the frequency table in accordance of categorization of HippoNV
temp = quantile(AD$HippoNV, seq(from = 0.05, to = 0.95, by = 0.05))
AD$HippoNV.category <- cut(AD$HippoNV, breaks=c(-Inf, temp, Inf))
tempData <- data.frame(xtabs(~DX_b1 + HippoNV.category, data = AD))
tempData <- tempData[seq(from = 2, to = 2*length(unique(AD$HippoNV.category))), by = 2,]
summary(xtabs(~DX_b1 + HippoNV.category, data = AD))

tempData$Total <- colSums(as.matrix(xtabs(~DX_b1 + HippoNV.category, data = AD)))
tempData$p.hat <- 1 - tempData$Freq/tempData$Total
tempData$HippoNV.category = as.numeric(tempData$HippoNV.category)
str(tempData)
```

We can use the `str(tempData)` to visualize the data we have converted, where 20 levels of `HippoNV` has been created; “Total” denotes the total number of subjects within each level, and “`p.hat`” denotes the proportion of the diseased subjects within each level (the prevalence).

```
str(tempData)
```

```
## 'data.frame': 20 obs. of 5 variables:
## $ DX_b1 : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2
## $ HippoNV.category: num 1 2 3 4 5 6 7 8 9 10 ...
## $ Freq : int 24 25 25 21 22 15 17 17 19 11 ...
## $ Total : num 26 26 26 26 25 26 26 26 34 ...
## $ p.hat : num 0.0769 0.0385 0.0385 0.1923 0.1538
...
```

We are now ready to further visualize the relationship between the two variables by drawing a scatterplot, as shown in Figure 3.4. We also use the “`loess`” method, which is a nonparametric smoothing method, to fit the relationship, which clearly shows a logit type functional shape of the relationship. This provides an empirical justification of the use of the logic transformation $\log \frac{p(x)}{1-p(x)}$, to $p(x)$ and $\beta_0 + \sum_{i=1}^p \beta_i x_i$, which gives birth to the logistic regression model.

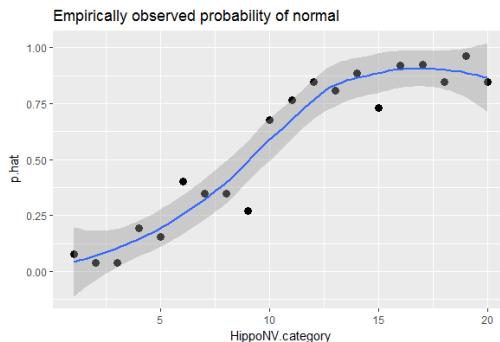


Figure 3.4: The empirical relationship between `HippoNV` and `DX_b1` takes a shape as the logit function

```
# Draw the scatterplot of HippoNV.category versus the probability
# of normal
library(ggplot2)

p <- ggplot(tempData, aes(x = HippoNV.category, y = p.hat))
p <- p + geom_point(size=3)
p <- p + geom_smooth(method = "loess")
p <- p + labs(title = "Empirically observed probability of normal",
  xlab = "HippoNV")
print(p)
```

III. A Product Ranking Problem by Pairwise Comparison

III.1 Rationale and Formulation

In recent years, we have witnessed a growing interest in estimating the ranks of a list of items. This same problem could be found in a variety of applications, such as the online advertisement of products on Amazon or movie recommendation by Netflix. These problems could be analytically summarized as: given a list of items denoted by $\mathbf{M} = \{M_1, M_2, \dots, M_p\}$, what is the rank (denoted by $\boldsymbol{\phi} = \{\phi_1, \phi_2, \dots, \phi_p\}$) we should attribute to them? Here, $\boldsymbol{\phi}$ is a vector of real values, i.e., the larger the ϕ_i , the higher the rank of M_i .

To obtain this ranking of the items, comparison data (either by domain expert or users) is often collected, e.g., a pair of items in \mathbf{M} , let's say, M_i and M_j , will be pushed to the expert/user who conduct the comparison to see if M_i is better than M_j , and then, a score, denoted as y_k , will be returned, i.e., a positive y_k indicates that the expert/user knowledge more tends to support that M_i is better than M_j , while a negative y_k indicates the opposite. Note that the larger the y_k , stronger the knowledge.

Following this line, we denote the initial data set as D_0 , which consists of the set of pairwise comparisons that are queried (denoted as a set \mathcal{S}_0) and the corresponding expert response data (denoted as a vector \mathbf{y}_0). The next question is, how to estimate the underlying ranking $\boldsymbol{\phi}$? And further, how to further collect pairwise comparison data to enhance the estimation

of $\boldsymbol{\phi}$, i.e., in other words, what should be the new comparisons in \mathbf{S}_1 so we can collect the corresponding new data \mathbf{y}_1 ?

III. 2 Theory/Method

Obviously, these are statistical questions. From the first glance, it looks unfamiliar. But in this paper¹, it is revealed that the underlying statistical model is a linear regression model! This surprising recognition indicates that we can readily use the rich array of methods and conclusions in linear regression framework to solve many problems in ranking!

To see that, first, it is important to make explicit what probabilistic relationship is implied in the pairwise comparison mechanism, that can be used to model the relationship between the parameter to be estimated ($\boldsymbol{\phi}$) and the data (D_0). Specifically, we could establish a probabilistic relationship between $\boldsymbol{\phi}$ and the observed D_0 , i.e., for the k^{th} comparison that involves items M_i and M_j , we could assume that y_k is distributed as:

$$y_k \sim N(\phi_i - \phi_j, \sigma^2/w_k).$$

This essentially assumes that if the item M_i is more (or less) important than the model M_j , we will expect to see positive (or negative) values of y_k . This is consistent with the nature of the expert/user comparison data in many applications. Note that, σ^2 encodes the overall accuracy level of the expert/user knowledge, as more knowledgeable expert/user will tend to have smaller σ^2 . Also, w_k encodes uncertainty in this particular comparison, acting as the local accuracy level of the expert/user knowledge. In practice, expert/user could also provide their confidence level, i.e., w_k , along with y_k . Alternatively, when this information is lacking, we could simply assume $w_k = 1$ for all the comparison data. Following this line, we could further illustrate how we could represent the comparison data in a more compact matrix form. This is shown in Figure 3.5.

¹ Osting, B., Brune, C. and Osher, S. Enhanced statistical rankings via targeted data collection. *Proceedings of the 30th International Conference on Machine Learning (ICML)* 2013.

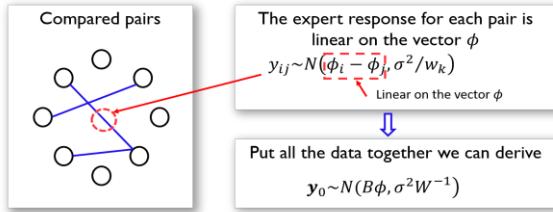


Figure 3.5: The data structure and its analytic formulation underlying the pairwise comparison. Each node is an item in M while each arc represents a comparison of two items

Note that it is straightforward to derive the structure of the matrix \mathbf{B} as shown in Figure 3.5:

$$\mathbf{B}_{kj} = \begin{cases} 1 & \text{if } j = \text{head}(k) \\ -1 & \text{if } j = \text{tail}(k) \\ 0 & \text{otherwise} \end{cases}$$

Here, $j = \text{tail}(k)$ if the k^{th} comparison is asked in the form as “if M_i is better than M_j ” (i.e., denoted as $M_i \rightarrow M_j$); otherwise, $j = \text{head}(k)$ for question asked in the form as $M_j \rightarrow M_i$.

Then, we can derive that

$$\mathbf{y} \sim N(\mathbf{B}\phi, \sigma^2 \mathbf{W}^{-1}).$$

where \mathbf{W} is the diagonal matrix of elements w_k for $k = 1, 2, \dots, K$. Thus, for the initial expert comparison data D_0 , we could derive that

$$\mathbf{y}_0 \sim N(\mathbf{B}_0\phi, \sigma^2 \mathbf{W}_0^{-1}).$$

where \mathbf{B}_0 is defined on the set S_0 . Using the framework developed in Chapter 2, we could derive the GLS estimator of ϕ as

$$\hat{\phi} = (\mathbf{B}_0^T \mathbf{W}_0 \mathbf{B}_0)^{-1} \mathbf{B}_0^T \mathbf{W}_0 \mathbf{y}_0.$$

The recognition of the linear regression formulation underling the ranking problem brings more insights and operational possibilities to solve the problem better. For example, as many design of experiments techniques have been developed for optimal data collection, while most are based on the linear regression framework, these techniques could find relevance in

this ranking problem, e.g., what new comparison data we should collect to optimize statistical accuracy and efficiency given limited budget? As shown in the paper, an E-optimal design method could be introduced here to optimally decide on which new comparison should be conducted. As this process involves Bayesian statistics, optimal design, and optimization, interested readers are encouraged to read the paper.

IV. Exercises

Data analysis

1. Create a new binary variable based on `AGE`, by labeling the subjects whose age is above the mean of `AGE` to be class “1” and labeling the subjects whose age is below the mean of `AGE` to be class “0”. Then, repeat the analysis shown in the R lab of this chapter for the logistic regression model and the analysis shown in the R lab of Chapter 2 for decision tree model. Identify the final models you would select, evaluate the models, and compare the regression model with the tree model.
2. Find two datasets from the UCI data repository or R datasets. Conduct a detailed analysis for both datasets using both logistic regression model and the tree model, e.g., for regression model, you may want to conduct model selection, model comparison, testing of the significance of the regression parameters, evaluation of the R-squared and significance of the model. Also comment on the application of your model on the context of the dataset you have selected.
3. Pick up any dataset you have used, and randomly split the data into two halves. Use one half to build the tree model and the regression model. Test the models’ prediction performances on the second half. Report what you have found, adjust your way of model building, and suggest a strategy to find the model you consider as the best.

Derivation

4. Use your pen and paper, write up the optimization process to estimate the regression parameters using the dataset in Table 2.4. If your optimization process requires more than 3 iterations to converge, delineate 3 iterations is sufficient.

Programming

5. Write your own R script to implement the IRLS algorithm of a logistic regression model. Compare the output from your script with the output from `glm()`.
6. Write your own R script to generate prediction of new data points using an estimated logistic regression model.
7. Write your own R script to implement the ranking problem formulated as a linear regression model. Test it on the dataset shown below to estimating the ϕ .

$M_6 \rightarrow M_4 = 2.33, M_5 \rightarrow M_6 = 1.80, M_4 \rightarrow M_3 = -7.45, M_2 \rightarrow M_8 = 13.18, M_2 \rightarrow M_6 = 4.37, M_1 \rightarrow M_5 = 0.32, M_7 \rightarrow M_2 = -0.43$.

(Here, for validation only, the comparison data is generated from $\phi = \{3.9, 10.2, 6.7, 1.7, 5.2, 3.4, 7.8, 2.3\}$.

CHAPTER 4: COMPUTATION

BOOTSTRAP AND RANDOM FOREST

I. Overview

Chapter 3 is about “**Computation**”. It is how we can work with computer, exploiting its remarkable power in iterations and conducting repetitive tasks which human beings often find burdensome to do. It is this capacity of computers in conducting repetitive tasks that enables applications of modern optimization algorithms, which underly many data analytics models. This capacity also provides powerful nonparametric statistical techniques, leading to developments of powerful computational statistical models that don’t require analytic tractability. A particular invention that has played a critical role in many data analytic applications is the Bootstrap technique. Building on the idea of Bootstrap and its variants, Random Forest was also invented together with many more powerful ensemble learning methods.

II. How Bootstrap Works

II.1 Rationale and Formulation

There are multiple perspectives to look at Bootstrap. One perspective that has been well studied in the seminar book¹ of Efron and Tibshirani is to treat Bootstrap as a simulation of the “sampling process”. As we know, sampling refers to the idea that we could draw samples again and again from the same population. Many statistical techniques make sense only when we consider the possibility of conducting sampling. For example, when we say the type 1 error in a hypothesis testing is 0.05, we mean that on average we may reject the null hypothesis 5 times even when it is true – if we conduct the same hypothesis testing 100 times (i.e., by repeating the sampling process 100 times, each time we calculate the test statistics and compare it with the critical value, and make a decision).

Of course, Bootstrap is not a real sampling process since no new data points are really collected. It is a simulated sampling process. Probably the idea of Bootstrap could be better demonstrated in Figure 4.1:



Figure 4.1: A demonstration of the Bootstrap process

As shown in Figure 4.1, a collected dataset has 5 samples. The 5 samples provide a representation of the underlying population. To mimic the

¹ Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 2993.

sampling process that draws samples from the underlying population, Bootstrap suggests that we could resample the 5 samples to generate Bootstrapped datasets instead of really drawing samples from the underlying population. The idea seems to be simple but is very effective.

II.2 Theory/Method

The importance of the sampling process to classic statistics: Many classic statistical theories are built on the sampling process. For example, let's consider the estimator of the mean of a normal population. Assume that we have a random variable X that follows a normal distribution, $X \sim N(\mu, \sigma^2)$. For simplicity, let's assume that we have known the variance σ^2 . So we want to estimate the mean μ . What we need to do is to perform a sampling process, by randomly drawing a few samples from the distribution. Denote these samples as x_1, x_2, \dots, x_n . To estimate μ , it seems natural to use the average of the samples, denoted as $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Thus, we propose to use \bar{x} as an estimator of μ , i.e., $\hat{\mu} = \bar{x}$.

A question arises, how good is \bar{x} to be an estimator of μ ?

To evaluate the uncertainty of the estimated μ by \bar{x} , in theory, we need to repeat the sampling process and the estimation again and again. If \bar{x} is in theory a good estimator of μ , then we should be able to repeatedly observe that \bar{x} is numerically close to μ in the replications. Fortunately, when assuming that X follows a normal distribution, we could derive that \bar{x} is another normal distribution, $\bar{x} \sim N(\mu, \sigma^2/n)$. Thus, without really doing the physical experiments to repeat the sampling process and drawing many samples, we can answer the previous question. First, we know that \bar{x} is an unbiased estimator as $E(\bar{x}) = \mu$. Also, we know that the larger the sample size, the better estimation of μ by \bar{x} , since the variance of the estimator is σ^2/n . Knowing the analytic form of \bar{x} is the key here.

Bootstrap – a computational remedy when the sampling process cannot be analytically articulated: But how about we don't know what is

the distribution of X ? Then it would make the question difficult to answer as we don't know what is the analytic form of \bar{x} .

Bootstrap provides such a computational remedy that enables us to investigate the properties of literally any estimator by computationally mimicking the sampling process. For example, while the distribution of X is unknown, we could follow the Bootstrap scheme illustrated in Figure 4.2 to evaluate the sampling distribution of \bar{x} .

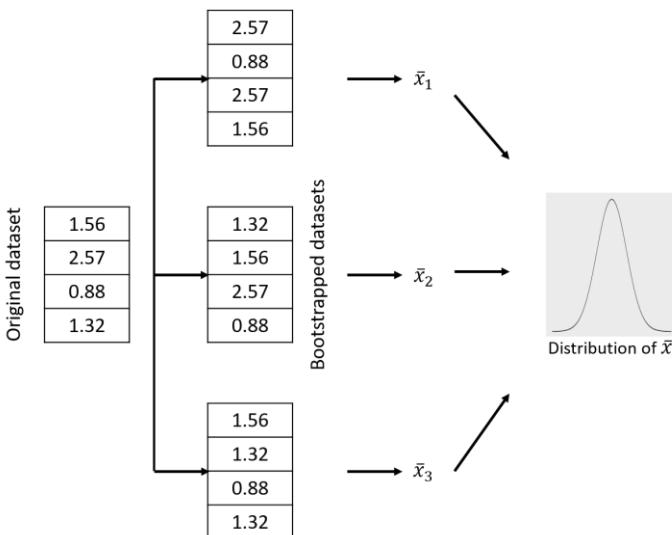


Figure 4.2: The (nonparametric) Bootstrap scheme for computationally evaluating the sampling distribution of \bar{x}

The Bootstrap scheme illustrated in Figure 4.2 is called nonparametric Bootstrap since no parametric information is used. This is not the only way we can conduct Bootstrap for studying the properties of any estimator in the sampling process. For example, a parametric Bootstrap scheme is illustrated in Figure 4.3 to perform the same study – to study the sampling distribution of \bar{x} . The only difference between the nonparametric Bootstrap scheme in Figure 4.2 and the parametric Bootstrap scheme in Figure 4.3 is that, when generating new samples, the nonparametric Bootstrap uses the

original dataset while the parametric Bootstrap uses the fitted distribution model.

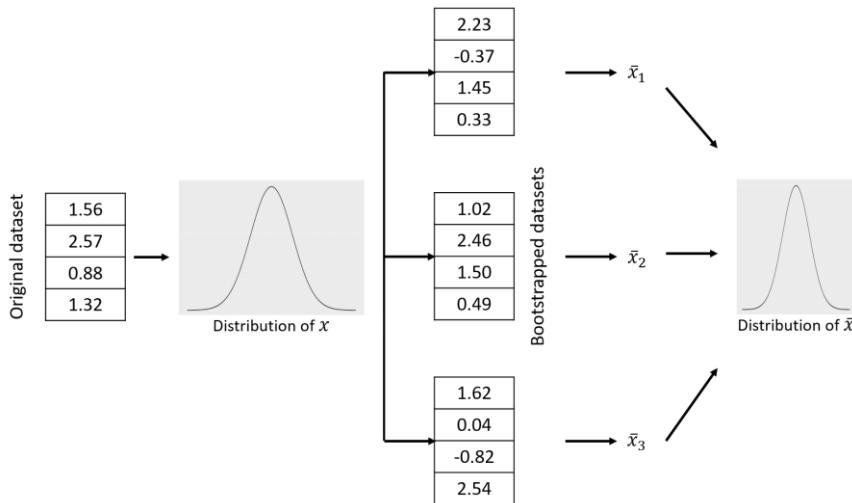


Figure 4.3: The (parametric) Bootstrap scheme for computationally evaluating the sampling distribution of \bar{x}

Bootstrap for regression model: How to evaluate the uncertainties of the regression parameters using Bootstrap?

In Chapter 2, we showed that by imposing the Gaussian assumptions on the error term of the regression model, we come to the recognition that the estimated regression parameters are also random variables, and the propagation of the uncertainty from the error term to the estimation of the regression parameters could be analytically articulated due to the benefit of the linear relationship assumed between predictors and outcome variable. In summary, the Gaussian assumption and the linear assumptions are critical for the analytic tractability.

Here, we introduce a more generic approach, based on the idea of Bootstrap, that could be applied on cases where we don't have to require those assumptions. Using Bootstrap to evaluate the linear regression model, we encounter a variety of options:

Option 1: we could simply resample the data points (i.e., the (\mathbf{x}, y) pairs) similarly as the nonparametric Bootstrap scheme. Then, for each sampled dataset, we can fit a regression model and obtain the fitted regression parameters. Suppose we repeat this sampling process 10,000 times, we could obtain 10,000 set of estimated regression parameters. These could be enough for us to evaluate the sampling distribution of the regression parameters and see if the parameters are significantly different from zero.

Option 2: we could simulate new samples of X using the nonparametric Bootstrap method on the samples of X only. Then, for the new samples of X , we draw samples of Y using the fitted conditional distribution model $P(Y|X)$. This is a combination of the nonparametric and parametric Bootstrap methods to simulate X and Y . Then, for each sampled dataset, we can fit a regression model and obtain the fitted regression parameters.

Option 3: we could fix the X , only sample for Y . In this way we implicitly assume that the uncertainty of the dataset mainly comes from Y . To sample Y , we draw samples using the fitted conditional distribution model $P(Y|X)$. In this way we could also generate many new datasets, such that we can generate many sets of fitted regression parameters.

The three options above are just some examples. There are other options that could be developed. As a matter of fact, as a more complicated model than simple parametric estimation in distribution fitting, how to conduct Bootstrap on regression models is a challenging problem that demands solutions from a variety of perspectives bearing different assumptions. Similarly, Bootstrap for other complex models such as time series models or decision tree models has demonstrated to be a challenging problem. This will be further discussed later in this Chapter when introducing the Random Forest.

II.3 R Lab

In what follows we implement the Bootstrap scheme in statistical tasks such as parameter estimation, samples comparison, and regression models. First, let's load the AD dataset into the R workspace:

```
#### Dataset of Alzheimer's Disease  
#### Objective: prediction of diagnosis  
AD <- read.csv('AD_b1.csv', header = TRUE)  
str(AD)
```

Let's pick up the variable **HippoNV**. The first statistical task we'd like to see is to estimate the mean of **HippoNV** in the population under study. Assuming that the variable **HippoNV** is distributed as a normal distribution, we could use the **fitdistr()** function from the R package “MASS” to estimate the parameters, mean and standard derivation, as shown in below:

```
require(MASS)  
  
fit <- fitdistr(AD$HippoNV, densfun="normal")
```

The **fitdistr()** function returns the estimated parameters together with their standard derivation. Note that, here, the standard derivation of the estimated parameters is derived based on the statistical theory underlying this estimation, that builds on the assumption of normality of the variable.

```
fit  
  
##      mean          sd  
##  0.471662891  0.076455789  
##  (0.003362522) (0.002377662)
```

To give a visual sense of this estimation, the R code in below shows the histogram of the variable **HippoNV** and the normal curve with the estimated parameters:

```
hist(AD$HippoNV, pch=20, breaks=25, prob=TRUE, main="")  
curve(dnorm(x, fit$estimate[1], fit$estimate[2]), col="red", lwd=2, add=T)
```

From Figure 4.4, it seems that the normality assumption is reasonable and the estimation of the parameters fits the empirical data well.

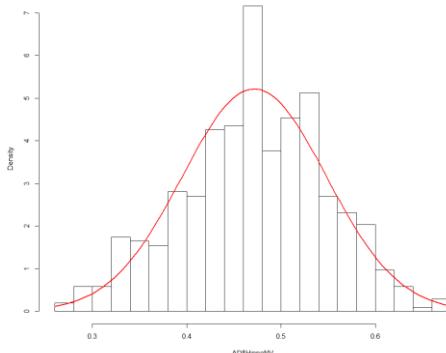


Figure 4.4: Histogram of HippoNV and its estimated normal curve

Now, let's focus on the question of how uncertain this estimation is. As we mentioned, the reason that the standard derivation of the estimated parameters could be provided by calling upon `fit()` is because the normality assumption is assumed. If we don't want to make this assumption, Bootstrap could be used in a nonparametric way as shown in Figure 4.2. The following R codes implement the nonparametric Bootstrap for this parameter estimation problem:

```
# draw R bootstrap replicates
R <- 10000
# init location for bootstrap samples
bs_mean <- rep(NA, R)
# draw R bootstrap resamples and obtain the estimates
for (i in 1:R) {
  resam1 <- sample(AD$HippoNV, dim(AD)[1], replace = TRUE)
  fit <- fitdistr(resam1 , densfun="normal")
  bs_mean[i] <- fit$estimate[1]
}
```

Here, 10,000 replications are simulated by the Bootstrap method. The `(bs_mean)` is a vector of 10,000 elements to record all the estimated mean parameter in these replications. These 10,000 estimated parameters could be taken as a set of samples. The following R code is used to compute the 95% CI of the estimated mean.

```

# sort the mean estimates to obtain bootstrap CI
bs_mean.sorted <- sort(bs_mean)
# 0.025th and 0.975th quantile gives equal-tail bootstrap CI
CI.bs <- c(bs_mean.sorted[round(0.025*R)], bs_mean.sorted[round(0.975*R+1)])
CI.bs

```

It could be seen that this 95% CI is pretty much close to the 95% CI provided by theoretical result, showing the validity and efficacy of the Bootstrap method to evaluate the uncertainty of a statistical operation.

```
CI.bs
```

```
## [1] 0.4649877 0.4781062
```

The following R codes draws a histogram of the `bs_mean` to give us some visual information about the Bootstrapped estimation of the mean.

```

## Plot the bootstrap distribution with CI
# First put data in data.frame for ggplot()
dat.bs_mean <- data.frame(bs_mean)

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.3.3

p <- ggplot(dat.bs_mean, aes(x = bs_mean))
p <- p + geom_histogram(aes(y=..density..))
p <- p + geom_density(alpha=0.1, fill="white")
p <- p + geom_rug()
# vertical line at CI
p <- p + geom_vline(xintercept=CI.bs[1], colour="blue", linetype=
"longdash")
p <- p + geom_vline(xintercept=CI.bs[2], colour="blue", linetype=
"longdash")
p <- p + labs(title = "Bootstrap distribution of mean estimate of
HippoNV")
print(p)

```

And the histogram is shown in Figure 4.5.

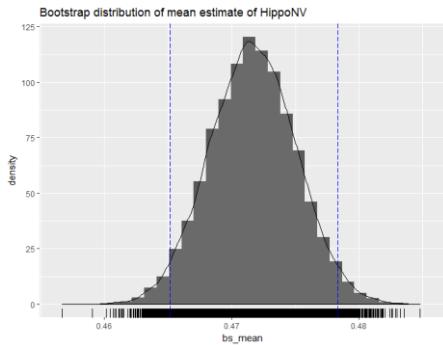


Figure 4.5: Histogram of the estimated mean parameter of HippoNV by Bootstrap with 10,000 replications

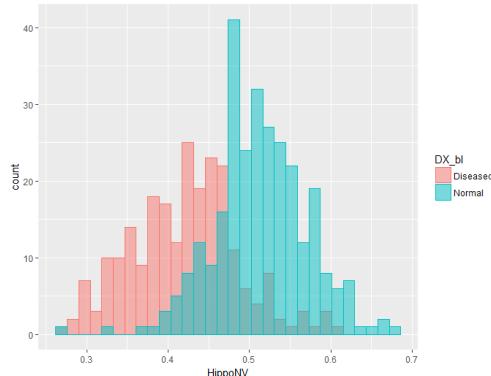


Figure 4.6: Histograms of HippoNV in the normal and diseased groups

While the estimation of the mean of HippoNV is a simple operation, in what follows we consider a relatively more complex statistical operation, comparison of the mean parameters of HippoNV across the two classes, normal and diseased. The following R code creates a temporary dataset for this purpose.

```
tempData <- data.frame(AD$HippoNV, AD$DX_b1)
names(tempData) = c("HippoNV", "DX_b1")
tempData$DX_b1[which(tempData$DX_b1==0)] <- c("Normal")
tempData$DX_b1[which(tempData$DX_b1==1)] <- c("Diseased")
```

We then use ggplot to visualize the two distributions by comparing their histograms, which is shown in Figure 4.6.

```
p <- ggplot(tempData,aes(x = HippoNV, colour=DX_b1))
p <- p + geom_histogram(aes(y = ..count.., fill=DX_b1), alpha=0.5,position="identity")
print(p)
```

The following R code shows how the nonparametric Bootstrap method as shown in Figure 4.2 can be implemented here.

```
# draw R bootstrap replicates
R <- 10000
# init location for bootstrap samples
bs0_mean <- rep(NA, R)
bs1_mean <- rep(NA, R)
# draw R bootstrap resamples and obtain the estimates
for (i in 1:R) {
  resam0 <- sample(tempData$HippoNV[which(tempData$DX_b1=="Normal")],
                    length(tempData$HippoNV[which(tempData$DX_b1=="Normal")]),
                    replace = TRUE)
  fit0 <- fitdistr(resam0 , densfun="normal")
  bs0_mean[i] <- fit0$estimate[1]
  resam1 <- sample(tempData$HippoNV[which(tempData$DX_b1=="Diseased")],
                    length(tempData$HippoNV[which(tempData$DX_b1=="Diseased")]),
                    replace = TRUE)
  fit1 <- fitdistr(resam1 , densfun="normal")
  bs1_mean[i] <- fit1$estimate[1]
}
bs_meanDiff <- bs0_mean - bs1_mean

# sort the mean estimates to obtain bootstrap CI
bs_meanDiff.sorted <- sort(bs_meanDiff)
# 0.025th and 0.975th quantile gives equal-tail bootstrap CI
CI.bs <- c(bs_meanDiff.sorted[round(0.025*R)], bs_meanDiff.sorted[round(0.975*R+1)])
CI.bs
```

Then, the 95% CI of the difference of the two mean parameters is:

```
CI.bs
```

```
## [1] 0.08066058 0.10230428
```

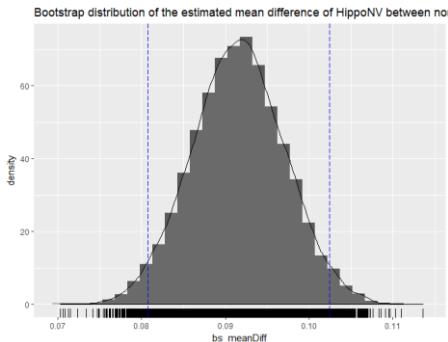


Figure 4.7: Histogram of the estimated mean difference of HippoNV in the two groups by Bootstrap with 10,000 replications

The following R codes draws a histogram of the `bs_meanDiff` to give us some visual information about the Bootstrapped estimation of the mean difference, which is shown in Figure 4.7.

```
## Plot the bootstrap distribution with CI
# First put data in data.frame for ggplot()
dat.bs_meanDiff <- data.frame(bs_meanDiff)

library(ggplot2)
p <- ggplot(dat.bs_meanDiff, aes(x = bs_meanDiff))
p <- p + geom_histogram(aes(y=..density..))
p <- p + geom_density(alpha=0.1, fill="white")
p <- p + geom_rug()
# vertical Line at CI
p <- p + geom_vline(xintercept=CI.bs[1], colour="blue", linetype="longdash")
p <- p + geom_vline(xintercept=CI.bs[2], colour="blue", linetype="longdash")
p <- p + labs(title = "Bootstrap distribution of the estimated mean difference of HippoNV between normal and diseased")
print(p)
```

Now let's implement the Bootstrap on the regression model, a more complicated statistical operation than the aforementioned two. First, we recall the use of `lm()` function to fit the regression model of `MMSCORE` using the demographics variables. Note that in this procedure, the standard

derivation of the estimated regression parameters could be derived by theory (i.e., by assuming the error term is a normal distribution).

```
# Use Bootstrap for multiple regression model
tempData <- data.frame(AD$MMSCORE, AD$AGE, AD$PTGENDER, AD$PTEDUCAT)
names(tempData) <- c("MMSCORE", "AGE", "PTGENDER", "PTEDUCAT")
lm.AD_demo <- lm(MMSCORE ~ AGE + PTGENDER + PTEDUCAT, data = tempData)
summary(lm.AD_demo)
```

The fitted regression model is:

```
## 
## Call:
## lm(formula = MMSCORE ~ AGE + PTGENDER + PTEDUCAT, data = tempData)
## 
## Residuals:
##      Min    1Q Median    3Q   Max 
## -8.4290 -0.9766  0.5796  1.4252  3.4539 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 27.70377   1.11131  24.929 < 2e-16 ***
## AGE         -0.02453   0.01282  -1.913   0.0563 .  
## PTGENDER     -0.43356   0.18740  -2.314   0.0211 *  
## PTEDUCAT     0.17120   0.03432   4.988 8.35e-07 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 2.062 on 513 degrees of freedom
## Multiple R-squared:  0.0612, Adjusted R-squared:  0.05571 
## F-statistic: 11.15 on 3 and 513 DF,  p-value: 4.245e-07
```

Now, let's discard the assumption of normality of the error term, and use Bootstrap to induce perturbation into the data and see if the significance of the estimated parameters could resist this perturbation.

```
# draw R bootstrap replicates
R <- 10000
# init location for bootstrap samples
bs_lm.AD_demo <- matrix(NA, nrow = R, ncol = length(lm.AD_demo$coefficients))
# draw R bootstrap resamples and obtain the estimates
for (i in 1:R) {
  resam_ID <- sample(c(1:dim(tempData)[1]), dim(tempData)[1], replace = TRUE)
```

```

resam_Data <- tempData[resam_ID,]
bs.lm.AD_demo <- lm(MMSCORE ~ AGE + PTGENDER + PTEDUCAT, data =
resam_Data)
bs.lm.AD_demo[i,] <- bs.lm.AD_demo$coefficients
}

```

As `bs.lm.AD_demo` records all the estimated regression parameters in the 10,000 replications, here, for illustration purpose, we can take a look at the regression coefficient of the variable `AGE` by the following R code.

```

bs.AGE <- bs.lm.AD_demo[,2]
# sort the mean estimates of AGE to obtain bootstrap CI
bs.AGE.sorted <- sort(bs.AGE)

# 0.025th and 0.975th quantile gives equal-tail bootstrap CI
CI.bs <- c(bs.AGE.sorted[round(0.025*R)], bs.AGE.sorted[round(0.975*R+1)])
CI.bs

```

Then we can see the 95% CI of `AGE` is shown in below, which includes 0 in the range. This is consistent with the conclusion made in the aforementioned analysis which shows that the variable `AGE` is insignificant (i.e., $p\text{-value}=0.0563$) by t-test that is based on the normality assumption.

`CI.bs`

```
## [1] -0.053940482  0.005090523
```

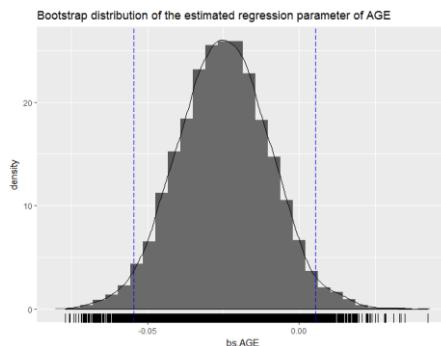


Figure 4.8: Histogram of the estimated regression parameter of `AGE` by Bootstrap with 10,000 replications

The following R codes draws a histogram of the Bootstrapped estimation of the regression parameter of AGE to give us some visual information about the Bootstrapped estimation, which is shown in Figure 4.8.

```
## Plot the bootstrap distribution with CI
# First put data in data.frame for ggplot()
dat.bs.AGE <- data.frame(bs.AGE.sorted)

library(ggplot2)
p <- ggplot(dat.bs.AGE, aes(x = bs.AGE))
p <- p + geom_histogram(aes(y=..density..))
p <- p + geom_density(alpha=0.1, fill="white")
p <- p + geom_rug()
# vertical line at CI
p <- p + geom_vline(xintercept=CI.bs[1], colour="blue", linetype=
"longdash")
p <- p + geom_vline(xintercept=CI.bs[2], colour="blue", linetype=
"longdash")
p <- p + labs(title = "Bootstrap distribution of the estimated re-
gression parameter of AGE")
print(p)
```

Then we can see the 95% CI of PTEDUCAT as shown in below, which is between 0.1021189 and 0.2429209. This is consistent with the conclusion made in the aforementioned analysis which shows that the variable PTEDUCAT is significant (i.e., p-value=8.35e-07) by t-test that is based on the normality assumption.

```
bs.PTEDUCAT <- bs_lm.AD_demo[,4]
# sort the mean estimates of PTEDUCAT to obtain bootstrap CI
bs.PTEDUCAT.sorted <- sort(bs.PTEDUCAT)

# 0.025th and 0.975th quantile gives equal-tail bootstrap CI
CI.bs <- c(bs.PTEDUCAT.sorted[round(0.025*R)], bs.PTEDUCAT.sorted
[round(0.975*R+1)])
CI.bs

CI.bs

## [1] 0.1021189 0.2429209
```

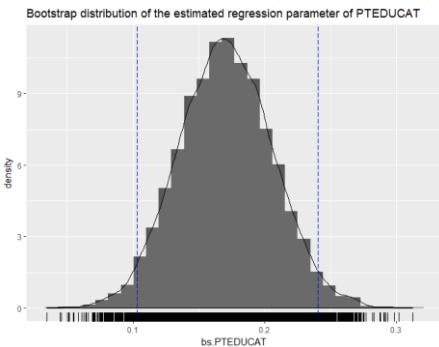


Figure 4.9: Histogram of the estimated regression parameter of **PTEDUCAT** by Bootstrap with 10,000 replications

The following R codes draws a histogram of the Bootstrapped estimation of the regression parameter of **PTEDUCAT**, which is shown in Figure 4.9.

```
## Plot the bootstrap distribution with CI
# First put data in data.frame for ggplot()
dat.bs.PTEDUCAT <- data.frame(bs.PTEDUCAT.sorted)

library(ggplot2)
p <- ggplot(dat.bs.PTEDUCAT, aes(x = bs.PTEDUCAT))
p <- p + geom_histogram(aes(y=..density..))
p <- p + geom_density(alpha=0.1, fill="white")
p <- p + geom_rug()
# vertical Line at CI
p <- p + geom_vline(xintercept=CI.bs[1], colour="blue", linetype="longdash")
p <- p + geom_vline(xintercept=CI.bs[2], colour="blue", linetype="longdash")
p <- p + labs(title = "Bootstrap distribution of the estimated regression parameter of PTEDUCAT")
print(p)
```

III. Random Forests

III.1 Rationale and Formulation

Building on the decision tree model, a random forest consists of multiple tree models. There are two main sources for randomness. First,

each tree is built on a randomly selected set of samples by applying Bootstrap on the original dataset. Second, in building a tree, specifically in splitting a node in the tree, a subset of features is randomly selected to choose the best split. Figure 4.10 shows this scheme of random forest.

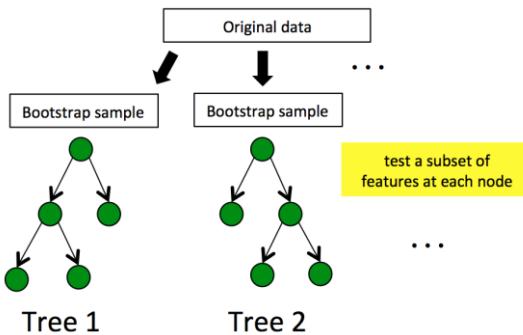


Figure 4.10: How random forest uses Bootstrap to grow trees

Random forest is a powerful machine learning method that has gained superior performances in many practical tasks, including many high-profiled data competitions over the past few years. It is a simple but effective mechanism to aggregate many simple models to tackle complex prediction task. Note that it is not necessary that in machine learning a random put-together of many simple models would lead to better performance than its constituting parts. Here we use the following example to show why the random forest, as a sum, is better than its parts.

The following R code generates a data set with two predictor variables and a class variable as the outcome variable. As shown in Figure 4.11, the two classes are separable by a linear boundary.

```
rm(list = ls(all = TRUE))
require(rpart)
require(dplyr)
require(ggplot2)
require(randomForest)
ndata <- 2000
X1 <- runif(ndata, min = 0, max = 1)
```

```
X2 <- runif(ndata, min = 0, max = 1)
data <- data.frame(X1, X2)
data <- data %>% mutate(X12 = 0.5 * (X1 - X2), Y = ifelse(X12 >= 0, 1, 0))
data <- data %>% select(-X12) %>% mutate(Y = as.factor(as.character(Y)))
ggplot(data, aes(x = X1, y = X2, color = Y)) + geom_point() + labs(title = "Data points")
```

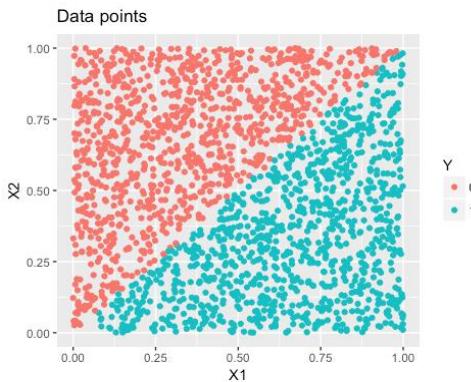


Figure 4.11: A linearly separable dataset with two predictors

Both the random forest and the decision tree are applied to the data. The classification boundaries the models can generate are shown in Figure 4.12 and Figure 4.13, for decision tree and random forest, respectively.

```
rf_model <- randomForest(Y ~ ., data = data)
tree_model <- rpart(Y ~ ., data = data)

pred_rf <- predict(rf_model, data, type = "prob")[, 1]
pred_tree <- predict(tree_model, data, type = "prob")[, 1]
data_pred <- data %>% mutate(pred_rf_class = ifelse(pred_rf < 0.5, 0, 1)) %>%
  mutate(pred_rf_class = as.factor(as.character(pred_rf_class)))
%>% mutate(pred_tree_class = ifelse(pred_tree < 0.5, 0, 1)) %>% mutate(pred_tree_class = as.factor(as.character(pred_tree_class)))
ggplot(data_pred, aes(x = X1, y = X2, color = pred_tree_class)) + geom_point()
```

```
  labs(title = "Classification boundary from a single decision  
tree")
```

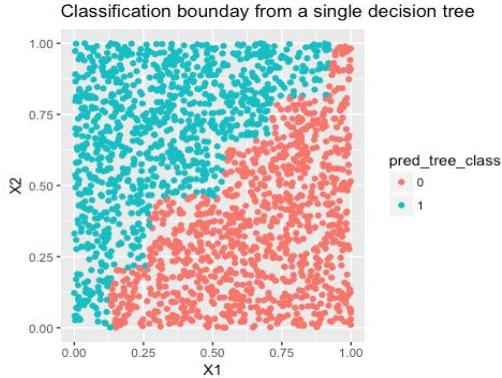


Figure 4.12: The decision boundary of one single decision tree

```
ggplot(data_pred, aes(x = X1, y = X2, color = pred_rf_class)) + g  
eom_point() +  
  labs(title = "Classification bounday from random forests")
```

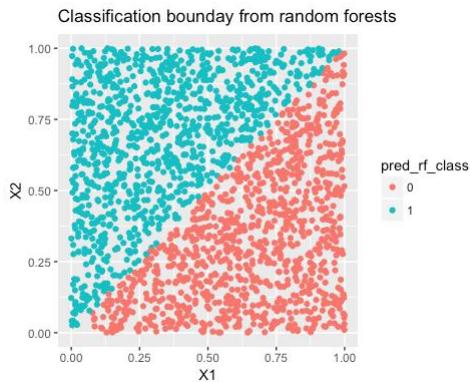


Figure 4.13: The decision boundary of a random forest

As we can see from Figure 4.12, the classification boundary generated by the decision tree model has difficult to approximate the linear boundary. There is an inherent limitation of a tree model to fit smooth boundaries due to its box-shaped nature resulting from its use of rules to segment the data space for making predictions. In contrast, the classification boundary of random forest is much smoother than the one of the decision tree, and is a better approximation of the linear classification boundary.

III.2 Theory/Method

Pretty much like decision tree, the theoretical line of random forest follows the algorithmic modeling framework which is very different from the data modeling framework of the linear regression models. Thus, random forest is more of a systematically organized set of heuristics, rather than highly regulated algebraic operations derived from a mathematical characterization. Motivated by this recognition, we present the process of random forest using a simple example with the data shown in below.

ID	X1	X2	Class
1	1	1	C0
2	1	0	C1
3	0	1	C1
4	0	0	C0

For random forests with m trees, each tree is built on a resampled dataset that consists of data instances randomly selected from the original data set, often with the same size as the original data set, **sampled with replacement**. As shown in Figure 4.14, the first resampled dataset includes data instances (represented by their IDs) $\{1,1,3,4\}$ and is used for building the first tree. The second resampled dataset includes data instances (represented by their IDs) $\{2,3,4,4\}$ and is used for building the second tree. And so on so forth, until the maximum number of trees is built.

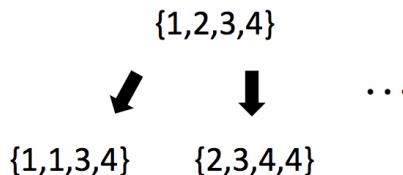


Figure 4.14: Bootstrap a dataset in random forest to build trees

To build the first tree, we begin with the root node that contains $\{1,1,3,4\}$. Then, we need to split the root node and reduce impurity. In the `randomForest` R package, the **Gini index** is used measure impurity. The Gini index for a data set is defined as

$$Gini = \sum_{c=1}^C p_c(1 - p_c),$$

where C is the number the classes in the dataset, and p_c is the proportion of data instances that come from the class c .

The Gini index plays the same role as the entropy we have introduced in Chapter 2. Here, using the following R code, we plot the Gini index and entropy values versus the percentage of class 1 (for two-class problems) to see their similarity, as shown in Figure 4.15.

```

entropy <- function(p_v) {
  e <- 0
  for (p in p_v) {
    if (p == 0) {
      this_term <- 0
    } else {
      this_term <- -p * log2(p)
    }
    e <- e + this_term
  }
  return(e)
}
gini <- function(p_v) {
  e <- 0
  for (p in p_v) {
    if (p == 0) {
      this.term <- 0
    } else {
      this.term <- p * (1 - p)
    }
  }
  return(e)
}

```

```

        }
        e <- e + this.term
    }
    return(e)
}

entropy.v <- NULL
gini.v <- NULL
p.v <- seq(0, 1, by = 0.01)
for (p in p.v) {
    entropy.v <- c(entropy.v, (entropy(c(p, 1 - p))))
    gini.v <- c(gini.v, (gini(c(p, 1 - p))))
}
plot(p.v, gini.v, type = "l", ylim = c(0, 1), xlab = "percentage
of class 1",
     col = "red", ylab = "impurity measure", cex.lab = 1.5, cex.ax
is = 1.5, cex.main = 1.5,
     cex.sub = 1.5)
lines(p.v, entropy.v, col = "blue")
legend("topleft", legend = c("Entropy", "Gini index"), col = c("b
lue", "red"), lty = c(1, 1), cex = 0.8)

```

It can be seen in Figure 4.15 that the two impurity measures are highly correlated. Both reach minimum of zero when there is only one class in the dataset, and maximum when there are equal number of data instances for different classes. In practice, thus, they produce similar trees.

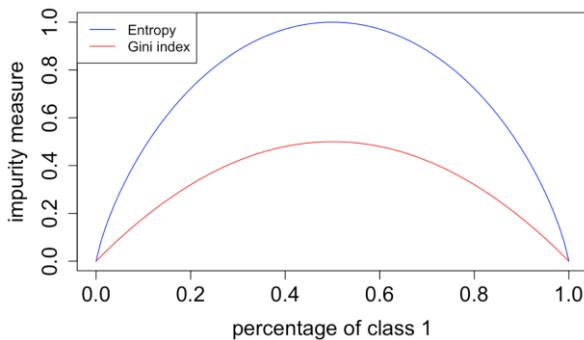


Figure 4.15: Gini index versus Entropy

Similar as the information gain, the **Gini gain** can be defined as

$$\nabla Gini = Gini - w_i Gini_i,$$

where $Gini$ is the Gini index at the node to be split; w_i and $Gini_i$, are the proportion of samples and the Gini index at the i^{th} children node, respectively.

Back to the example, the Gini index of the root node of the first tree is calculated as

$$\frac{3}{4} * \frac{1}{4} + \frac{1}{4} * \frac{3}{4} = 0.375.$$

The possible splitting rule candidates include four options: $X_1 = 0$, $X_2 = 0$, $X_1 = 1$ and $X_2 = 1$. Since both variables have two distinct values, both splitting rules $X_1 = 0$ and $X_1 = 1$ will produce the same children nodes, and both splitting rules $X_2 = 0$ and $X_2 = 1$ will produce the same children nodes. Therefore, we can reduce the possible splitting rule candidates to two: $X_1 = 0$ and $X_2 = 0$.

Further, as we mentioned earlier in this section, the second source of randomness in a random forest is to randomly select the variables for splitting a node. In general, for a data set with p predictor variables, \sqrt{p} variables are randomly selected for splitting. In our simple example, as there are two variables, we assume that X_1 is randomly selected for splitting the root node. Thus, $X_1 = 0$ is used for splitting the root node which generates the decision tree model as shown in Figure 4.16.

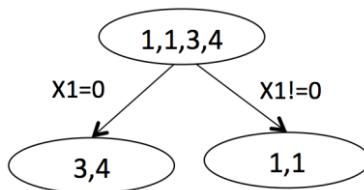


Figure 4.16: The decision tree with one split

The Gini gain can be calculated as

$$0.375 - 0.5 * 0 - 0.5 * 0.5 = 0.125.$$

Let's continue to grow the tree. Now, at the internal node containing data $\{3,4\}$, assume that X_2 is randomly selected. The node can be further split as shown in Figure 4.17.

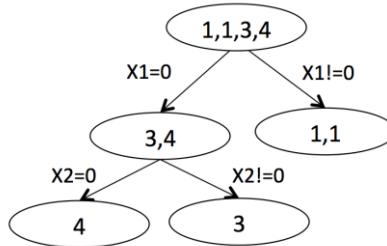


Figure 4.17: The decision tree with two splits

At this point, all nodes cannot be split further, and each leaf node can be labeled with the majority class of the node such that they become decision nodes. Thus, the final tree model is shown in Figure 4.18. Applying this decision tree to the 4 training data points, we can get the error rate as 25%.

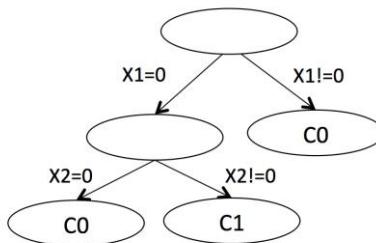


Figure 4.18: The final decision tree model with decision nodes

Similarly, the second, third, ..., m^{th} trees can be built. Usually, in random forest models, the pruning is not needed. Rather, we control the

depth of the tree models to be created (i.e., use the parameter `nodesize` in the function `randomForest`).

To make a prediction for a data point, each tree makes a prediction for the data point, and the random forest model combines these predictions and selects the most popular prediction among all trees as the final prediction.

Note that, each tree in random forests can be weak classifier or even wrong model. But when in aggregation, the joint predictions become stronger. In what follows, we apply random forest on the toy example data with different number of trees. For each random forest model, we run the experiments 200 times and collect its overall performance using boxplots as shown in Figure 4.19.

```
require(dplyr)
require(ggplot2)
require(randomForest)
set.seed(1)
data <- rbind(c("0", "0", "C0"), c("1", "0", "C1"), c("0", "1", "C1"),
  c("0", "0", "C0")) %>% as.data.frame()
colnames(data) <- c("X1", "X2", "Classs")

results <- NULL
for (i in c(1:9, (1:10) * 10)) {
  for (replicate in 1:200) {
    rf.model <- randomForest(Classs ~ ., data = data, ntree = i,
      keep.inbag = TRUE)
    pred.rf <- predict(rf.model, data, type = "class")
    err <- (length(which(pred.rf == data$Classs))/length(data$Classs))
    results <- rbind(results, c(i, err))
  }
}
colnames(results) <- c("num_trees", "accuracy")
results <- as.data.frame(results) %>% mutate(num_trees = as.character(num_trees))
levels(results$num_trees) <- unique(results$num_trees)
results$num_trees <- factor(results$num_trees, unique(results$num_trees))
ggplot() + geom_boxplot(data = results, aes(y = accuracy, x = num_trees)) +
  geom_point(size = 3)
```

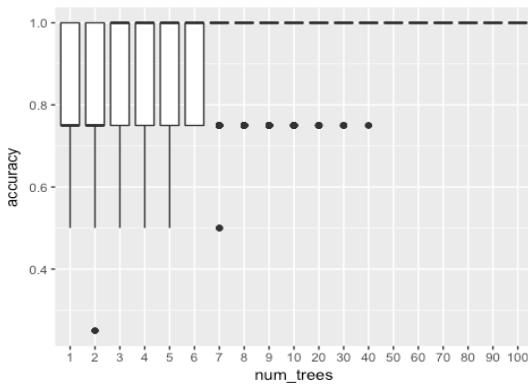


Figure 4.19: Accuracy versus number of trees in a random forest model

As shown in Figure 4.19, we can notice that when there are a small number of trees (e.g., smaller than 50), the performance is not stable. When the number of trees is greater than 50, the accuracy stabilizes. In practice, it is usually hard to say how many trees are best. Considerable amount of efforts of model tuning and selection is usually needed to make random forest works best on a dataset.

III.3 R Lab

We apply both decision tree and random forests to the AD dataset. Half of the datasets are used for training and the other half for testing. This is run for 20 times, and the boxplots of the errors from decision tree and random forests are plotted in Figure 4.20 using the following R code.

```
library(rpart)
library(dplyr)
library(tidyr)
library(ggplot2)
require(randomForest)
set.seed(1)

theme_set(theme_gray(base_size = 15))
```

```

path <- "../../data/AD_b1.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_b1")
data[, target_indx] <- as.factor(paste0("c", data[, target_indx]))
rm_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))
data <- data[, -rm_indx]

err.tree <- NULL
err.rf <- NULL
for (i in 1:20) {
  train.ix <- sample(nrow(data), floor(nrow(data)/2))
  tree <- rpart(DX_b1 ~ ., data = data[train.ix, ])
  pred.test <- predict(tree, data[-train.ix, ], type = "class")
  err.tree <- c(err.tree, length(which(pred.test != data[-train.ix, ]$DX_b1))/length(pred.test))

  rf <- randomForest(DX_b1 ~ ., data = data[train.ix, ])
  pred.test <- predict(rf, data[-train.ix, ], type = "class")
  err.rf <- c(err.rf, length(which(pred.test != data[-train.ix, ]$DX_b1))/length(pred.test))
}
err.tree <- data.frame(err = err.tree, method = "tree")
err.rf <- data.frame(err = err.rf, method = "random_forests")

ggplot() + geom_boxplot(data = rbind(err.tree, err.rf), aes(y = err, x = method)) +
  geom_point(size = 3)

```

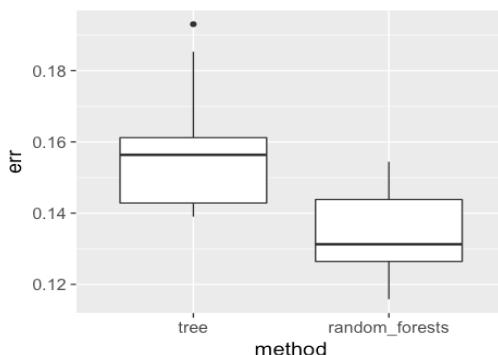


Figure 4.20: Performance of random forest versus tree model on the AD data

From Figure 4.20 we can see that the error rates of decision tree are higher than random forests. Now we investigate the impact of the number of trees and the number of features on the performance of random forest. First, let's consider the number of trees (i.e., use the parameter `ntree` in the function `randomForest`). For each number of trees, 20 runs are conducted, and the boxplots for each setting are shown in Figure 4.21.

```

library(rpart)
library(dplyr)
library(tidyverse)
library(ggplot2)
require(randomForest)
set.seed(1)

theme_set(theme_gray(base_size = 15))

path <- "../../data/AD_b1.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_b1")
data[, target_indx] <- as.factor(paste0("c", data[, target_indx]))
rm_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))
data <- data[, -rm_indx]

results <- NULL
for (itree in c(1:9, 10, 20, 50, 100, 200, 300, 400, 500, 600, 700)) {
  for (i in 1:20) {
    train.ix <- sample(nrow(data), floor(nrow(data)/2))
    rf <- randomForest(DX_b1 ~ ., ntree = itree, data = data[train.ix, ])
    pred.test <- predict(rf, data[-train.ix, ], type = "class")
  }
  this.err <- length(which(pred.test != data[-train.ix, ]$DX_b1))/length(pred.test)
  results <- rbind(results, c(itree, this.err))
  # err.rf <- c(err.rf, Length(which(pred.test !=
  # data[-train.ix, ]$DX_b1))/Length(pred.test) )
}
}

colnames(results) <- c("num_trees", "error")
results <- as.data.frame(results) %>% mutate(num_trees = as.character(num_trees))

```

```
levels(results$num_trees) <- unique(results$num_trees)
results$num_trees <- factor(results$num_trees, unique(results$num_trees))
ggplot() + geom_boxplot(data = results, aes(y = error, x = num_trees)) + geom_point(size = 3)
```

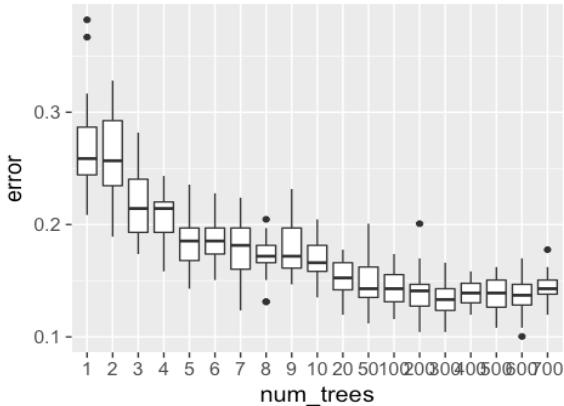


Figure 4.21: Error versus number of trees in a random forest model

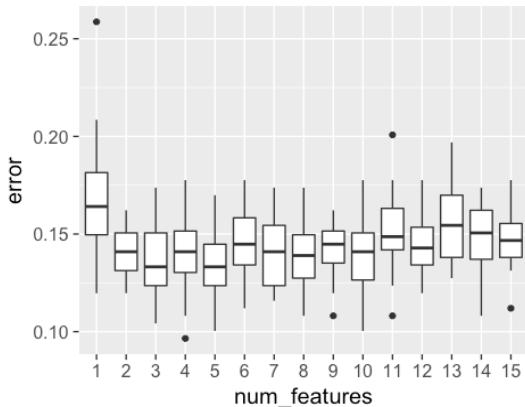


Figure 4.22: Error versus number of features in a random forest model

It can be seen in Figure 4.21 that, when the number of trees is small, particularly less than 10, the improvement on prediction performance of random forest is substantial with additional trees added. However, the error rates become stable after the number of trees reaches 100.

Next, let's consider the number of features (i.e., use the parameter `mtry` in the function `randomForest`). Here, 100 trees are used. For each number of features, 20 runs are conducted, and the boxplots for each setting are shown in Figure 4.22. It can be seen that the error rates are not significantly different when the number of features changes.

```
library(rpart)
library(dplyr)
library(tidyr)
library(ggplot2)
require(randomForest)
set.seed(1)
theme_set(theme_gray(base_size = 15))
path <- "../../data/AD_bl.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_bl")
data[, target_indx] <- as.factor(paste0("c", data[, target_indx]))
rm_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))
""))
data <- data[, -rm_indx]
nFea <- ncol(data) - 1
results <- NULL
for (iFeatures in 1:nFea) {
    for (i in 1:20) {
        train.ix <- sample(nrow(data), floor(nrow(data)/2))
        rf <- randomForest(DX_bl ~ ., mtry = iFeatures, ntree = 1
00, data = data[train.ix,
                ])
        pred.test <- predict(rf, data[-train.ix, ], type = "class
")
        this.err <- length(which(pred.test != data[-train.ix, ]$D
X_bl))/length(pred.test)
        results <- rbind(results, c(iFeatures, this.err))
        # err.rf <- c(err.rf, length(which(pred.test !=
# data[-train.ix,]$DX_bl))/length(pred.test) )
    }
}
colnames(results) <- c("num_features", "error")
```

```

results <- as.data.frame(results) %>% mutate(num_features = as.character(num_features))
levels(results$num_features) <- unique(results$num_features)
results$num_features <- factor(results$num_features, unique(results$num_features))
ggplot() + geom_boxplot(data = results, aes(y = error, x = num_features)) +
  geom_point(size = 3)

```

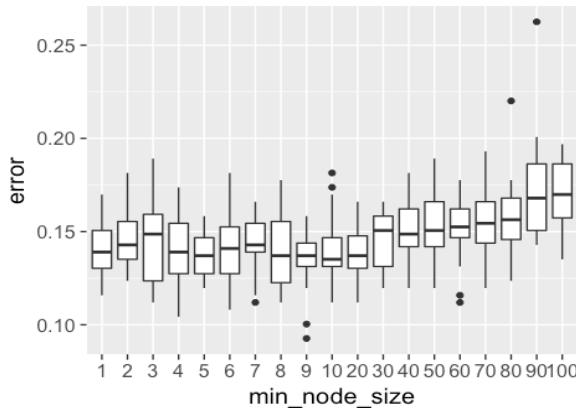


Figure 4.23: Error versus node size in a random forest model

Further, we experiment with the minimum node size (i.e., use the parameter `nodelsize` in the function `randomForest`), that is, the minimum number of instances at a node. This is a parameter to control the depth of the trees. Again each setting is run 20 times and boxplots of their performances are shown in Figure 4.23.

```

library(dplyr)
library(tidyr)
library(ggplot2)
require(randomForest)
set.seed(1)

theme_set(theme_gray(base_size = 15))

path <- "../../data/AD_b1.csv"

```

```

data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_bl")
data[, target_indx] <- as.factor(paste0("c", data[, target_indx]))
rm_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))
""))
data <- data[, -rm_indx]

results <- NULL
for (inodesize in c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50,
  60, 70, 80,
  90, 100)) {
  for (i in 1:20) {
    train.ix <- sample(nrow(data), floor(nrow(data)/2))
    rf <- randomForest(DX_bl ~ ., ntree = 100, nodesize = inodesize, data = data[train.ix,
      ])
    pred.test <- predict(rf, data[-train.ix, ], type = "class")
  }
  this.err <- length(which(pred.test != data[-train.ix, ]$DX_bl))/length(pred.test)
  results <- rbind(results, c(inodesize, this.err))
  # err.rf <- c(err.rf, length(which(pred.test !=
  # data[-train.ix, ]$DX_BL))/length(pred.test) )
}
}

colnames(results) <- c("min_node_size", "error")
results <- as.data.frame(results) %>% mutate(min_node_size = as.character(min_node_size))
levels(results$min_node_size) <- unique(results$min_node_size)
results$min_node_size <- factor(results$min_node_size, unique(results$min_node_size))
ggplot() + geom_boxplot(data = results, aes(y = error, x = min_node_size)) +
  geom_point(size = 3)

```

It can be seen that, the error rates start to rise at minimum node size equal to 40. And the error rates are not substantially different when minimum node size less than 40. More importantly, this shows that a fully-grown tree, that is, minimum node size equal to 1, does not hurt the accuracy performance of random forests.

III.4 Remarks

Random forest provides a great example to show when randomness should be consciously introduced into the model to boost its performance. This seems to be counterintuitive, as a model is supposed to characterize randomness and extract the constancy out of randomness. Actually, the randomness in the random forest, enabled by the use of Bootstrap to randomize choices of data instances and the use of random feature selection for building trees, is the key for its success. We provide an intuitive explanation that, why random forests work better than a single decision tree with the introduction of these randomness. Assuming that the trees in random forests are independent, and each tree has an accuracy of 0.6. For 100 trees, the probability of random forests to make the right prediction reaches as high as 0.97:

$$\sum_{k=51}^{100} C(n, k) * 0.6^k * 0.4^{100-k}.$$

Note that, the assumption of the independency between the trees in random forests is the key here. This does not hold in reality in a strict sense. However, the randomness added to each tree makes them less correlated.

IV. Exercises

Data analysis

1. Use **AGE** as the new outcome variable. Build a random forest model to predict it. Identify the final models you would select, evaluate the model, and compare it with the decision tree model.
2. Find two datasets from the UCI data repository or R datasets. Conduct a detailed analysis for both datasets using the random forest model. Also comment on the application of your model on the context of the dataset you have selected.
3. Pick up any dataset you have used, and randomly split the data into two halves. Use one half to build the random forest model. Test the model's prediction performance on the second half. Report what you have found, adjust your way of model building, and suggest a strategy to find the model you consider as the best.

Programming

4. Write your own R script to use Bootstrap to evaluate the significance of the regression parameters of logistic regression model. Compare your results with the output from `glm()`.
5. Write your own R script to implement the random forest algorithm. Use any dataset, compare the output from your script with the output from `randomForest()`.
6. Based on your script in 5, replace the decision tree model with logistic regression model, to generate a “random forest of logistic regression” model. Compare its performance with random forest on some datasets you have worked on.

CHAPTER 5: PERFORMANCE

CROSS VALIDATION AND OOB

I. Overview

Chapter 4 is about “**Performance**”. This is often a concept that seems to be self-evident, and therefore, ignored by people to give further consideration. A model performs well – what does that mean anyway?

For example, let’s consider the prediction of a rare disease. By statistics it has been known that only 0.001% of the population of the United States have this disease. The Center of Disease Control (CDC) now hires a data analytics expert to build a model, and it is said that the model can achieve a prediction accuracy as high as 90%. Isn’t this a good model? However, it is easy to beat this performance, if we consider a very trivial model that simply predicts all the upcoming cases as negative (no disease). Wouldn’t this trivial model achieve a prediction accuracy as high as 99.999%?

Now let’s look at another example. Figure 5.1 shows three models to fit the same dataset that has two classes of data points. The first model is a linear model ($f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$) with a straight line as the

decision boundary. Obviously, this model has its inherent limitation such that many data points have to be misclassified with a linear line. To add in some curvature into the decision boundary justified by the nonlinear shape of the two classes' boundaries, some second order terms and an interaction term of the two predictors are introduced to the model ($f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2$), giving rise to the model shown in the middle panel of Figure 5.1. While this improved model still could not classify the two classes completely, more interaction terms defined on the predictors are introduced into the model ($f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2 + \beta_{112} x_1^2 x_2 + \beta_{122} x_1 x_2^2 + \dots$). As shown in the right panel of Figure 5.1, this model can achieve 100% of prediction accuracy.

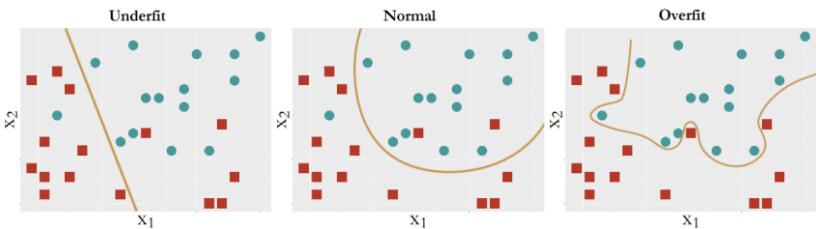


Figure 5.1: Three models to fit a dataset

The three models in Figure 5.1, from left to right, illustrates “**underfit**”, “**good fit**”, and “**overfit**”, respectively. The underfit model, apparently, fails to incorporate something systematical in the dataset to help classify the two classes. The overfit model allows the noises to affect the model. Noises, by definition, only happen by accident. While the model, by definition, is to generalize the constancy of the data rather than its unrepeatable randomness. A dataset could be randomly generated, but the mechanism of generating the randomness is the constancy, as revealed in many phenomena such as Brownian motion. Thus, the model in the middle panel

of Figure 5.1 seems to be able to maintain a balance, using the structural constancy in the data to form the model, while resisting the suspicious noises in the data.

A similar study could be conducted on regression problems. As we have mentioned in Chapter 2, the metric R-squared is used to measure the goodness-of-fit of the regression model on training data. Look at the definition of the R-squared,

$$R^2 = 1 - \frac{SSE}{SST}.$$

Here, SST is the total sum of squares, SSE is the residual sum of squares, and it is known that SST-SSE is the explained sum of squares by the model. Thus, R^2 higher the better, meaning more variance in the data could be explained by the model. However, on the other hand, we can see that SST is always fixed, while SSE could only decrease if more variables are put into the model even if these new added variables have no relationship with the outcome variable.

Further, the R-squared is compounded by the variance of predictors as well. As the underlying regression model is

$$Y = \beta X + \epsilon,$$

the variance of Y , $\text{var}(Y) = \beta^2 \text{var}(X) + \text{var}(\epsilon)$. The R-squared takes the form as

$$\text{R-squared} = \frac{\beta^2 \text{var}(X)}{\beta^2 \text{var}(X) + \text{var}(\epsilon)}.$$

Thus, it seems that R-squared is not only impacted by how well X can predict Y , but also by the variance of X as well.

Thus, the drawback of using R-squared is that it doesn't account for model complexity. The adjusted R-squared was developed to provide a remedy for this. Some other criteria such as the **AIC** and **BIC** were also developed which have a good balance between the model fit (just like R-

squared) and model complexity (i.e., how many predictors are used in the model).

II. Cross-Validation

II.1 Rationale and Formulation

The examples shown above collectively point out the complexity of defining the performance of a model and the danger of evaluating the performance of a model using training data. To solve this problem, a common strategy is to look at multiple dimensions of the performance of a model, and use **cross-validation** to obtain the performance metrics on a validation dataset that is not used in model training. The ideal situation is that, there is a **training dataset** to train the model and an independent **testing dataset** to validate the model. The testing dataset should not be available when training the model, which is the key for validation purpose. Thus, in training the model with a given dataset, we need to try our best to make sure the model can perform well on the testing dataset. Cross-validation serves this purpose to train the model without accessibility to a testing dataset. The only information that the cross-validation uses, which is really an assumption, is that the testing dataset and the training dataset are randomly generated by the same distribution. With a given dataset to train the model, the cross-validation techniques mimic the ideal situation, aim to predict the model's success on the unseen testing datasets as its ultimate goal.

II.2 Theory/Method

The first approach, probably the simplest one, is the “hold-out” method. With a given dataset, the hold-out method further divides the given dataset into two parts, a training dataset and a testing dataset. Then, the model is trained on the training dataset. Its performance is evaluated on the testing dataset. Note that, when deciding on the final model that will be used on

the testing dataset to obtain its performance, the testing dataset could not be used to guide the model selection on the training stage. In other words, the testing dataset is simply for evaluation only.



Figure 5.2: The hold-out method

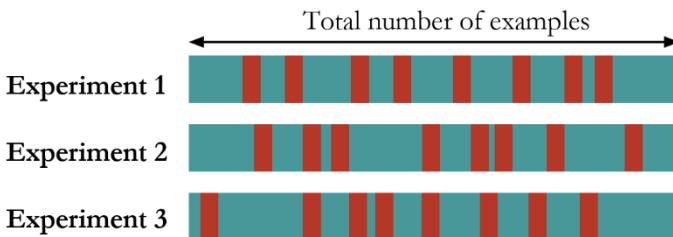


Figure 5.3: The random sampling method

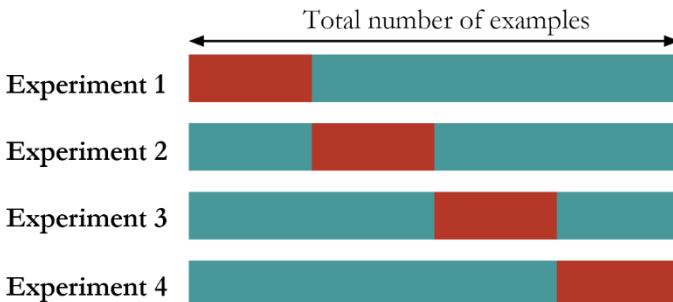


Figure 5.4: The K-fold cross-validation method (here, K=4)

The hold-out method is simple but is criticized for its one-time division of the dataset into two parts, which maybe prone to random errors. Thus, the random sampling method suggests to repeat this division process many times, i.e., as shown in Figure 5.3, the process is repeated 3 times. For each

time, the model training and selection only use the training dataset, and the model evaluation only uses the testing dataset. The performance of the model on the three experiments could be reported either in average or in a boxplot that shows both the average performance and its variance.

Somehow like a mix of the random sampling method and the hold-out method, the K-fold cross-validation method suggests to first divide the dataset into K folds, and then, train the model using K-1 folds of the dataset and test the model using the remaining fold. This process could be repeated K times. The performance of the model on the experiments could be reported either in average or in a boxplot that shows both the average performance and its variance.

II.3 R Lab

The R lab in this section is built on the script provided in milanor.net¹. Here, we simulate a simple dataset with one predictor and outcome variable. We use the `ns()` function to simulate the relationship between the two variables, which can generate the B-spline basis matrix for natural cubic splines. The nice merit of using this method is that the relationship between the two variables should be more complex than linear, but the complexity is controlled by the degree of freedom (`df`) parameter, i.e., the larger the `df`, the more complex the relationship. Thus, the complexity of the relationship is quantitatively characterized on a continuum.

```
# Write a nice simulator to generate dataset with one predictor and one outcome
# from a polynomial regression model
seed <- rnorm(1)
set.seed(seed)
gen_data <- function(n, coef, v_noise) {
  eps <- rnorm(n, 0, v_noise)
  x <- sort(runif(n, 0, 100))
  X <- cbind(1, ns(x, df = (length(coef) - 1)))
  y <- as.numeric(X %*% coef + eps)
```

¹ <http://www.milanor.net/blog/cross-validation-for-predictive-analytics-using-r/>

```

    return(data.frame(x = x, y = y))
}

```

The dataset that is generated by the R code showing above is presented in Figure 5.5, as the scattered data points.

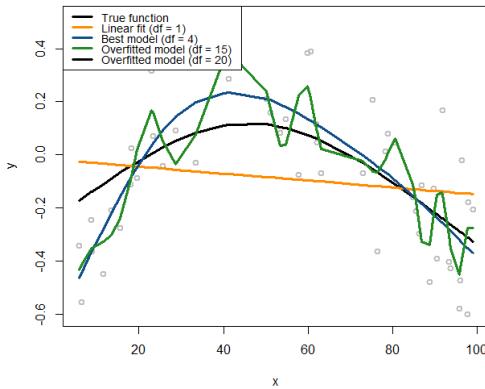


Figure 5.5: The simulated data from a nonlinear regression model with B-spline basis matrix ($\text{df} = 4$), and various fitted models with different degree of freedoms

We then fit the data with a variety of models, starting from $\text{df} = 1$ (corresponds to the linear model) to $\text{df} = 20$.

```

# Fit the data using different models with different degrees of freedom (df)
# install.packages("splines")
require(splines)

## Loading required package: splines

# Simulate one batch of data, and see how different model fits with df from 1 to 50

n_train <- 100
coef <- c(-0.68, 0.82, -0.417, 0.32, -0.68)
v_noise <- 0.2
n_df <- 20
df <- 1:n_df
tempData <- gen_data(n_train, coef, v_noise)

```

```

x <- tempData[, "x"]
y <- tempData[, "y"]
fit <- apply(t(df), 2, function(degf) lm(y ~ ns(x, df = degf)))

# Plot the data
x <- tempData$x
X <- cbind(1, ns(x, df = (length(coef) - 1)))
y <- tempData$y
plot(y ~ x, col = "gray", lwd = 2)
lines(x, X %*% coef, lwd = 3, col = "black")
lines(x, fitted(fit[[1]]), lwd = 3, col = "darkorange")
lines(x, fitted(fit[[4]]), lwd = 3, col = "dodgerblue4")
# lines(x, fitted(fit[[10]]), lwd = 3, col = "darkorange")
lines(x, fitted(fit[[20]]), lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("True function", "Linear fit (df = 1)", "Best model (df = 4)", "Overfitted model (df = 15)", "Overfitted model (df = 20)", 1
wd = rep(3, 4), col = c("black", "darkorange", "dodgerblue4",
"forestgreen"), text.width = 32, cex = 0.
85)

```

As shown in Figure 5.5, the linear model obviously underfits the data as it lacks the flexibility to characterize the complexity sufficiently. The model that has `df` = 20 overfits the data, evidenced by its complex shape with many change points, up and downs. As we know that the underlying true model is smooth, the model with `df` = 20 tries too hard to fit the local patterns that were only induced by noise.

Note that, in this example, we have known that the true `df` is 4, which helps us to recognize the overfitted and underfitted models. In reality we don't have this knowledge, so it is dangerous to keep increasing the model complexity to aggressively pursue the accuracy performance on the training data. To see the danger, let's do another experiment.

First, we use the following R code to generate test data from the same distribution of the training data.

```

# Generate test data from the same model
n_test <- 50
xy_test <- gen_data(n_test, coef, v_noise)

```

Then, we fit the same set of models from linear to $\text{df}=20$ using the training dataset. And we compute the prediction errors of these models using the training dataset and testing dataset separately.

```
# Compute the training and test errors for each model
mse <- sapply(fit, function(obj) deviance(obj)/nobs(obj))
pred <- mapply(function(obj, degf) predict(obj, data.frame(x = xy_
    _test$x)),
    fit, df)
te <- sapply(as.list(data.frame(pred)), function(y_hat) mean((xy_-
    test$y - y_hat)^2))
```

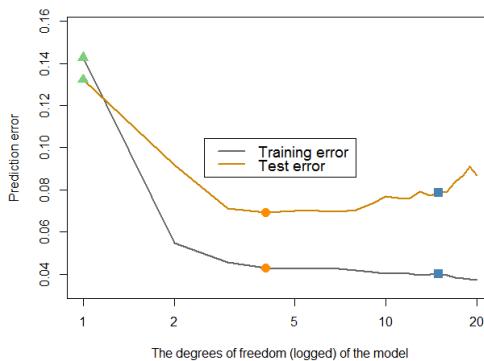


Figure 5.6: Prediction errors of the models (from $\text{df}=0$ to $\text{df}=20$) on the training dataset and testing data

The following R code is used to draw the Figure 5.6.

```
# Plot the errors
plot(df, mse, type = "l", lwd = 2, col = gray(0.4), ylab = "Predi-
    cation error",
    xlab = "The degrees of freedom (logged) of the model", ylim =
    c(0.9*min(mse), 1.1*max(mse)), log = "x")

lines(df, te, lwd = 2, col = "orange3")

points(df[1], mse[1], col = "palegreen3", pch = 17, cex = 1.5)
points(df[1], te[1], col = "palegreen3", pch = 17, cex = 1.5)
points(df[which.min(te)], mse[which.min(te)], col = "darkorange",
    pch = 16,
    cex = 1.5)
points(df[which.min(te)], te[which.min(te)], col = "darkorange",
```

```

pch = 16,
cex = 1.5)
points(df[15], mse[15], col = "steelblue", pch = 15, cex = 1.5)
points(df[15], te[15], col = "steelblue", pch = 15, cex = 1.5)
legend(x = "center", legend = c("Training error", "Test error"),
lwd = rep(2, 2),
col = c(gray(0.4), "orange3"), text.width = 0.3, cex = 1.2)

```

As we can see from Figure 5.6, the prediction error on the training dataset keeps decreasing with the increase of the `df`. This is consistent with our theory, and it is important to keep in mind that this triumph of model complexity doesn't really mean what it seems. It only indicates a universal phenomenon that a more complex model can fit the data better. On the other hand, we could observe that the testing error curve shows a U-shape curve, indicating an optimal model could be identified in the examined spectrum of model complexity.

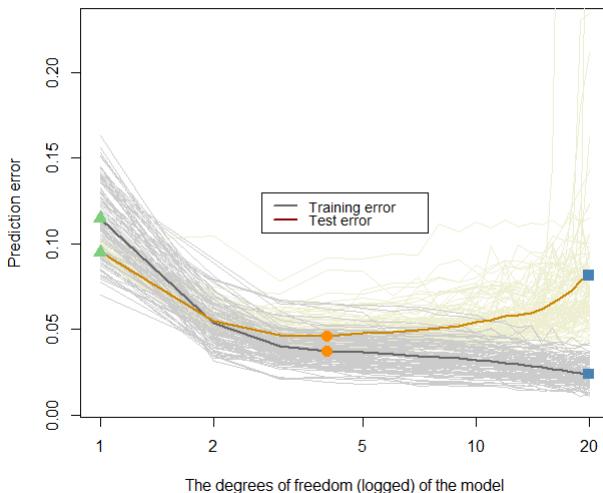


Figure 5.7: Prediction errors of the models (from `df=0` to `df=20`) on the training dataset and testing data of 100 replications. The two highlighted curves represent the mean curves of the 100 replications of the training and testing error curves, respectively

The following R code repeats this experiment 100 times and present the results in Figure 5.7.

```
# Repeat the above experiments in 100 times
n_rep <- 100
n_train <- 50
coef <- c(-0.68, 0.82, -0.417, 0.32, -0.68)
v_noise <- 0.2
n_df <- 20
df <- 1:n_df
xy <- res <- list()
xy_test <- gen_data(n_test, coef, v_noise)
for (i in 1:n_rep) {
  xy[[i]] <- gen_data(n_train, coef, v_noise)
  x <- xy[[i]][, "x"]
  y <- xy[[i]][, "y"]
  res[[i]] <- apply(t(df), 2, function(degf) lm(y ~ ns(x, df = degf)))
}
# Compute the training and test errors for each model
pred <- list()
mse <- te <- matrix(NA, nrow = n_df, ncol = n_rep)
for (i in 1:n_rep) {
  mse[, i] <- sapply(res[[i]], function(obj) deviance(obj)/nobs(obj))
  pred[[i]] <- mapply(function(obj, degf) predict(obj, data.frame(x = xy_test$x)),
                       res[[i]], df)
  te[, i] <- sapply(as.list(data.frame(pred[[i]])), function(y_hat) mean((xy_test$y -
    y_hat)^2))
}
# Compute the average training and test errors
av_mse <- rowMeans(mse)
av_te <- rowMeans(te)

# Plot the errors
plot(df, av_mse, type = "l", lwd = 2, col = gray(0.4), ylab = "Prediction error",
      xlab = "The degrees of freedom (logged) of the model", ylim =
      c(0.7*min(mse), 1.4*max(mse)), log = "x")
for (i in 1:n_rep) {
  lines(df, te[, i], col = "lightyellow2")
```

```

}
for (i in 1:n_rep) {
  lines(df, mse[, i], col = gray(0.8))
}
lines(df, av_mse, lwd = 2, col = gray(0.4))
lines(df, av_te, lwd = 2, col = "orange3")
points(df[1], av_mse[1], col = "palegreen3", pch = 17, cex = 1.5)
points(df[1], av_te[1], col = "palegreen3", pch = 17, cex = 1.5)
points(df[which.min(av_te)], av_mse[which.min(av_te)], col = "darkorange", pch = 16, cex = 1.5)
points(df[which.min(av_te)], av_te[which.min(av_te)], col = "darkorange", pch = 16, cex = 1.5)
points(df[20], av_mse[20], col = "steelblue", pch = 15, cex = 1.5)
points(df[20], av_te[20], col = "steelblue", pch = 15, cex = 1.5)
legend(x = "center", legend = c("Training error", "Test error"),
lwd = rep(2, 2),
  col = c(gray(0.4), "darkred"), text.width = 0.3, cex = 0.8
5)

```

Next, let's see how well the cross-validation could help to overcome the danger of overfitting. Let's consider the scenario that only the 100 samples are provided to us for both model training and testing. Thus, we use the 10-folder cross-validation on the 100 samples, using the following R code, to train the model.

```

# Cross-validation
set.seed(seed)

n_train <- 100
xy <- gen_data(n_train, coef, v_noise)
x <- xy$x
y <- xy$y

fitted_models <- apply(t(df), 2, function(degf) lm(y ~ ns(x, df = degf)))
mse <- sapply(fitted_models, function(obj) deviance(obj)/nobs(obj))

n_test <- 10000
xy_test <- gen_data(n_test, coef, v_noise)
pred <- mapply(function(obj, degf) predict(obj, data.frame(x = xy_test$x)),
  fitted_models, df)
te <- sapply(as.list(data.frame(pred)), function(y_hat) mean((xy_
```

```

test$y - y_hat)^2))

n_folds <- 10
folds_i <- sample(rep(1:n_folds, length.out = n_train))
cv_tmp <- matrix(NA, nrow = n_folds, ncol = length(df))
for (k in 1:n_folds) {
  test_i <- which(folds_i == k)
  train_xy <- xy[-test_i, ]
  test_xy <- xy[test_i, ]
  x <- train_xy$x
  y <- train_xy$y
  fitted_models <- apply(t(df), 2, function(deg) lm(y ~ ns(x, df = deg)))
  x <- test_xy$x
  y <- test_xy$y
  pred <- mapply(function(obj, deg) predict(obj, data.frame(ns(x, df = deg))), fitted_models, df)
  cv_tmp[k, ] <- sapply(as.list(data.frame(pred)), function(y_hat)
mean((y -
y_hat)^2))
}
cv <- colMeans(cv_tmp)

```

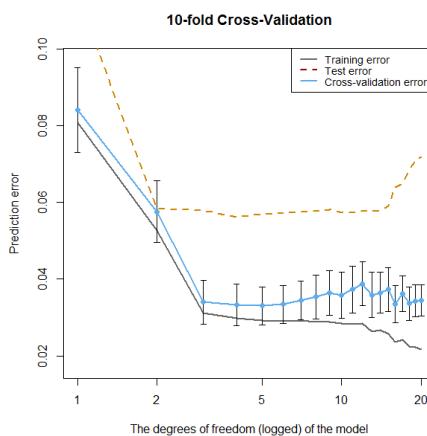


Figure 5.8: Prediction errors of the models (from $\text{df}=0$ to $\text{df}=20$) on the training dataset without cross-validation, on the training dataset using 10-folder cross-validation, and testing data of 50 samples.

Then we can visualize the result in Figure 5.8. Note that, in Figure 5.8, we overlay the result of the 10-folder cross-validation (based on the 100 samples) with the prediction error on a separate testing dataset (extra 50 samples) to get an idea how close the 10-folder cross-validation can match the ideal case with an extra batch of testing data.

```
# install.packages("Hmisc")
require(Hmisc)

plot(df, mse, type = "l", lwd = 2, col = gray(0.4), ylab = "Prediction error",
      xlab = "The degrees of freedom (logged) of the model", main =
      = paste0(n_folds,
               "-fold Cross-Validation"), ylim = c(0.8*min(mse),
               1.2*max(mse)), log = "x")
lines(df, te, lwd = 2, col = "orange3", lty = 2)
cv_sd <- apply(cv_tmp, 2, sd)/sqrt(n_folds)
errbar(df, cv, cv + cv_sd, cv - cv_sd, add = TRUE, col = "steelblue2",
       pch = 19,
       lwd = 0.5)
lines(df, cv, lwd = 2, col = "steelblue2")
points(df, cv, col = "steelblue2", pch = 19)
legend(x = "topright", legend = c("Training error", "Test error",
      "Cross-validation error"),
      lty = c(1, 2, 1), lwd = rep(2, 3), col = c(gray(0.4), "darkred",
      "steelblue2"),
      text.width = 0.4, cex = 0.85)
```

As shown in Figure 5.8, the 10-folder cross-validation could generate fair evaluation of the models just like an independent unseen testing dataset. Although its estimation of the error is smaller than the error estimation on the testing dataset, it could capture the change point of model complexity beyond which the model starts to overfit the data. Thus, it could be used to identify a good model that fits the data well, without overfitting.

Now let's apply the idea of 10-folder cross-validation on the AD data, for building a linear regression model with the demographic variables.

```
#### Dataset of Alzheimer's Disease
#### Objective: prediction of diagnosis
# filename
AD <- read.csv('AD.bl.csv', header = TRUE)
str(AD)
```

```

# fit a model with demographics only
lm.AD_demo <- lm(MMSCORE ~ AGE + PTGENDER + PTEDUCAT, data = AD)
summary(lm.AD_demo)

n_folds <- 10
folds_i <- sample(rep(1:n_folds, length.out = dim(AD)[1]))
cv_tmp <- matrix(NA, nrow = n_folds, 1)
cv_err <- matrix(NA, nrow = 50*n_folds, 2)
for (k in 1:n_folds) {
  test_i <- which(folds_i == k)
  train_xy <- AD[-test_i, ]
  test_xy <- AD[test_i, ]
  y <- test_xy$MMSCORE
  lm.AD_demo <- lm(MMSCORE ~ AGE + PTGENDER + PTEDUCAT, data = train_xy)
  pred <- predict(lm.AD_demo, test_xy)
  cv_tmp[k] <- mean((y - pred)^2)
  temp <- y - pred
  cv_err[(1+((k-1)*50)):(k*50),1] = rep(k, 50)
  cv_err[(1+((k-1)*50)):(k*50),2] <- temp[1:50]
}

```

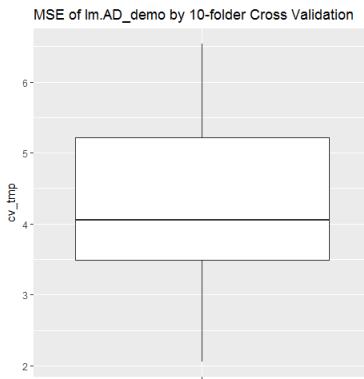


Figure 5.9: Prediction errors of the linear regression model using 10-folder cross-validation

We can use the boxplot to draw the distribution of the prediction errors (evaluated by MSE) collected by the 10-folder cross-validation, shown in Figure 5.9:

```
library(ggplot2)
p <- ggplot(data.frame(cv_tmp), aes(x= factor(""), y=cv_tmp)) + geom_boxplot() + xlab("") ## box plot
p <- p + labs(title="MSE of lm.AD_demo by 10-folder Cross Validation")
print(p)
```

Further, while it is not usual in applications to see the prediction errors within the folders, here, it is of interest to present this intermediate result to gain a visual understanding of cross-validation. The R code below draws the boxplots of the prediction errors of the 10 folders, shown in Figure 5.10.

Visualize the distributions of the prediction errors in the folders

```
cv_err <- data.frame(cv_err)
names(cv_err) = c("Folder", "Error")
ggplot(data = cv_err, aes(x = Folder, y = Error)) +
  geom_boxplot(aes(colour=factor(Folder)), fill=NA) +
  geom_point(aes(color = factor(Folder)))
```

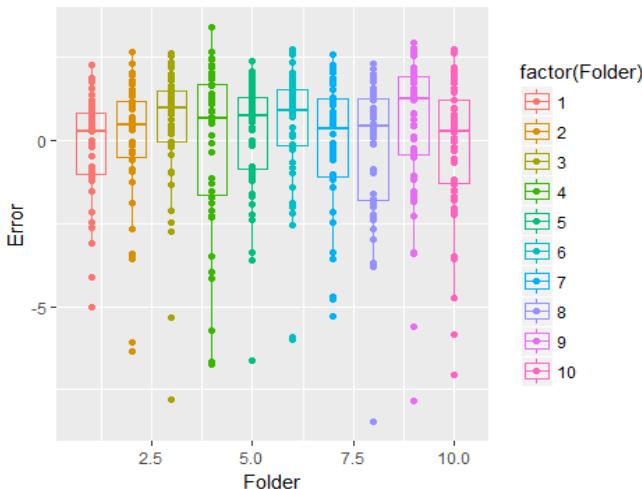


Figure 5.10: Prediction errors of the linear regression model using 10-fold cross-validation; each boxplot corresponds to one folder

II.4 Remarks

More about cross-validation: Usually, there is a relationship between the performance of the model on training dataset and its performance on

testing dataset, as shown in Figure 5.11. Note that this relationship is theoretical, but has very high relevance with real applications. In our experiments, as shown in Figures 5.6 and 5.7, we have seen this relationship. This relationship predicts that, while the performance on the training data will decrease if we increase the model complexity, at a certain point, the gain on performance by increasing model complexity will stop. Beyond this point, the performance would be worse. Thus, a model that has a good performance on the training data and a reasonable complexity is likely to be among the best models that will perform well on the testing data (unseen).

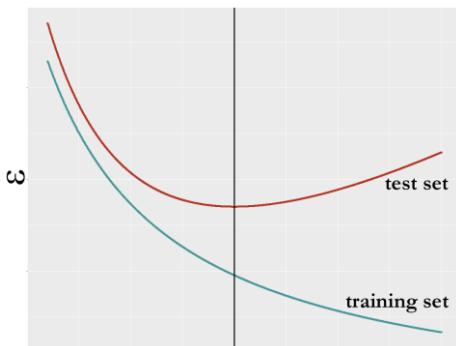


Figure 5.11: A theoretical relationship between the performance of the model on training dataset and its performance on testing dataset

The ROC curve. While cross-validation is useful for estimating the model's performance on unseen testing data, it still needs evaluation metrics to evaluate the model's performance. In some applications such as the rare disease example mentioned earlier in this chapter, how to evaluate the performance of a model itself could be a complex issue.

There have been many performance metrics developed in the literature. An important one, for classification problem, is the ROC curve. As we have seen the limitation of merely using accuracy as the performance metric of a classification model, the ROC has been commonly used as a better metric. The ROC stands for **Receiver Operating Characteristics**. As in a binary

classification problem that there are two classes, we often care about accuracies of prediction on both classes. If the classification problem is in a medical application, one class represents disease (positive) while another one represents normal (negative), then we may further name the correct prediction on a positive case as **true positive** (TP) and name the correct prediction on a negative case as **true negative** (TN). Correspondingly, we can define the **false positive** (FP) as incorrect prediction on a positive case and **false negative** (FN) as incorrect prediction on a negative case. This is summered in the following table:

Table 5.1: The confusion matrix

The confusion matrix		Reality	
		Positive	Negative
Model Prediction	Positive	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

Now, recall that, in a logistic regression model, before we reach the endpoint of the model that is binary prediction, we obtain the intermediate result $p(\mathbf{x}) = \frac{1}{1+e^{-(\beta_0 + \sum_{i=1}^p \beta_i x_i)}}$. Then, a cut-off value (e.g., 0.5) is used to classify the cases whose $p(\mathbf{x})$ is larger than the cut-off value as one class and otherwise as another class. This means that, for each cut-off value, we can obtain a confusion matrix with different values on the TP and FP. This is shown in Figure 5.12.

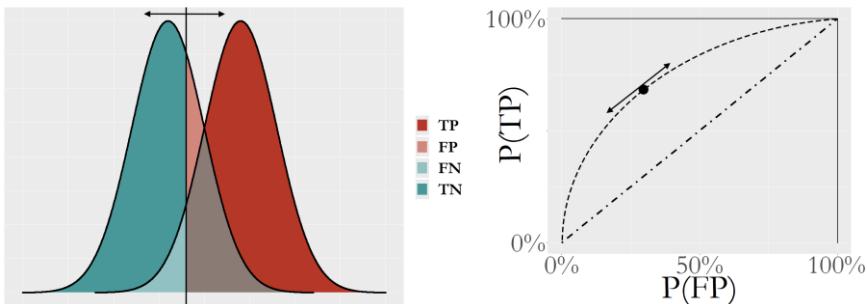


Figure 5.12: For a logistic regression model of two classes, the logistic model can produce the intermediate results $p(\mathbf{x})$ for the cases of both classes. (a) shows the distributions of $p(\mathbf{x})$ of both classes and a particular cut off value; (b) shows the corresponding confusion matrix; (c) shows the ROC curve that synthesizes all the scenarios of all cut off values

As we can see from Figure 5.12, the ROC curve is a succinct way to synthesizes all the scenarios of all cut-off values. Thus, it provides a more holistic way to evaluate a model (actually, more about to evaluate the potential of a model). A model that lacks potential for prediction will be close to the 45° line, representing random guess on both classes. A better model will show a ROC curve that is closer to the upper left corner point.

Based on ROC, a metric that is named the **AUC** (the area under the curve) is proposed to summarize the ROC curve of a model. The higher the AUC, the better the model.

In what follows we show how to derive these performance metrics using the logistic regression model. First, let's build a logistic regression model using the AD data as what we have done in Chapter 3.

```
# ROC and more performance metrics of Logistic regression model
# Load the AD dataset
AD <- read.csv('AD_b1.csv', header = TRUE)
str(AD)
```

```

# Split the data into training and testing sets
n = dim(AD)[1]
n.train <- floor(0.8 * n)
idx.train <- sample(n, n.train)
AD.train <- AD[idx.train,]
AD.test <- AD[-idx.train,]

# Automatic selection of the model
logit.AD.full <- glm(DX.bl ~ ., data = AD.train[,c(1:16)], family
= "binomial")
logit.AD.final <- step(logit.AD.full, direction="both", trace = 0)
summary(logit.AD.final)

```

Then, we can use the function, `confusionMatrix()` from the R package “`e1071`” to derive the performance metrics:

```

# install.packages("e1071")
require(e1071)

require(caret)

# Prediction scores
pred = predict(logit.AD.final, newdata=AD.test,type="response")
confusionMatrix(data=factor(pred>0.5), factor(AD.test[,1]==1))

```

The results are shown in below:

```

## Confusion Matrix and Statistics
##
##          Reference
## Prediction  FALSE TRUE
##   FALSE      48    7
##   TRUE       7   42
##
##                  Accuracy : 0.8654
##                         95% CI : (0.7845, 0.9244)
##   No Information Rate : 0.5288
##   P-Value [Acc > NIR] : 3.201e-13
##
##                  Kappa : 0.7299
##   Mcnemar's Test P-Value : 1
##
##                  Sensitivity : 0.8727
##                  Specificity : 0.8571
##   Pos Pred Value : 0.8727
##   Neg Pred Value : 0.8571
##   Prevalence : 0.5288

```

```

##           Detection Rate : 0.4615
##   Detection Prevalence : 0.5288
##   Balanced Accuracy : 0.8649
##
##   'Positive' Class : FALSE
##

```

The ROC curve could be drew using the R Package “**ROCR**”:

```

# Generate the ROC curve using the testing data
# Compute ROC and Precision-Recall curves
require('ROCR')

linear.roc.curve <- performance(prediction(pred, AD.test[,1]),
                                    measure='tpr', x.measure='fpr' )
plot(linear.roc.curve, lwd = 2, col = "orange3",
      main = "Validation of the logistic model using testing data")

```

The ROC curve is shown in Figure 5.13.

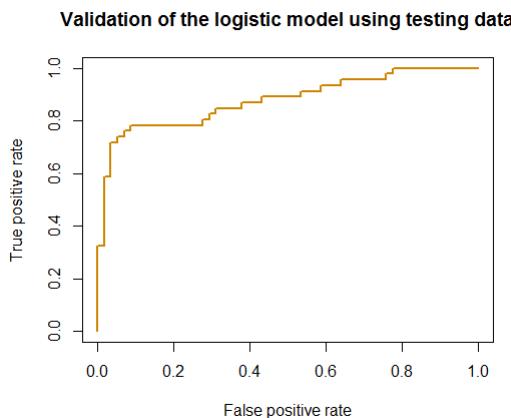


Figure 5.13: ROC curve of the logistic regression model

III. Out-of-bag error in Random Forest

III.1 Rationale and Formulation

The out-of-bag (OOB) error in a random forest model provides a computationally convenient approach to evaluate the model without using a

testing dataset, neither a cross-validation procedure. Recall that, for a random forest model with K trees, each tree is built on a bootstrapped dataset from the original training set S . There are totally K bootstrapped datasets, denoted as B_1, B_2, \dots, B_K .

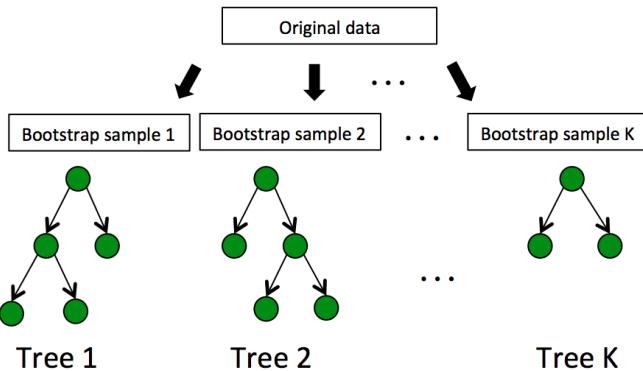


Figure 5.14: The framework of random forest

As the size of each bootstrapped dataset is the same size (denoted as N) as the original training data, and each data point in the bootstrapped dataset is selected independently from other data points, therefore, the probability of a data point from the training data is missing from a bootstrapped dataset is

$$\left(1 - \frac{1}{N}\right)^N.$$

When N is sufficiently large, we can have

$$\lim_{N \rightarrow \infty} \left(1 - \frac{1}{N}\right)^N = e^{-1} \approx 0.37.$$

Therefore, roughly 37% of the data points from S are not contained in any bootstrapped dataset B_i , and thus, not used for training tree i . These excluded data points are referred as the out-of-bag samples for the bootstrapped dataset B_i and tree i . Note that when N is small, the

probability of a data point missing from a bootstrapped dataset is smaller, e.g., the probability becomes 0 when $N = 1$, and $1/4$ when $N = 2$.

As there are 37% of probability that a data point is not used for training a tree, we can infer that, a data point is not used for training about 37% of the trees. Therefore, for each data point, in theory, there are 37% of trees trained without this data point. These trees can be used to predict on this data point, which can be considered as testing an unseen data point. The out-of-bag error estimation can then be calculated by aggregating the out-of-bag testing error of all the data points. The out-of-bag error can be calculated after random forests are built, and are significantly less computationally than cross-validation. Note that the out-of-bag estimates are calculated by 37% of the trees in the random forest model, therefore, it is expected that the full random forest model with K trees would perform better on any data point than a subset of trees. However, as the performance of the random forest model stabilizes as the number of trees increases, the difference may be small when K is sufficiently large.

Suppose that we have a training dataset of 5 instances (IDs as 1,2,3,4,5). Three trees are built using three bootstrapped datasets, as shown in the Table 5.2.

Table 5.2: Three trees and the corresponding bootstrapped datasets

Bootstrap	Tree
1,1,4,4,5	1
2,3,3,4,4	2
1,2,2,5,5	3

Then, we can estimate the out-of-bag (OOB) errors as shown in Table 5.3 (the true classes of the instances are shown in the top row):

Table 5.3: The out-of-bag (OOB) errors

Tree	Training data	1 (C1)	2 (C2)	3 (C2)	4 (C1)	5 (C2)
1	1,1,4,4,5		C1	C2		
2	2,3,3,4,4	C1				C2
3	1,2,2,5,5			C2	C1	

We can see that, as the data instance (ID = 1) is not used in training Tree 2, we can use Tree 2 to predict on this data instance, and we see that it correctly predict the class as C1. Similarly, Tree 1 is used to predict on data instance (ID=2), and the prediction is wrong. Finally we can see that the overall out-of-bag (OOB) error is 1/6.

III.3 R Lab

We apply the random forest model to the AD dataset and the out-of-bag (OOB) error can be obtained. Random forests are run 50 times. Separately, we use 63/100 of the data for training random forests and use the rest 37/100 data to get testing error (referred to as the validation error), so the proportion of the testing instances similar to the random forests OOB samples. This is repeated 50 times as well. The error rates from both methods are plotted in the boxplots in Figure 5.15. It can be seen that the average error from both methods are similar, but the OOB error seems to have less variance compared to the validation error.

```
library(ggplot2)
require(randomForest)
set.seed(1)

theme_set(theme_gray(base_size = 15))

path <- "../data/AD_b1.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_b1")
target <- paste0("class_", as.character(data[, target_indx]))
rm_indx <- which(colnames(data) %in% c("DX_b1", "ID", "TOTAL13",
"MMSCORE"))
X <- data
X <- X[, -rm_indx]
```

```

err.mat <- NULL
for (i in 1:50) {
  rf <- randomForest(X, as.factor(target))
  err.mat <- rbind(err.mat, c("OOB_error", mean(rf$err.rate[, "OOB"])))
}
for (i in 1:50) {
  train.ix <- sample(nrow(X), floor(63 * nrow(X)/100))
  rf <- randomForest(X[train.ix, ], as.factor(target[train.ix]))
  pred.test <- predict(rf, X[-train.ix, ], type = "class")
  err.mat <- rbind(err.mat, c("validation_error", length(which(pred.test != target[-train.ix]))/length(pred.test)))
}

err.mat <- as.data.frame(err.mat)
colnames(err.mat) <- c("error_type", "error")
err.mat$error = as.numeric(as.character(err.mat$error))
err.mat$error_type <- factor(err.mat$error_type)

ggplot() + geom_boxplot(data = err.mat, aes(x = error_type, y = error)) + geom_point(size = 3)

```

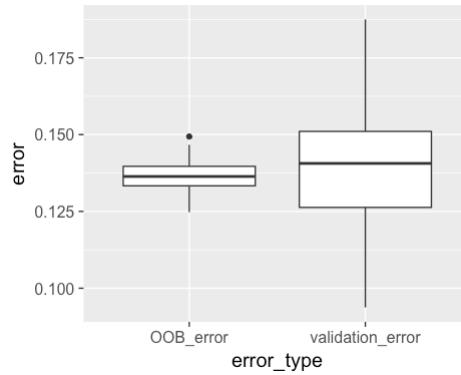


Figure 5.15: Boxplots of the OOB error and validation error

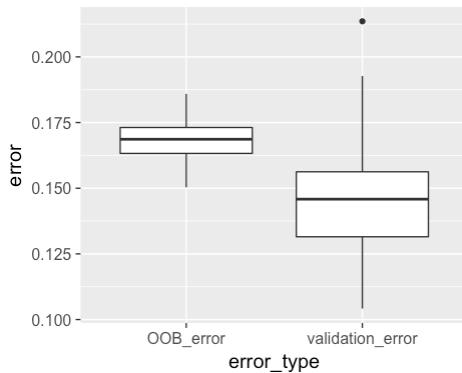


Figure 5.16: Boxplots of the OOB error and validation error

In the previous experiment, the number of trees in the random forest models is the default of 500. Since we expect that, when the number of trees is small, 37% of the trees may not have larger errors than using all the trees. Here, we re-run the previous experiment, but with the number of trees as 50. The OOB errors and validation errors are plotted in Figure 5.16. As expected, the OOB error is clearly larger than the validation error, since only around $37\% * 50$ trees are used for prediction.

```

err.mat <- NULL
for (i in 1:50) {
  rf <- randomForest(X, as.factor(target), ntree = 50)
  err.mat <- rbind(err.mat, c("OOB_error", mean(rf$err.rate[, "OOB"])))
}
for (i in 1:50) {
  train.ix <- sample(nrow(X), floor(63 * nrow(X)/100))
  rf <- randomForest(X[train.ix, ], as.factor(target[train.ix]),
  ntree = 50)
  pred.test <- predict(rf, X[-train.ix, ], type = "class")
  err.mat <- rbind(err.mat, c("validation_error", length(which(
  (pred.test != target[-train.ix])))/length(pred.test))))
}

err.mat <- as.data.frame(err.mat)

```

```

colnames(err.mat) <- c("error_type", "error")
err.mat$error = as.numeric(as.character(err.mat$error))
err.mat$error_type <- factor(err.mat$error_type)

ggplot() + geom_boxplot(data = err.mat, aes(x = error_type, y = error)) + geom_point(size = 3)

```

IV. Exercises

Data analysis

1. Find ten classification datasets from the UCI data repository or R datasets. Using these datasets, conduct experiments to see if the cross-validation method on training data can provide an approximation of the testing error on a testing data, as shown in Figure5.8.
2. Using the datasets you picked up in 1, use cross-validation to select the best logistic regression model, the best decision tree model, and the best random forest model. Compare the models.
3. Using these datasets, build random forest models and compare the OOB error rates from the random forest models with 10-folder cross-validation error.

Programming

In the book¹ by Prof. Cosma Rohilla Shalizi, an interesting experiment is proposed to show another disadvantage of the concept R-squared. Three simple datasets are simulated via:

1. Simulate 100 data points of predictor X from a uniform distribution $Unif(0,1)$; then, simulate 100 corresponding values of response variable Y from $N(\sqrt{X}, 0.05^2)$.

¹ Shalizi, C.R. *Advanced data analysis from an elementary point of view*. Book Manuscript: <http://www.stat.cmu.edu/~cshalizi/ADAdaEPoV/ADAdaEPoV.pdf>.

2. Simulate 100 data points of predictor X from $N(0.5, 0.1^2)$; then, simulate 100 corresponding values of response variable Y from $N(\sqrt{X}, 0.05^2)$.
3. Simulate 100 data points of predictor X from a uniform distribution $Unif(2,3)$; then, simulate 100 corresponding values of response variable Y from $N(\sqrt{X}, 0.05^2)$.

Built three regression models on the three datasets. Comment on the R-squareds of the three fitted models (which aim to fit the same regression model anyway).

CHAPTER6: DIAGNOSIS RESIDUALS AND HETEROGENEITY

I. Overview

Chapter 5 is about “**Diagnosis**”. Diagnosis, in one sense, is to see if the assumptions that determine the theoretical validity of the model fit the empirical characteristics of the data. For example, when we use linear regression model, we use a whole set of subsequent methods such as the t-test and F-test to gain more understanding of the model, while these methods are built on the assumptions such as the normality of the errors. Identification of assumption violation certainly indicates some concerns, limiting the strength of our conclusion, but doesn’t mean the model is not useful. The model is still useful, telling part of the truth. Actually, the model, together with the potential gap between the theoretical assumptions and the empirical data characteristics, should be taken as a whole, that jointly form the analytics practice. Many diagnostic tools are developed for maintaining a critical attitude towards the models which are essentially artificial representations/approximations of the reality, yet there is a big difference between being critical and being dismissive. An even more radical assertion

was once pointed out in the seminar book¹ that even a model that doesn't fit generates knowledge, revealed not by the failed model but by the misfit of the model as a fact.

II. Residual Analysis in Regression

In this chapter, we take a pragmatism approach to present some of these concepts, by combining the background, theory, and R lab into one section.

Residual analysis: Diagnosis of regression models have been theorized and well articulated in a few monographs. Many interesting concepts have been developed in the literature, such as the **multicollinearity**, **heteroscedasticity**, **cook's distance**, **leverage**, and **Q-Q plot**, to name a few.

Let's use the final regression model we identified in Chapter 2 for an example. The following R code reproduces this final model:

```
AD <- read.csv('AD.bl.csv', header = TRUE)
AD$ID = c(1:dim(AD)[1])
str(AD)

# fit a full-scale model
AD_full <- AD[,c(1:16)]
lm.AD <- lm(MMSCORE ~ ., data = AD_full)
summary(lm.AD)

# Automatic model selection
lm.AD.F <- step(lm.AD, direction="backward", test="F")
```

The returned final model is summarized by calling the function **summary()**.

```
## MMSCORE ~ PTEDUCAT + FDG + AV45 + HippoNV + rs744373 + rs61093
2 +
##      rs3764650 + rs3865444
##
```

¹ Jaynes, E.T. *Probability theory: the logic of science*. Cambridge Press, 2003.

```

##          Df Sum of Sq    RSS      AIC F value    Pr(>F)
## <none>            1537.5 581.47
## - rs3764650  1     7.513 1545.0 581.99  2.4824  0.115750
## - rs744373   1    12.119 1549.6 583.53  4.0040  0.045924 *
## - rs610932   1    14.052 1551.6 584.17  4.6429  0.031652 *
## - rs3865444   1    21.371 1558.9 586.61  7.0612  0.008125 **
## - AV45        1    50.118 1587.6 596.05 16.5591 5.467e-05 ***
## - PTEUCAT     1    82.478 1620.0 606.49 27.2507 2.610e-07 ***
## - HippoNV     1   118.599 1656.1 617.89 39.1854 8.206e-10 ***
## - FDG         1   143.852 1681.4 625.71 47.5288 1.614e-11 ***
## ...
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

The R package “`ggfortify`” provides a nice graphic bundle to show some important diagnostic figures.

```

# Conduct diagnostics of the model
# install.packages("ggfortify")
library("ggfortify")

autoplot(lm.AD.F, which = 1:6, ncol = 3, label.size = 3)

```

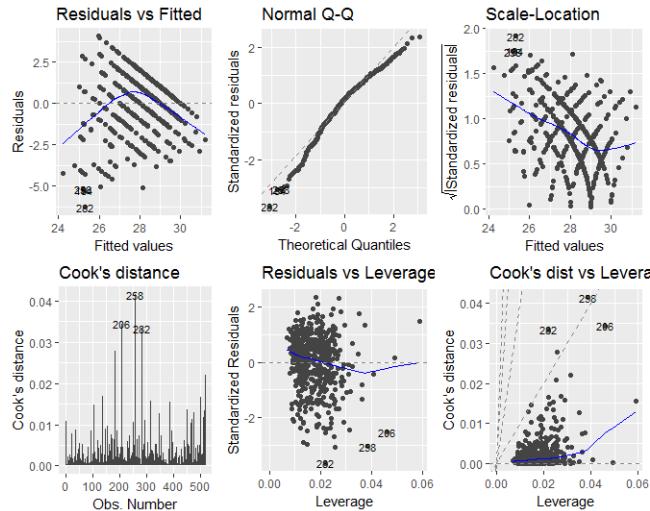


Figure 6.1: Diagnostic figures of regression model on the AD dataset

The diagnostic figures shown in Figure 6.1 are information-rich. One way to interpret them is to put them into a contrast with the way they suppose to be. For example, for the left figure in the first row, which is the scatterplot of the residuals versus fitted values of the outcome variable, it is supposed to show purely random distributions of the data points. In other words, any pattern that shows non-random characteristics, such as the curved relationship between the residuals and fitted values, and the unusual parallel lines of the data points, indicates deviance from the assumptions such as independence of the observations and constancy/homoscedasticity of the variance of the errors.

The Q-Q plot, as the middle figure in the first row, shows violation of the normality assumption of the error term. And some particularly violating data points such as the data points 282 and 256 are labelled.

The Cook's distance shown in the left figure in the second row, shows the influential data points that have larger than average influence on the parameter estimation. The Cook's distance of a data point is built on the idea of how much change will be induced on the estimated parameters if the data point is deleted.

The leverage of a data point, on the other hand, shows the influence of the data point in another way. Mathematically, the leverage of a data point is $\frac{\partial \hat{y}_i}{\partial y_i}$, reflecting how sensitive the prediction on the data point by the model is decided by the observed outcome value y_i . In other words, what data point will result in high leverage value? For data points that are surrounded by many close-by data points, their leverages won't be large, since the impact of removal of them will be compensated by other similar data points in the nearby. Thus, we could infer that the data points that sparsely occupy their neighbor areas will have large leverages. These data points could either be outliers that severely deviate from the linear trend represented by the majority of the data points, or could be valuable data points that align with the linear trend but lack neighbor data points, and

thus, changes on their observations will generate a large impact on the predictions on the data points nearby their locations. Thus, it is important to note that, a data point that is influential doesn't necessary imply that it is bad. It only suggests that some more in-depth examination of the data point is needed.

While the information shown in Figure 6.1 is telling, we don't know how bad it is. In other words, we need a baseline version of these figures to establish an expectation so we can compare Figure 6.1 with. To do so, we can simulate a dataset while all the assumptions of the linear regression model are met, to get a sense what these diagnostic figures would look like. The R code in below shows how we can simulate a dataset with 100 samples from the regression model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon, \quad \varepsilon \sim N(0,1).$$

```
# For comparison, let's simulate data
# from a model that fits the assumptions
x1 <- rnorm(100, 0, 1)
x2 <- rnorm(100, 0, 1)
beta1 <- 1
beta2 <- 1
mu <- beta1 * x1 + beta2 * x2
y <- rnorm(100, mu, 1)
lm.XY <- lm(y ~ ., data = data.frame(y,x1,x2))
summary(lm.XY)
```

We can see that the fitted model fairly reflects the underlying model.

```
##
## Call:
## lm(formula = y ~ ., data = data.frame(y, x1, x2))
##
## Residuals:
##      Min      1Q  Median      3Q     Max
## -2.6475 -0.6630 -0.1171  0.7986  2.5074
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.0366   0.1089   0.336   0.738
## x1          0.9923   0.1124   8.825 4.60e-14 ***
## x2          0.9284   0.1159   8.011 2.55e-12 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.088 on 97 degrees of freedom
## Multiple R-squared:  0.6225, Adjusted R-squared:  0.6147
## F-statistic: 79.98 on 2 and 97 DF,  p-value: < 2.2e-16
```

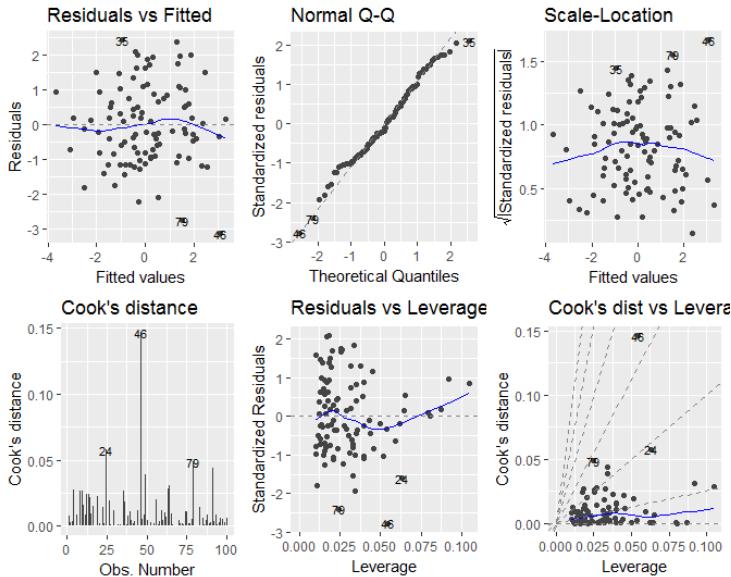


Figure 6.2: Diagnostic figures of regression model on a simulation dataset

Then, we can generate the same set of diagnostic figures. Many interesting contrasts could be observed. For example, from the left figure on the first row in Figure 6.2, we can see that, different from the one in Figure 6.1, now we don't see any significant non-random statistical pattern. The relationship between the residual and fitted values seems to be null. From the QQ-plot, we can also see that the normality assumption is held well. On the other hand, from the cook's distance and the leverage plot, some data points are observed to be influential just as what we can observe from Figure 6.1. As we know that we have simulated the data strictly

following the assumptions of the linear regression model, this experiment shows that it is normal to expect some data points exhibiting abnormality according to the cook's distance and the leverage.

```
# Conduct diagnostics of the model
library("ggfortify")
autoplot(lm.XY, which = 1:6, ncol = 3, label.size = 3)
```

Multicollinearity analysis: The diagnostic figures shown above are all about diagnosis on the data points. This perspective of looking at the dataset point by point (vertically) is the conventional way to define model diagnostics. There is another perspective which is to look at the dataset variable by variable (horizontally). In regression model, the problem of multicollinearity has been well known as a serious problem. Multicollinearity refers to the phenomenon that many predictor variables highly correlate with each other, resulting in great ambiguity in the model parameter estimation due to the ill condition of the matrix $\mathbf{X}^T \mathbf{X}$, i.e., small changes on \mathbf{X} will result in large and unpredictable changes on the inverse matrix $\mathbf{X}^T \mathbf{X}$, which will eventually result in great instability of the parameter estimation in $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$.

We can do a simple analysis to study this problem of multicollinearity. Consider a system that generates the observation following the relationships shown in below:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \varepsilon, \quad \varepsilon \sim N(0, \sigma_\varepsilon^2),$$

$$x_1 = 2x_2 + \epsilon, \quad \epsilon \sim N(0, 0.1\sigma_\varepsilon^2)$$

The system has the symptom of multilinearity as two of the variables are highly correlated. Thus, theoretically, we could value the regression model that is shown in above as the ground truth model equally as we value the following models:

$$y = \beta_0 + (2\beta_1 + \beta_2)x_2 + \beta_3 x_3 \dots + \beta_p x_p,$$

$$y = \beta_0 + (\beta_1 + 0.5\beta_2)x_1 + \beta_3 x_3 + \cdots + \beta_p x_p,$$

$$y = \beta_0 + 1000x_1 + (\beta_2 + \beta_1 - 2000)x_2 + \beta_3x_3 + \cdots + \beta_px_p.$$

Thus, the problem of multilinearity creates this inherent ambiguity of the models that could be taken as faithful representation of how the data was generated. Consequently, it makes no sense that an estimation method, essentially as a reverse-engineering approach, that could recover the truth while the truth itself is ambivalent.

There are some methods that we can use to diagnose for multilinearity. As it is a condition that many variables are highly correlated with each other, we may present the correlations among the predictor variables. The R package “corrplot” is a package that has been widely used for visualizing correlation matrix.

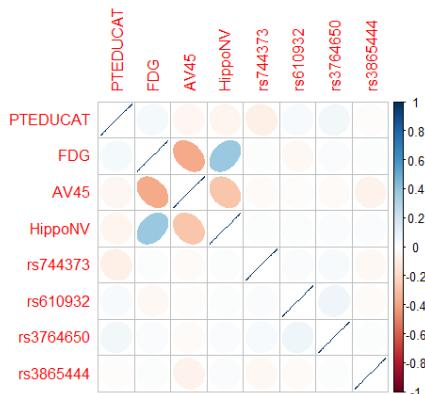


Figure 6.3: Correlations of the predictors in the regression model of **MMSCORE**

The following R code shows how to use “`corrplot`” to visualize the correlations among the predictors in the regression model we have built in Chapter 2 for predicting **MMSCORE**. Result is shown in Figure 6.3.

```
# Extract the covariance matrix of the regression parameters
Sigma = vcov(AD)
```

```
# Visualize the correlation matrix of the estimated regression parameters
# install.packages("corrplot")
library(corrplot)

corrplot(cov2cor(Sigma), method="ellipse")
```

From Figure 6.3 we could observe that there is significant correlations between the variables, **FDG**, **AV45**, and **HippoNV**, indicating a concern for multicollinearity. On the other hand, it seems that the correlations are only moderate, and not all the variables are densely correlated with each other.

We can further visualize the correlation matrix of the estimated regression parameter:

```
Sigma = vcov(lm.AD.F)

corrplot(cov2cor(Sigma), method="ellipse")
```

Then we can observe a similar pattern in Figure 6.4 as shown in Figure 6.3.

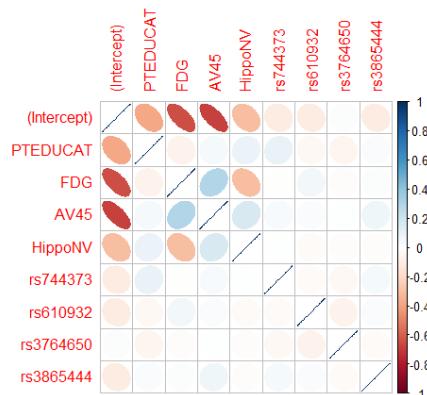


Figure 6.4: Correlations of the estimated parameters in the regression model of **MMSCORE**

The concern with multicollinearity, as we have discussed and illustrated using an analysis, is that it may result in unreliable model estimations, due to

the inherent ambiguity and instability in the numerical operations in least square estimation. The `corrplot` could help to visually check the data for multicollinearity, but it could not answer the question whether we should worry about the model we have built. To answer this question, we could further use Bootstrap to introduce perturbation into the data and see if the models fitted on different bootstrapped samples will change. Recall that we have done this in Chapter 4, we could draw the conclusion that the multicollinearity issue is not severe here in the AD dataset.

Ranking of variable: A related topic to the multicollinearity problem is the ranking of features. For example, the R package “`leaps`” implements the exhaustive evaluation of all subsets regression, i.e., try every combination of variables with a minimum number of features in the regression model. The results will show which models achieve highest R-squared value, and which variables frequently appear on these models.

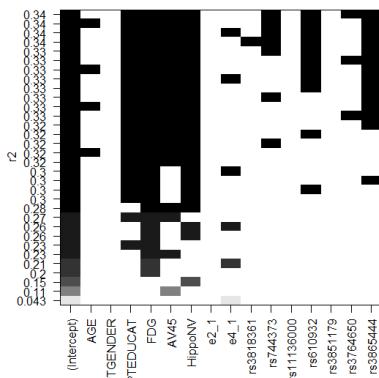


Figure 6.5: Regression models ranked by their R-squared and the constitutive features

The following R code implements this method on the AD dataset. Result is shown in Figure 6.5.

```
# Evaluate the variable importance by all subsets regression
# install.packages("Leaps")
library(leaps)

leaps<-regsubsets(MMSCORE ~ ., data = AD,nbest=4)
# view results
summary(leaps)

# plot a table of models showing variables in each model.
# models are ordered by the selection statistic.
plot(leaps,scale="r2")
```

From Figure 6.5 we could observe that, the variables, `PTEDUCAT`, `FDG`, `AV45`, and `HippoNV`, are most important features. And `rs3865444` shows moderate significance. Other variables show less significance.

III. Diagnosis in Random Forests

Random forests make few assumptions about the data. It handles mixed categorical and numerical variables, nonlinearities, and variable interactions in a more automatic way than data models such as linear regression. However, random forests are complex and consist of multiple weak trees that are hard to interpret. Cross-validation error or **out-of-bag (OOB)** error can be used to evaluate the random forests' model accuracy. However, to be able to trust random forests, it is desirable to extract interpretable insights from random forests. Here we introduce a few ways of diagnosis for random forest.

Variable Importance: Each variable's usefulness in predicting the outcome variable can be measured by the variable importance scores from random forests. There are two types of importance scores. The first one is the total decrease of node impurity across all tree nodes that are split by a variable (retrospectively). The second is measured by the accuracy decrease by permuting the variable (proactively).

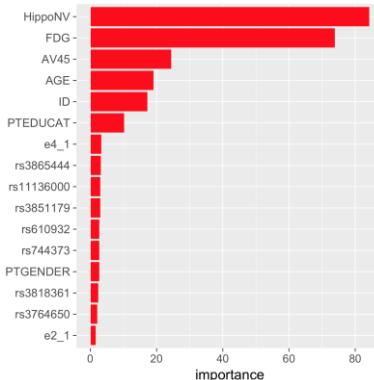


Figure 6.6: Important scores of variables in random forest

Here we plot the first type of importance score, the impurity gain importance score, from the random forest model built for the AD data. Result is shown in Figure 6.6.

```

library("RWeka")
library("randomForest")
library("RRF")
library("inTrees")
library("ggplot2")

path <- "../../data/AD_b1.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_b1")
rm_indx <- which(colnames(data) %in% c("DX_b1", "TOTAL13", "MMSCO
RE"))
rf <- randomForest(data[, -rm_indx], as.factor(data[, target_indx]))
imp <- as.data.frame(rf$importance)
colnames(imp)[colnames(imp) == "MeanDecreaseGini"] <- "importance
"
imp <- imp[order(imp$importance, decreasing = FALSE), , drop = FA
LSE]
imp$feature <- rownames(imp)
imp$feature <- factor(imp$feature, levels = as.character(imp$feat
ure))
theme_set(theme_gray(base_size = 18))
ggplot(data = imp, aes(x = feature, y = importance)) + geom_bar(s
tat = "identity",

```

```
aes(factor(feature)), fill = "red") + theme(axis.title.y = element_blank(),
axis.text.y = element_text(hjust = 1, size = 15)) + coord_flip()
```

Not a surprise, we can see from Figure 6.6 that the variables `HippoNV`, `FDG`, `AV45`, and `AGE`, are most important features in the random forest model. Interestingly, we can also see that the feature “`ID`” has the fourth largest importance score. In normal sense, this is a nuisance feature that is supposed to be random assignments of IDs to subjects. Thus, we might suspect that random forest is too powerful in extracting nonlinear patterns from data, thus it is tricked by noises in a dataset, a trade-off that we could overcome by more careful model selection and validation using cross-validation and in-depth analysis.

But on the other hand, this happens in many practical situations that some supposedly random assignments by humans are actually not the same as pure random noise, but rather encode some systematical patterns. As Prof. R.A. Fisher said, who is a pioneer in design of experiments (DOE) and modern statistics, “if one tries to think of numbers at random, one thinks of numbers very far from at random¹”. In addition, out-of-bag error doesn’t reduce significantly when the variable `ID` is removed. This may indicate that the creation of the `ID` for the subjects probably contained certain information about the subjects.

In some other applications, we also find that, if we create a binary variable to indicate the missing data instances in a feature, sometimes this binary indicator variable could be significant. Which means, the missing data itself as a fact is also informative to predict an outcome! This is not uncommon in healthcare applications. For instance, when a patient’s condition is severe, this patient may lack measurements of many clinical variables. Thus, the missing values of these variables provide valuable

¹ Fisher, R.A. Cigarettes, cancer, and statistics. *The Centennial Review of Arts & Science*, 1958.

information in predicting if the patient's condition is severe. This issue is also referred as variable **leakage** in machine learning.

Partial dependency plot: Variable importance scores indicate whether a variable is informative in predicting the outcome variable, but do not provide information about how the outcome variable is influenced by the variables. Partial dependency plot can be used to visualize the relationship between the variables of interest and the outcome variable, averaged on other variables. For the AD data, we apply the partial dependency plots to the top two important variables. It is clear that the relationships between the outcome variable with both predictor variables are significant. And we could also see the orientation of both relationships, i.e., the larger the HippoNV or FDG, the more likely that the subjects belong to normal (the class "normal" is coded as -1).

```
randomForest::partialPlot(rf, data, HippoNV, "1")
randomForest::partialPlot(rf, data, FDG, "1")
```

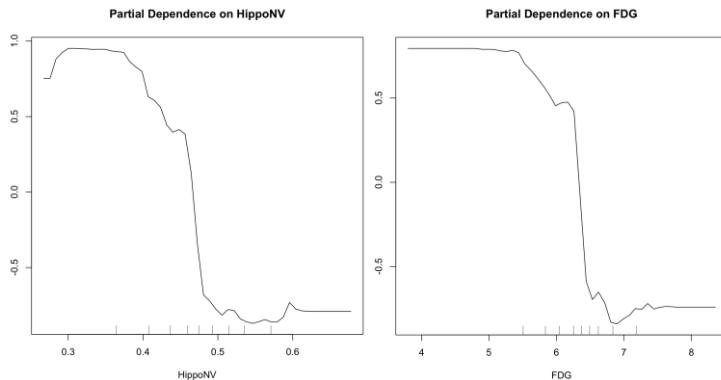


Figure 6.7: Partial dependency plots of variables in random forest

inTrees

Partial dependence plots provide how predictor variables interact with the outcome variable, while this interaction effect is averaged on other variables. However, it is difficult to visualize the synergistic effect of

multiple variables on the outcome variable, and a more quantitative approach can be desirable at times. The inTrees framework can be used for this purpose, which will be discussed in detail in Chatper 10. In the framework of inTrees, rules can be extracted, cleaned, and summarized from random forests.

```
treeList <- RF2List(rf) # transform rf object to an inTrees' for mat
exec <- extractRules(treeList, data[, -rm_idx]) # R-executable conditions

## 3695 rules (length<=6) were extracted from the first 100 trees.

class <- paste0("class_", as.character(data[, target_idx]))
rules <- getRuleMetric(exec, data[, -target_idx], class)
rules <- pruneRule(rules, data[, -target_idx], class)
rules <- selectRuleRRF(rules, data[, -target_idx], class)
rules <- presentRules(rules, colnames(data[, -target_idx]))
```

Here are the rules from inTrees applied to the AD data. `len` indicates the number of variable-value pairs in the condition of a rule, `freq` is the percentage of instances satisfying the condition, and `err` is the error rate of the rule. Without the need to choose which variables we would like to study, the selected rules from inTrees indicate all the important variable interactions it could identify, i.e., in the generated rules shown below, it shows how HippoNV and FDG interact with the outcome variable.

```
##      len freq     err
## [1,] "2" "0.3"   "0.0060000000000001"
## [2,] "2" "0.503" "0.115"
## [3,] "3" "0.464" "0.125"
## [4,] "2" "0.373" "0.114"
## [5,] "2" "0.335" "0.04"
## [6,] "3" "0.286" "0.0679999999999999"
## [7,] "3" "0.114" "0.0679999999999999"
## [8,] "3" "0.209" "0.074"
## [9,] "3" "0.176" "0.088"
## [10,] "4" "0.023" "0.333"
## [11,] "4" "0.017" "0"
## [12,] "4" "0.11"  "0.035"
## [13,] "4" "0.031" "0.125"
## [14,] "3" "0.099" "0.118"
##      condition
```

```

## [1,] "FDG<=6.35981 & HippoNV<=0.47428125"
## [2,] "FDG>6.323505 & HippoNV>0.401237706"
## [3,] "PTEDUCAT>12.5 & AV45<=1.5079 & HippoNV>0.463772954"
## [4,] "FDG<=6.464415 & HippoNV<=0.4766025375"
## [5,] "FDG>6.35679 & HippoNV>0.4764150235"
## [6,] "FDG>6.29287 & HippoNV>0.406667579 & rs3851179>0.5"
## [7,] "AV45>1.17371 & HippoNV<=0.4713683765 & rs3851179<=0.5"
## [8,] "FDG>5.69764 & HippoNV>0.4784356305 & rs3865444<=0.5"
## [9,] "FDG>6.513695 & AV45>1.011585 & e4_1<=0.5"
## [10,] "FDG<=6.30134 & AV45<=1.25324 & HippoNV>0.507448786 & e4_1>0.5"
## [11,] "PTEDUCAT<=12.5 & AV45>1.103075 & AV45<=1.1183375 & e4_1<=0.5"
## [12,] "AV45>1.057595 & AV45<=1.740035 & HippoNV<=0.421919192 & rs744373<=0.5"
## [13,] "AGE<=69.55 & FDG<=6.30834 & rs744373>0.5 & rs3865444>0.5"
## [14,] "HippoNV<=0.461434739 & rs744373<=0.5 & rs3851179<=0.5"

##      pred      impRRF
## [1,] "class_1" "1"
## [2,] "class_0"  "0.158820055688818"
## [3,] "class_0"  "0.0882618347383523"
## [4,] "class_1"  "0.073427455249915"
## [5,] "class_0"  "0.0687467199669404"
## [6,] "class_0"  "0.0557289471858664"
## [7,] "class_1"  "0.030470587008693"
## [8,] "class_0"  "0.0255667487231907"
## [9,] "class_0"  "0.019899126787755"
## [10,] "class_1" "0.0179324928636556"
## [11,] "class_1" "0.0142529598174707"
## [12,] "class_1" "0.0133909560650111"
## [13,] "class_1" "0.0106427916981619"
## [14,] "class_1" "0.0101981643861376"

```

Residual analysis

For problems that have a continuous outcome variable, one can apply residual analysis to random forests through the “`plotmo`” R package. Here we perform residual analysis to the AD data where the variable `AGE` is used as the outcome variable. First, we plot the residual vs. fitted figure as shown in Figure 6.8. If the model fits the data well, the data points should spread around the horizontal line (ideally residual = 0). However, in Figure 6.8, there is a linear pattern between the fitted values and residuals. This indicates that the random forest model missed some linear relationship in the AD dataset.

```
require(randomForest)
require(plotmo)
set.seed(1)
path <- "../../data/AD_hd.csv"
data <- read.csv(path, header = TRUE)
target <- data$AGE
rm_indx <- which(colnames(data) %in% c("AGE", "ID", "TOTAL13", "MSCORE"))
X <- data[, -rm_indx]
rf.mod <- randomForest(X, target)
plotres(rf.mod, which = 3)
```

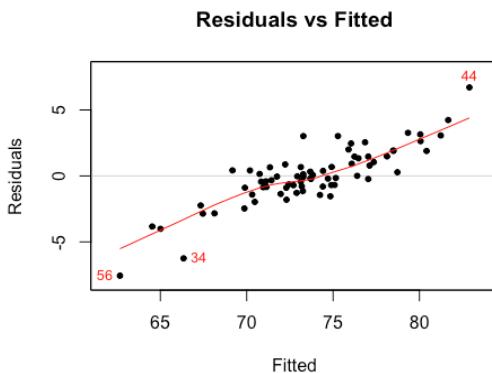


Figure 6.8: Residuals versus fitted in the random forest model

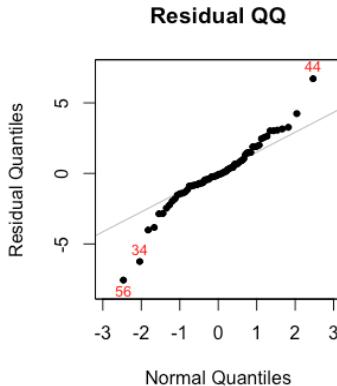


Figure 6.9: The Q-Q plot of residuals of the random forest model

Next, we plot the Q-Q plot that can be interpreted pretty much in the same way as in Figure 6.1. If the random forests fit the data well, the residuals should be pure noise, such that a straight line is expected. However, it can be seen that the residuals deviate from the straight line.

```
plotres(rf.mod, which = 4)
```

Both two residual analysis figures show that the random forest model has underfitting problems, particularly at the two ends of the prediction spectrum. This is expected as the learning boundary of random forests are parallel to the axis. To better illustrate this, we simulate a dataset where the outcome variable has a linear relationship with one single variable. Random forests are applied to the simulated dataset. From the residual vs. fitted Figure as shown in Figure 6.10, we can see that there are three points substantially deviated from the horizontal line and are colored in red. From the Q-Q plot, it is clear that severe derivations happen at the two ends.

```
require(ggplot2)
set.seed(1)
X <- data.frame(X1 = runif(30, min = -1, max = 1))
target <- 0.5 * X$X1 # + 0.5 * X$X2
rf <- randomForest(X, target)
plotres(rf, which = 3)
```

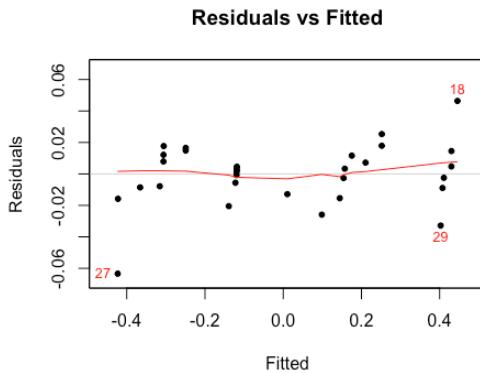


Figure 6.10: Residuals versus fitted in the random forest model fitted on the simulated dataset

```
plotres(rf, which = 4)
```

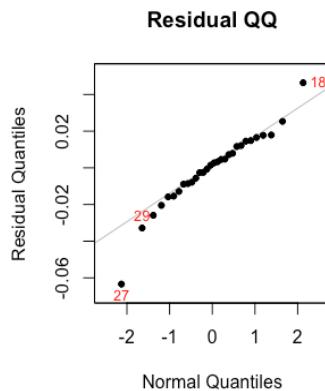


Figure 6.11: The Q-Q plot of residuals of the random forest model fitted on the simulated dataset

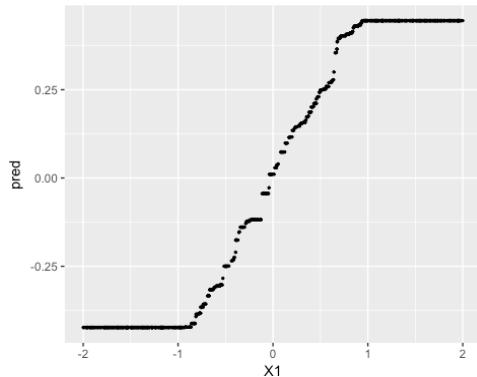


Figure 6.12: Underfitting patterns of random forest to capture linear relationships in dataset

Now we simulate a testing dataset with more data points, and use the trained random forests for prediction. The learning boundary is plotted in Figure 6.12. It can be seen that, at certain intervals, the predicted values remain constant. This is particularly clear at the two ends of the predictor variable, where the predictions have a lower bound and upper bound. This is different from linear regression where the prediction can further increase towards infinitely. This shows that the random forest model can have underfitting problem if there are linear patterns in the dataset.

```
testing <- data.frame(X1 = runif(1000, min = -2, max = 2))
target <- 0.5 * testing$X1 # + 0.5 * X$X2
pred <- predict(rf, testing, type = "response")
pred.data <- cbind(testing, target, pred)
ggplot(pred.data, aes(x = X1, y = pred)) + geom_point(size = 0.5)
```

IV. Clustering

IV.1 Rationale and Formulation

The residual analysis methods mentioned above have implicitly assumed that, the lack of fit of the model to the data is probably resulted from some outliers in the data that are quite different from a majority of the data. In

many applications, the outliers are sparse, randomly distributed, and form no structure. But in some applications, you may find that the implied structure with a majority and a few outliers doesn't apply to the dataset. Rather, it is possible that there are a few majorities that make the dataset heterogeneous.

Thus, while it is not a usual habit in an analytics book to put clustering algorithm together with residual analysis, here, we highlight the utility of clustering method for better understanding of the structure embedded in data to build better prediction models.

Let's start with the Gaussian mixture model (GMM), that has been one of the most popular clustering model. GMM assumes that the data come from not just one distribution but a few. As shown in Figure 6.13, the data is sampled from a mix of 4 distributions.

```
# Simulate a clustering structure
X <- c(rnorm(200, 0, 1), rnorm(200, 10, 2), rnorm(200, 20, 1), rnorm(200, 40, 2))
Y <- c(rnorm(800, 0, 1))
plot(X,Y, ylim = c(-5, 5), pch = 19, col = "gray25")
```

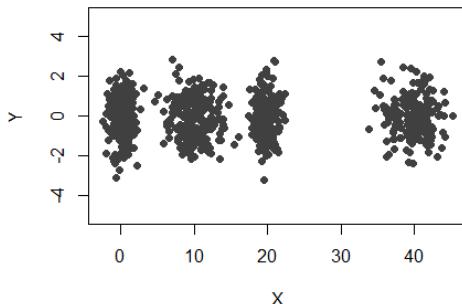


Figure 6.13: A mixture of four Gaussian distributions

This leads to the following formulation of data-generating mechanism. Suppose that there are M distributions mixed together. For each data point x_n , the probability that it comes from the m^{th} distribution is denoted as π_m ,

while $\sum_{m=1}^M \pi_m = 1$. In GMM, we assume that all the distributions are Gaussian distributions, i.e., such that we denote the m^{th} distribution as $N(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$. The task of GMM is to learn the unknown parameters of the distributions $\{\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m, m = 1, 2, \dots, M\}$ and the probability vector $\boldsymbol{\pi}$ that includes the elements $\{\pi_m, m = 1, 2, \dots, M\}$. For simplicity in the presentation, let's use $\boldsymbol{\Theta}$ to denote all these parameters.

IV.2 Theory/Method

To learn these parameters from data, first, we need to derive the likelihood function. First, we realize that, if we have known which distribution the data point \mathbf{x}_n was sampled, it would be straightforward to derive the likelihood function. Following this idea, we invent a binary indicator variable, denoted as z_{nm} , while $z_{nm} = 1$ indicates that \mathbf{x}_n was sampled from the m^{th} distribution. Then, the complete log-likelihood function is:

$$\begin{aligned} l(\boldsymbol{\Theta}) &= \log \prod_{n=1}^N p(\mathbf{x}_n | z_{nm} = 1; \boldsymbol{\Theta}), \\ &= \log \prod_{n=1}^N p(\mathbf{x}_n, z_{nm} | \boldsymbol{\Theta}), \\ &= \log \prod_{n=1}^N \prod_{m=1}^M [p(\mathbf{x}_n | z_{nm} = 1, \boldsymbol{\Theta}) p(z_{nm} = 1)]^{z_{nm}}, \\ &= \sum_{n=1}^N \sum_{m=1}^M [z_{nm} \log p(\mathbf{x}_n | z_{nm} = 1, \boldsymbol{\Theta}) + z_{nm} \log \pi_m]. \end{aligned}$$

Meanwhile, we can derive that

$$p(\mathbf{x}_n | z_{nm} = 1; \boldsymbol{\Theta}) = (2\pi)^{-p/2} |\boldsymbol{\Sigma}_m|^{-1/2} \exp \left\{ -\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_m)^T \boldsymbol{\Sigma}_m^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_m) \right\}.$$

Thus,

$$l(\boldsymbol{\Theta}) = \sum_{n=1}^N \sum_{m=1}^M \left[z_{nm} \log \left((2\pi)^{-p/2} |\boldsymbol{\Sigma}_m|^{-1/2} \exp \left\{ -\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_m)^T \boldsymbol{\Sigma}_m^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_m) \right\} \right) + z_{nm} \log \pi_m \right].$$

To optimize for $\boldsymbol{\Theta}$, we need to overcome the challenge that z_{nm} s are latent and unknown. Here, an intuitive proposal could be:

1. Even we don't know z_{nm} , but we can estimate it if we have known $\boldsymbol{\Theta}$. For instance, it is easy to know that $p(z_{nm} = 1 | \mathbf{X}, \boldsymbol{\Theta}) =$

$\frac{p(x_n|z_{nm}=1, \Theta)\pi_m}{\sum_{k=1}^M p(x_n|z_{nk}=1, \Theta)\pi_k}$. Thus, given Θ , the best estimate of z_{nm} could be the expectation of z_{nm} as $\langle z_{nm} \rangle_{p(z_{nm}|\mathbf{x}, \Theta)} = 1 \cdot p(z_{nm} = 1|\mathbf{X}, \Theta) + 0 \cdot p(z_{nm} = 0|\mathbf{X}, \Theta) = \frac{p(x_n|z_{nm}=1, \Theta)\pi_m}{\sum_{k=1}^M p(x_n|z_{nk}=1, \Theta)\pi_k}$.

2. We can fill in $l(\Theta)$ with the estimated z_{nm} and optimize it to update Θ . Feed this updated back to Step 1 and repeat the iterations, until all the parameters in the iterations don't change significantly.

To do step 2, we need to derive the estimated $l(\Theta)$, which is denoted as $\langle l(\Theta) \rangle_{p(\mathbf{z}|\mathbf{x}, \Theta)}$. It can be seen that

$$\langle l(\Theta) \rangle_{p(\mathbf{z}|\mathbf{x}, \Theta)} = \sum_{n=1}^N \sum_{m=1}^M [\langle z_{nm} \rangle_{p(z_{nm}|\mathbf{x}, \Theta)} \log p(x_n|z_{nm} = 1, \Theta) + \langle z_{nm} \rangle_{p(z_{nm}|\mathbf{x}, \Theta)} \log \pi_m].$$

To optimize for the parameters $\{\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m, m = 1, 2, \dots, M\}$, we take derivatives of $\langle l(\Theta) \rangle_{p(\mathbf{z}|\mathbf{x}, \Theta)}$ regarding to these parameters and put them equal to zero:

$$\frac{\partial \langle l(\Theta) \rangle_{p(\mathbf{z}|\mathbf{x}, \Theta)}}{\partial \boldsymbol{\mu}_m} = \sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm}|\mathbf{x}, \Theta)} \frac{\partial \log p(x_n|z_{nm}=1, \Theta)}{\partial \boldsymbol{\mu}_m} = \mathbf{0}.$$

We can derive that

$$\begin{aligned} & \frac{\partial \log p(x_n|z_{nm}=1, \Theta)}{\partial \boldsymbol{\mu}_m} = \\ & \frac{\partial \log((2\pi)^{-p/2} |\boldsymbol{\Sigma}_m|^{-1/2} \exp\{-\frac{1}{2}(x_n - \boldsymbol{\mu}_m)^T \boldsymbol{\Sigma}_m^{-1} (x_n - \boldsymbol{\mu}_m)\})}{\partial \boldsymbol{\mu}_m} = \\ & -\frac{1}{2} \frac{\partial (x_n - \boldsymbol{\mu}_m)^T \boldsymbol{\Sigma}_m^{-1} (x_n - \boldsymbol{\mu}_m)}{\partial \boldsymbol{\mu}_m} = (x_n - \boldsymbol{\mu}_m)^T \boldsymbol{\Sigma}_m^{-1}. \end{aligned}$$

Thus, putting these together we can have

$$\sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm}|\mathbf{x}, \Theta)} (x_n - \boldsymbol{\mu}_m)^T \boldsymbol{\Sigma}_m^{-1} = \mathbf{0}.$$

This gives us the equation to estimate $\boldsymbol{\mu}_m$ as

$$\boldsymbol{\mu}_m = \frac{\sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm}|\mathbf{x}, \Theta)} x_n}{\sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm}|\mathbf{x}, \Theta)}}.$$

Similarly, we take derivatives of $\langle l(\Theta) \rangle_{p(\mathbf{z}|\mathbf{x}, \Theta)}$ regarding $\boldsymbol{\Sigma}_m$ and put them equal to zero:

$$\frac{\partial \langle l(\Theta) \rangle_{p(\mathbf{z}|\mathbf{x}, \Theta)}}{\partial \boldsymbol{\Sigma}_m} = \sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm}|\mathbf{x}, \Theta)} \frac{\partial \log p(x_n|z_{nm}=1, \Theta)}{\partial \boldsymbol{\Sigma}_m} = \mathbf{0}.$$

We can derive that

$$\begin{aligned} \frac{\partial \log p(x_n | z_{nm}=1, \Theta)}{\partial \Sigma_m} &= \\ \frac{\partial \log((2\pi)^{-p/2} |\Sigma_m|^{-1/2} \exp\left\{-\frac{1}{2}(x_n - \mu_m)^T \Sigma_m^{-1} (x_n - \mu_m)\right\})}{\partial \Sigma_m} &= \\ \frac{1}{2} \frac{\partial\{|\Sigma_m|^{-1/2} - (x_n - \mu_m)^T \Sigma_m^{-1} (x_n - \mu_m)\}}{\partial \Sigma_m} &= \frac{1}{2} [\Sigma_m - (x_n - \mu_m)(x_n - \mu_m)^T]. \end{aligned}$$

Thus, putting these together we can have

$$\sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm} | \mathbf{x}, \Theta)} [\Sigma_m - (x_n - \mu_m)(x_n - \mu_m)^T] = \mathbf{0}.$$

This gives us the equation to estimate Σ_m as

$$\Sigma_m = \frac{\sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm} | \mathbf{x}, \Theta)} (x_n - \mu_m)(x_n - \mu_m)^T}{\sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm} | \mathbf{x}, \Theta)}}.$$

Lastly, in order to optimize for $\{\pi_m, m = 1, 2, \dots, M\}$, we face with the problem that $\sum_{m=1}^M \pi_m = 1$. To address this, we introduce the Lagrange multiplier λ and optimize for

$$\langle l(\Theta) \rangle_{p(z | \mathbf{x}, \Theta)} - \lambda (\sum_{m=1}^M \pi_m - 1).$$

We take derivatives of it regarding π_m and put them equal to zero:

$$\frac{\partial [\langle l(\Theta) \rangle_{p(z | \mathbf{x}, \Theta)} - \lambda (\sum_{m=1}^M \pi_m - 1)]}{\partial \pi_m} = \frac{\partial \langle l(\Theta) \rangle_{p(z | \mathbf{x}, \Theta)}}{\partial \pi_m} - \lambda = 0.$$

It is known that

$$\frac{\partial \langle l(\Theta) \rangle_{p(z | \mathbf{x}, \Theta)}}{\partial \pi_m} = \frac{1}{\pi_m} \sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm} | \mathbf{x}, \Theta)}.$$

Thus, for $m = 1, 2, \dots, M$ we arrive at

$$\sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm} | \mathbf{x}, \Theta)} - \lambda \pi_m = 0.$$

Adding these M equations together, we have

$$\sum_{m=1}^M \sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm} | \mathbf{x}, \Theta)} - \lambda \sum_{m=1}^M \pi_m = 0.$$

Since $\sum_{m=1}^M \pi_m = 1$, we can get that

$$\lambda = \sum_{m=1}^M \sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm} | \mathbf{x}, \Theta)} = N.$$

Thus,

$$\pi_m = \frac{\sum_{n=1}^N \langle z_{nm} \rangle_{p(z_{nm} | \mathbf{x}, \Theta)}}{N}.$$

IV.3 R Lab

The R package “**Mclust**” could be used to implement the GMM model while the underlying algorithm is the EM algorithm. Again, using the simulated data with four clusters, the following R code is to identify clusters.

```
# use GMM to identify the clusters
require(mclust)

XY.clust <- Mclust(data.frame(X,Y))
summary(XY.clust)

plot(XY.clust)
```

Then, we can obtain:

```
## -----
## Gaussian finite mixture model fitted by EM algorithm
## -----
##
## Mclust VVI (diagonal, varying volume and shape) model with 4 components:
##
##   log.likelihood    n  df        BIC        ICL
##             -3666.07 800 19 -7459.147 -7459.539
##
## Clustering table:
##   1   2   3   4
## 199 201 200 200
```

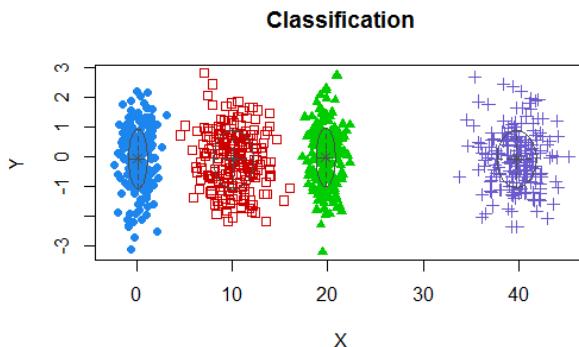


Figure 6.14: Clustering results of the simulated data

Note that, here, we didn't specify how many clusters should **Mclust** find. It seems that, by using model selection criteria such as **BIC** which balances model fit and model complexity (here refers to the number of clusters), mclust correctly identified the four clusters. It can also been seen that, for each cluster, the data points are almost 200.

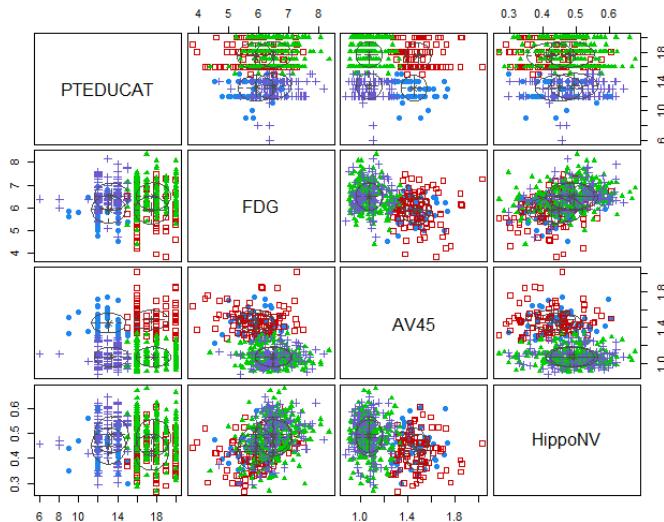


Figure 6.15: Clustering results of the AD data

Now let's implement GMM on the AD data using **Mclust**. Result is shown in Figure 6.15.

```
# install.packages("mclust")
require(mclust)
AD.Mclust <- Mclust(AD[,c(3,4,5,6,10,12,14,15)])
summary(AD.Mclust)

AD.Mclust$data = AD.Mclust$data[,c(1:4)]
# plot(AD.Mclust)

## -----
## Gaussian finite mixture model fitted by EM algorithm
## -----
##
```

```

## Mclust EEI (diagonal, equal volume and shape) model with 4 components:
##
##   log.likelihood    n df          BIC          ICL
##             -3235.874 517 43 -6740.414 -6899.077
##
## Clustering table:
##   1   2   3   4
##  43 253  92 129

```

Interestingly, it seems that in the AD data, four clusters are identified as well. And results are shown in Figure 6.15. It seems a reasonable clustering result, although the boundaries between clusters are not as distinct as the boundaries in Figure 6.14. In real applications, particularly for those applications for which we haven't known enough, clustering is more like an exploration tool. It could generate suggestive results, but probably not confirmative conclusions.

IV.4 Remarks

Clustering-based prediction models: As the existence of clustering structure in a dataset violates the assumption of many prediction models such as the regression model that assume the data come from a homogeneous distribution, evidences in the literature and practices have shown that it will increase performance of prediction if we could identify the clusters and build prediction models separately for the clusters. In the literature, some algorithms have been developed to integrate clustering and prediction models jointly. For example, the Treed Regression method¹ is one of the earlier examples that propose to build a tree to stratify the dataset and create regression models on the leaves. Similarly, the logistic model trees model² also builds the tree to allocate data points into different leaves and build different logistic regression model for each leaf. Motivated

¹ Alexander, W. and Grimshaw, S. Treed regression. *Journal of computational and graphical statistics*, 1996.

² Landwehr, N., Hall, M. and Frank, E. Logistic model trees, *Machine learning*, 2004.

by this line of thoughts, more models have been developed with different combination of tree models and prediction models (or other types of statistical models) on the leaves¹².

The EM algorithm: The iterative two-step algorithm is actually the idea of the Expectation-Maximization (EM) algorithm. The EM algorithm is commonly used to solve for this type of problems that involve latent variables. The idea of the EM algorithm in GMM is to follow the iterative two-steps as shown in below:

1. The E-step: Derive the posterior distribution of \mathbf{Z} as $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\Theta})$. Calculate the expectation of $l(\boldsymbol{\Theta})$ according to this distribution, i.e., denoted as $\langle l(\boldsymbol{\Theta}) \rangle_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\Theta})}$.
2. The M-step: obtain $\boldsymbol{\Theta}$ by maximizing $\langle l(\boldsymbol{\Theta}) \rangle_{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\Theta})}$.

The power of the EM algorithm draws on the Jensen's inequality. The Jensen's inequality says that, let f be a convex function defined on an interval I . If $x_1, x_2, \dots, x_n \in I$ and $\gamma_1, \gamma_2, \dots, \gamma_n \geq 0$ with $\sum_{i=1}^n \gamma_i = 1$, then $f(\sum_{i=1}^n \gamma_i x_i) \leq \sum_{i=1}^n \gamma_i f(x_i)$.

Below we show how it work:

$$\begin{aligned} \log p(\mathbf{X}; \boldsymbol{\Theta}) &= \log \int p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\Theta}) d\mathbf{Z}, \\ &= \log \int Q(\mathbf{Z}) \frac{p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\Theta})}{Q(\mathbf{Z})} d\mathbf{Z}, \\ &\geq \int Q(\mathbf{Z}) \log \frac{p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\Theta})}{Q(\mathbf{Z})} d\mathbf{Z}, \\ &= \int Q(\mathbf{Z}) \log p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\Theta}) d\mathbf{Z} - \int Q(\mathbf{Z}) Q(\mathbf{Z}) d\mathbf{Z}. \end{aligned}$$

The EM algorithm proposed to use $Q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\Theta})$. At each M-step, it can be see that, while the goal is to maximize $\log p(\mathbf{X}; \boldsymbol{\Theta})$, we could maximize its lower bound $\int Q(\mathbf{Z}) \log p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\Theta}) d\mathbf{Z}$, such that the objective

¹ Gramacy, R. and Lee, H. Bayesian treed gaussian process models with an application to computer modeling, *Journal of American statistical association*, 2008.

² Liu, H., Chen, X., Lafferty, J. and Wasserman, L. Graph-valued regression, NIPS 2009.

function $\log p(\mathbf{X}; \boldsymbol{\Theta})$ is improved with guarantee that the new objective function won't decrease along the iterations.

IV. Exercises

Data analysis

1. Find five regression datasets from the UCI data repository or R dataset. Conduct a detailed analysis using the linear regression model. Conduct model selection and validation. Conduct residual analysis of your final models, and comment on your results.
2. Find five classification datasets from the UCI data repository or R datasets. Conduct a detailed analysis using the logistic regression model. Conduct model selection and validation. Conduct residual analysis of your final models, and comment on your results.
3. For the five classification datasets you have selected from the UCI data repository or R datasets, conduct a detailed analysis using the random forest model. Conduct model selection and validation. Conduct residual analysis of your final models, and comment on your results.

Derivation

4. Derive a decision tree model that builds logistic regression model in its leaf nodes. You can get some help by reading this article¹. Name this hybrid decision tree model!

Programming

5. Write your own R script to implement the hybrid decision tree model you have developed.

¹ <https://cran.r-project.org/web/packages/rpart/vignettes/usercode.pdf>

6. Simulate a dataset that fits this model. Then, build your hybrid decision tree model, logistic regression model, and random forest model on this simulated dataset. Compare their performances.
7. Repeat 6 on some other datasets you select from the UCI data repository or R dataset.

CHAPTER 7: BALANCE

SVM AND ENSEMBLE LEARNING

I. Overview

Chapter 6 is about “**balance**”. As in Chapter 4 we have introduced the concept of overfitting, here, we further expand the issue of overfitting and introduce two famous models that provide two different approaches to address the overfitting issue. The two methods are the Support Vector Machine (SVM) and Ensemble Learning that includes RF as a particular case. The solutions provided in both methods to control the risk of overfitting are architectural, rather than some technical adjustments or implementation tricks.

II. Support Vector Machine

II.1 Rationale and Formulation

As we have learned that, the complexity of the final model should match the complexity of the signal embed in the noise. Overfitting happens when the model not only fits the signal part of the data, but also the noise part.

Overfitting could lead to promising results on the training data since the model actually “memorizes” the training data rather than generalizing the training data in the form as a model. Since the noise in the training data won’t reappear in future unseen data, it is sure that the overfitted model won’t perform well on future unseen data.

Thus, a question that appears in every practice of data analytics is, what model should I use? Is the model too simple? Or too complex?

Let’s give this question a nice and specific context. Consider the classification problem that uses linear model to represent the decision boundary, as shown in below.

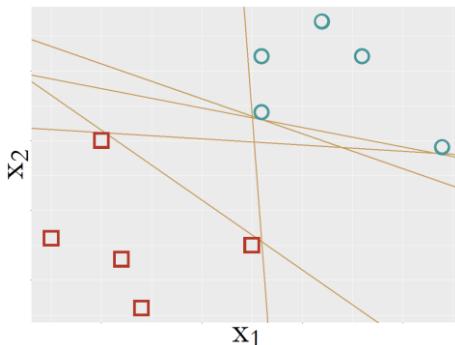


Figure 7.1: Which model (e.g., which line) should we use as our classification model to separate the two classes of data points?

From Figure 7.1, we can see that, if we only consider the classification error on the given data points of the two classes, it seems that all the models (represented as the lines) could achieve perfect classification. Thus, classification error is not sufficient in this case for us to decide on the optimal model. What else should we bring into the thought process?

As we have mentioned that, the objective of the model is to predict on future unseen data, now we may turn our attention from the given training data shown in Figure 7.1 to future unseen data. What could the future unseen data look like? Will all the models shown in Figure 7.1 perform equally well on the future unseen data?

It seems that, the lines that are close to the data points may bear a risk of performing bad on future unseen data. This is because that it seems to be very plausible that future red data points may allocate a little bit outside of the current region of the red data points, and thus, if we pick up the line that is close to red data points, the model may just misclassify the new red data points.

In other words, the lines that are too close to either side of the data points lack a safe **margin**. To reduce risk, we like to have the margin as large as possible. This is in the same spirit of risk management. This gave birth to the idea of SVM, as shown in Figure 7.2, where the model SVM suggests is the one that has the **maximum margin**.

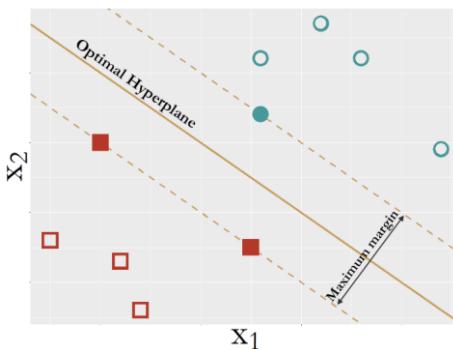


Figure 7.2: The model that has the maximum margin – the basic idea of SVM

II.2 Theory/Method

Derivation of the SVM formulation: Denote the training data points as $\{(\mathbf{x}_n, y_n), n = 1, 2, \dots, N\}$. We are now ready to derive the mathematical framework corresponding to the idea shown in Figure 7.2. The goal is to identify a model, $\mathbf{w}^T \mathbf{x} + b$, using which we can make binary classification:

If $\mathbf{w}^T \mathbf{x} + b > 0$, then $y = 1$;

Otherwise, $y = -1$;

As we aim to maximize the margin, first, we need to be able to denote the margin mathematically in terms of the model parameters \mathbf{w} and b .

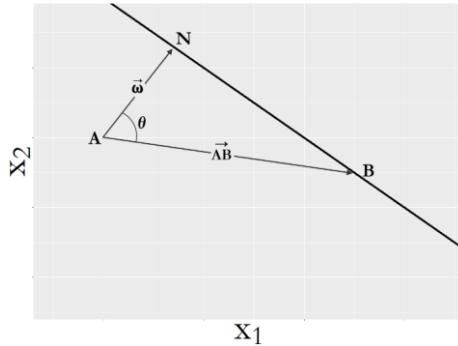


Figure 7.3: Illustration of how to derive the margin

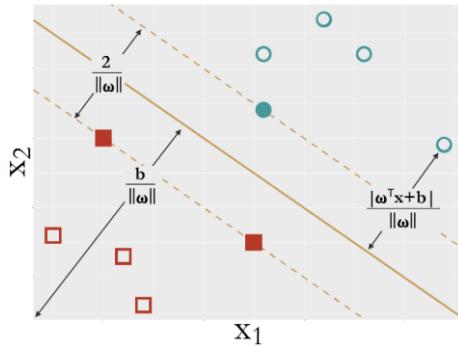


Figure 7.4: Formulation of the idea of maximum margin

Note that, as shown in Figure 7.3, for a data point A , its perpendicular distance to the line $\mathbf{w}^T \mathbf{x} + b = 0$ can be derived as:

$$\|AN\| = \|AB\| \cos \theta = \|AB\| \frac{\overrightarrow{AB} \cdot \vec{\omega}}{\|AB\| \|\vec{\omega}\|} = \frac{\overrightarrow{AB} \cdot \vec{\omega}}{\|\vec{\omega}\|}.$$

Denote the coordinates of the two data points A and B as \mathbf{x}_a and \mathbf{x}_b , respectively. Then, we can see that

$$\frac{\overrightarrow{AB} \cdot \vec{\omega}}{\|\vec{\omega}\|} = \frac{\mathbf{w}^T (\mathbf{x}_a - \mathbf{x}_b)}{\|\mathbf{w}\|}.$$

As the data point B is an arbitrary data point on the line $\mathbf{w}^T \mathbf{x} + b = 0$, it means that $\mathbf{w}^T \mathbf{x}_b = -b$. Thus, we can further derive that

$$\|AN\| = \frac{\mathbf{w}^T(\mathbf{x}_a - \mathbf{x}_b)}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T \mathbf{x}_a + b}{\|\mathbf{w}\|}.$$

Now we can lay this derivation on Figure 7.2 to obtain Figure 7.4.

As shown in Figure 7.4, with a clear characterization of the margin of the data points to the decision line in terms of the model parameters \mathbf{w} and b , we still need more to write up the objective function of SVM that can maximizes the margin. We know that, as shown in Figure 7.2, the margin is only determined by \mathbf{w} , but it seems that in our derivation the margin also depends on the data points. Actually, this indicates that there is a numerical dimension to fix, as currently the whole formulation is underdetermined.

Thus, in the formulation of SVM, it was suggested to fix numerical scale of the model with a constraint:

$$|\mathbf{w}^T \mathbf{x}_n + b| = 1 \text{ for any } \mathbf{x}_n \text{ that is on the margin.}$$

With this fix, now we are ready to derive the margin in the SVM model as $\frac{2}{\|\mathbf{w}\|}$. To maximize the margin of the model is equivalent to minimize $\|\mathbf{w}\|$. This gives us the objective function of the SVM model.

Now let's derive the constraints of the SVM model. To derive the model from these training data points, obviously, we need to make sure the model can perform correctly on the training data. As the data points on the margin satisfy $|\mathbf{w}^T \mathbf{x}_n + b| = 1$, the data points that are beyond the margin will satisfy $|\mathbf{w}^T \mathbf{x}_n + b| > 1$.

Thus, the final SVM formulation is:

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|,$$

$$\text{Subject to: } y_n (\mathbf{w}^T \mathbf{x}_n + b) \geq 1 \text{ for } n = 1, 2, \dots, N.$$

To solve this problem, first, we can use the method of lagrange multiplier:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N \alpha_n [y_n (\mathbf{w}^T \mathbf{x}_n + b) - 1].$$

This could be rewritten as

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{n=1}^N \alpha_n y_n \mathbf{w}^T \mathbf{x}_n - b \sum_{n=1}^N \alpha_n y_n + \sum_{n=1}^N \alpha_n.$$

Differentiating $L(\mathbf{w}, b, \boldsymbol{\alpha})$ with respect to \mathbf{w} and b , and setting to zero yields:

$$\begin{aligned}\mathbf{w} &= \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n, \\ \sum_{n=1}^N \alpha_n y_n &= 0.\end{aligned}$$

Then, we can rewrite $L(\mathbf{w}, b, \boldsymbol{\alpha})$ as

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m.$$

This is because that:

$$\begin{aligned}\frac{1}{2} \mathbf{w}^T \mathbf{w} &= \frac{1}{2} \mathbf{w}^T \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n = \frac{1}{2} \sum_{n=1}^N \alpha_n y_n \mathbf{w}^T \mathbf{x}_n = \\ \frac{1}{2} \sum_{n=1}^N \alpha_n y_n (\sum_{n=1}^N \alpha_n y_n \mathbf{x}_n)^T \mathbf{x}_n &= \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m.\end{aligned}$$

Then, finally, we can derive the model of SVM by solving its dual form problem:

$$\max_{\boldsymbol{\alpha}} \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m,$$

Subject to: $\alpha_n \geq 0$ for $n = 1, 2, \dots, N$ and $\sum_{n=1}^N \alpha_n y_n = 0$.

This is a **quadratic programming** problem that can be solved using many existing packages.

Note that, the learned model parameters could be represented as:

$$\hat{\mathbf{w}} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n \text{ and } \hat{b} = 1 - \hat{\mathbf{w}}^T \mathbf{x}_n \text{ for any } \mathbf{x}_n \text{ whose } \alpha_n > 0.$$

And we know that, based on the KKT condition of the SVM formulation, the following equations must hold:

$$\alpha_n [y_n (\mathbf{w}^T \mathbf{x}_n + b) - 1] = 0 \text{ for } n = 1, 2, \dots, N.$$

Thus, for any data point, e.g., the nth data point, it is either

$$\alpha_n = 0 \text{ or } y_n (\mathbf{w}^T \mathbf{x}_n + b) - 1 = 0.$$

Support vectors: Actually, this leads to the following interesting phenomenon, which leads to the definition of the “**support vectors**” as shown in Figure 7.5. The support vectors are what have been taken by the learning algorithm to constitute its decision function, which thus hold crucial implications for the SVM model. First, based on some theoretical

evidences, the number of support vectors is usually a metric that can indicate the healthiness of the model, i.e., the smaller the better. Second, it also reveals that the main statistical information the SVM model uses is from the support vectors. Thus, some works have been inspired by this aspect to accelerate the computation of SVM model training by discarding potentially non-support-vectors.

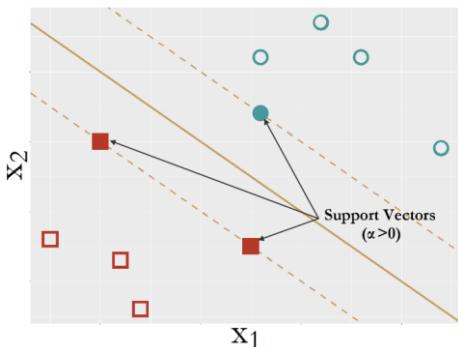


Figure 7.5: Support vectors of SVM are the data points that are on the margins

Extension to nonseparable cases: Note that, we have assumed that the two classes are separable. It is easy to relax this assumption. Ideally, in SVM, we hope that all the data points are either on or beyond the margin. We could relax this idealism and allow some data points to be within the margins or even on the wrong side of the decision line. To do so, we introduce the slack variables:

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n \text{ for } n = 1, 2, \dots, N.$$

As shown in Figure 7.6, the data points that are within the margins will have the corresponding slack variables as $0 \leq \xi_n \leq 1$, and the data points that are on the wrong side of the decision line have the corresponding slack variables as $\xi_n > 1$.

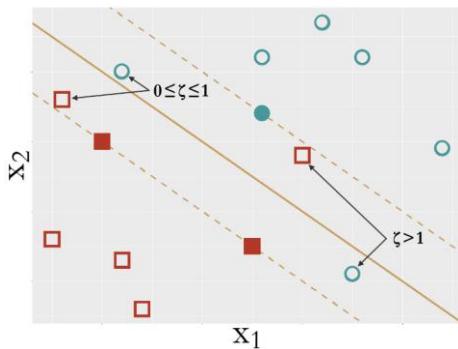


Figure 7.6: Behaviors of the slack variables

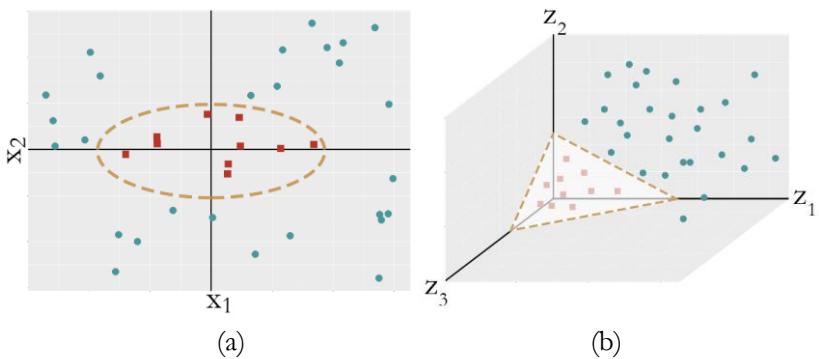


Figure 7.7: (a) A linearly inseparable dataset; (b) with transformation (a) becomes separable

The corresponding formulation of the SVM model becomes:

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n,$$

Subject to: $y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n$ and $\xi_n \geq 0$, for $n = 1, 2, \dots, N$.

Here, C is a user-specified parameter to control how much tolerance we can assign for the slack variables.

Extension to nonlinear SVM: So far we have presented SVM in linear models. Sometimes, the decision boundary could not be characterized as linear models, as shown in Figure 7.7 (a).

To create a nonlinear model within the framework of linear model, we could conduct transformation of the original variables. Here, we conduct the following transformation from \mathbf{x} to \mathbf{z} :

$$\begin{aligned} z_1 &= x_1^2, \\ z_2 &= \sqrt{2}x_1x_2, \\ z_3 &= x_2^2. \end{aligned}$$

Then, in the new coordinates system, as shown in Figure 7.7 (b), the data points of the two classes become separable. This is the approach we often use in regression models as well, to create explicit transformation that asks us to write up how the features \mathbf{z} could be represented as \mathbf{x} .

A remarkable thing about SVM is that, its formulation allows implicit transformation. This implicit transformation could be done by the use of kernel function. The dual formulation of SVM on the transformed variables is:

$$\max_{\alpha} \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m \mathbf{z}_n^T \mathbf{z}_m,$$

Subject to: $0 \leq \alpha_n \leq C$ for $n = 1, 2, \dots, N$ and $\sum_{n=1}^N \alpha_n y_n = 0$.

It can be seen that, the dual formulation of SVM shown above doesn't really need the information of individual \mathbf{z}_n . Rather, only the inner product of $\mathbf{z}_n^T \mathbf{z}_m$ is needed. As \mathbf{z} is essentially functional of \mathbf{x} , i.e., $\mathbf{z} = \phi(\mathbf{x})$, it can be seen that $\mathbf{z}_n^T \mathbf{z}_m$ is essentially function of \mathbf{x}_n and \mathbf{x}_m . Thus, we can write it up as $\mathbf{z}_n^T \mathbf{z}_m = K(\mathbf{x}_n, \mathbf{x}_m)$. This is called the "**kernel function**". A kernel function is a function that theoretically entails a transformation $\mathbf{z} = \phi(\mathbf{x})$ such that $K(\mathbf{x}_n, \mathbf{x}_m)$ implies that it can be written as an inner product $K(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x})^T \phi(\mathbf{x})$. In other words, our effort now is not to seek explicit transformations that may be tedious and difficult, rather, we seek kernel functions that entail such transformations.

Nowadays we have had many such kernel functions to use. For example, the Gaussian radial basis kernel function is defined as

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2},$$

where the transformation $\mathbf{z} = \phi(\mathbf{x})$ is implicit and infinitely long to represent any smooth function.

The polynomial kernel function is defined as

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^q.$$

Also, the linear kernel function is defined as

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j.$$

And there are still many new kernel functions to be developed to enrich our capacity of representing nonlinear decision boundaries in real-world applications. With a given kernel function, SVM learns the model by solving the following optimization problem:

$$\max_{\alpha} \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m K(\mathbf{x}_n, \mathbf{x}_m),$$

Subject to: $0 \leq \alpha_n \leq C$ for $n = 1, 2, \dots, N$ and $\sum_{n=1}^N \alpha_n y_n = 0$.

However, in the kernel space, it will no longer be possible to write up the parameter \mathbf{w} the same way as in linear models.

For any new data point, denoted as \mathbf{x}_* , the learned SVM model predict on it as

$$\begin{aligned} \text{If } \sum_{n=1}^N \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}_*) + b > 0, \text{ then } y = 1; \\ \text{Otherwise, } y = -1. \end{aligned}$$

II.3 R Lab

Let's try on an example. Consider a dataset:

$$\begin{aligned} \mathbf{x}_1 &= (-1, -1), y_1 = -1; \\ \mathbf{x}_2 &= (-1, +1), y_2 = +1; \\ \mathbf{x}_3 &= (+1, -1), y_3 = +1; \\ \mathbf{x}_4 &= (+1, +1), y_4 = -1. \end{aligned}$$

We could use R to visualize this dataset. The R code is shown in below:

```
# Package installation
# pkgs <- c( 'ggplot2', 'kernLab', 'ROCR' )
# install.packages( pkgs )
# source( 'http://bioconductor.org/biocLite.R' )
# biocLite( 'ALL' )
```

```

# For the toy problem
x = matrix(c(-1,-1,1,1,-1,1,-1,1), nrow = 4, ncol = 2)
y = c(-1,1,1,-1)
linear.train <- data.frame(x,y)

# Visualize the distribution of data points of two classes
require( 'ggplot2' )

p <- qplot( data=linear.train, X1, X2, colour=factor(y), xlim = c
(-1.5,1.5), ylim = c(-1.5,1.5))
p <- p + labs(title = "Scatterplot of data points of two classes")
print(p)

```

The dataset is visualized in Figure 7.8. It is clear that the dataset presents a linearly inseparable problem, calling for the use of kernel to build nonlinear classification boundary.



Figure 7.8: A linearly inseparable dataset

Now, consider the kernel function, $K(\mathbf{x}_n, \mathbf{x}_m) = (\mathbf{x}_n^T \mathbf{x}_m + 1)^2$, which corresponds to the transformation:

$$\phi(\mathbf{x}_n) = [1, \sqrt{2}x_{n,1}, \sqrt{2}x_{n,2}, \sqrt{2}x_{n,1}x_{n,2}, x_{n,1}^2, x_{n,2}^2]^T.$$

The objective function becomes:

$$\max_{\alpha} \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m K(\mathbf{x}_n, \mathbf{x}_m),$$

Subject to: $\alpha_n \geq 0$ for $n = 1, 2, \dots, N$ and $\sum_{n=1}^N \alpha_n y_n = 0$.

We can calculate the kernel matrix as

$$\mathbf{K} = \begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}.$$

Then, we can solve the quadratic programming problem and get that

$$\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 0.125.$$

In this particular case, as we can write up the transformation explicitly, we can write up $\hat{\mathbf{w}}$ explicitly as:

$$\hat{\mathbf{w}} = \sum_{n=1}^4 \alpha_n y_n \phi(\mathbf{x}_n) = [0, 0, 0, 1/\sqrt{2}, 0, 0]^T.$$

Then, we can write up the decision function explicitly as:

$$f(\mathbf{x}_*) = \hat{\mathbf{w}}^T \phi(\mathbf{x}_*) = x_{*,1} x_{*,2}.$$

As you can see, this is the decision boundary for a typical XOR problem. On the other hand, we can use R to build the SVM model on this dataset and see if our results could be reproduced in R. To do so, we use the R package “**kernlab**” and its function `ksvm`. The R code is shown in below:

```
# Train a Linear SVM
x <- cbind(1, poly(x, degree = 2, raw = TRUE))
coefs = c(1,sqrt(2),sqrt(2),sqrt(2),1,1)
x <- x * t(matrix(rep(coefs,4),nrow=6,ncol=4))
linear.train <- data.frame(x,y)
require('kernlab')

linear.svm <- ksvm(y ~ ., data=linear.train, type='C-svc', kernel
='vanilladot', C=10, scale=c())
```

The function `alpha()` returns the values of α_n for $n = 1, 2, \dots, N$. Here, note that, the function actually scaled the vector α . Thus, our manual results are consistent with the results obtained by using R.

```
alpha(linear.svm) #scaled alpha vector
```

```
## [[1]]
## [1] 0.2499619 0.2499619 0.2499873 0.2499873
```

Now let's do more examples. First, let's generate a dataset with linearly separable boundary.

```
# Generate a dataset with linear boundary
n <- 200
p <- 2
```

```

n.pos <- n/2
x.pos <- matrix(rnorm( n*p, mean=0, sd=1 ), n.pos, p)
x.neg <- matrix(rnorm( n*p, mean=2, sd=1 ), n-n.pos, p)
y <- c(rep(1, n.pos), rep(-1, n-n.pos))
n.train <- floor( 0.8 * n )
idx.train <- sample( n, n.train )
is.train <- rep( 0, n )
is.train[idx.train] <- 1
linear.data <- data.frame( x=rbind( x.pos, x.neg ), y=y, train=is.train )
# Extract train and test subsets of the dataset
linear.train <- linear.data[linear.data$train==1, ]
linear.train <- subset( linear.train, select=-train )
linear.test <- linear.data[linear.data$train==0, ]
linear.test <- subset( linear.test, select=-train )
str(linear.train)

## 'data.frame':   160 obs. of  3 variables:
## $ x.1: num  0.707 -0.92 0.87 -0.621 2.101 ...
## $ x.2: num  -2.959 1.37 -0.526 0.989 0.833 ...
## $ y : num  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...

str(linear.test)

## 'data.frame':   40 obs. of  3 variables:
## $ x.1: num  -1.857 -1.703 0.923 1.448 -0.723 ...
## $ x.2: num  1.7934 0.0927 1.4485 1.1649 -0.1459 ...
## $ y : num  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...

```

We can visualize the data as shown in Figure 7.9.

```

# Visualize the distribution of data points of two classes
require( 'ggplot2' )
p <- qplot( data=linear.data, x.1, x.2, colour=factor(y) )
p <- p + labs(title = "Scatterplot of data points of two classes")
print(p)

```

We then use the **ksvm()** function to build the SVM model:

```

# Train a Linear SVM
require( 'kernlab' )
linear.svm <- ksvm(y ~ ., data=linear.train, type='C-svc', kernel
='vanilladot', C=10, scale=c())

```

By typing in **linear.svm**, we can see more details of the built model. In this analysis, out of 200 data points, only 7 data points are needed to be the support vectors to define the linear boundary to separate two classes. This

is a good and healthy sign of the generalizability of the model to achieve robust success on unseen future data if the unseen future data would come from the same distribution of the training data.

We can also visualize the built model using the following R code, as shown in Figure 7.10. The black points shown in Figure 7.10 are the support vectors.

```
# Plot the model
plot( linear.svm, data=linear.train )
```

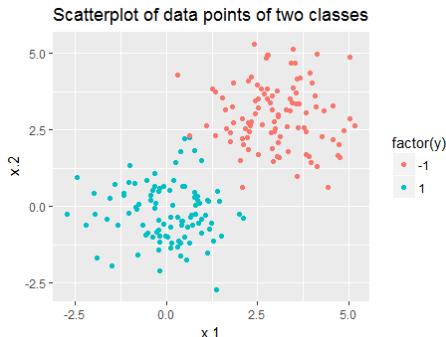


Figure 7.9: A randomly generated dataset with linearly separable boundary

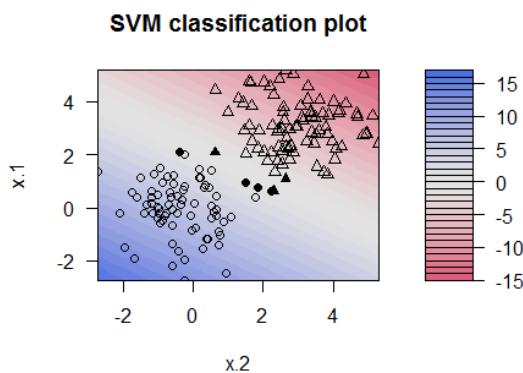


Figure 7.10: Visualization of the linear SVM model built for the simulated dataset

To verify our hypothesis that the model would obtain robust performance on unseen testing data, here, we present the ROC curve of the linear SVM model on the testing data that is not used in training the model.

```
# Generate the ROC curve using the testing data
# Prediction scores
linear.prediction.score <- predict(linear.svm, linear.test, type=
'decision')
# Compute ROC and Precision-Recall curves
require( 'ROCR' )

linear.roc.curve <- performance( prediction( linear.prediction.sc
ore, linear.test$y ), 
                                 measure='tpr', x.measure='fpr' )
plot(linear.roc.curve, lwd = 2, col = "orange3",
      main = "Validation of the linear SVM model using testing dat
a")
```

As shown in Figure 7.11, indeed, the ROC curve shows the linear SVM model predicts well on the testing data.

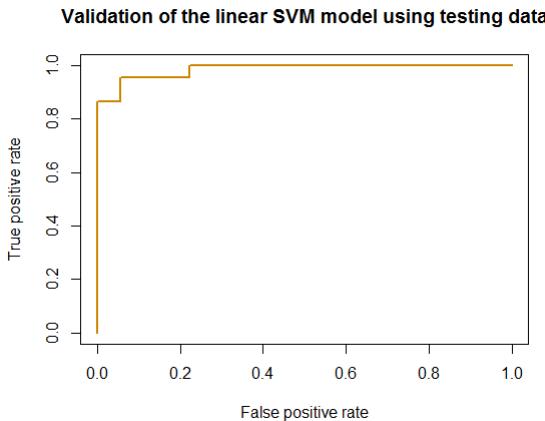


Figure 7.11: The ROC curve of the linear SVM model on testing data

On the other hand, similarly as what we have discussed in Chapter 5, in practice we will not use a testing data to guide the model selection. Thus,

the performance of the trained model has to be evaluated using the training data.

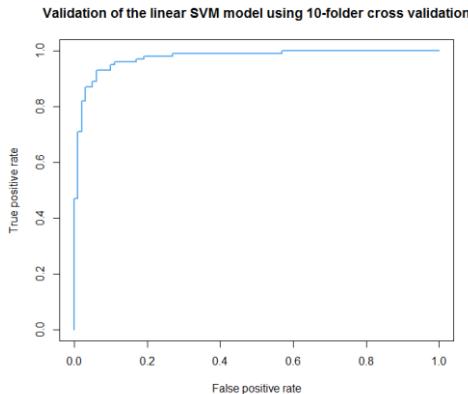


Figure 7.12: The ROC curve of the linear SVM model by 10-fold cross-validation

To achieve this, the cross-validation has been shown in Chapter 5 that can approximate the testing performance of the model. Here, again, we use this example to show that indeed the cross-validation provides such an effective way. The R code is shown in below:

```
# Generate the ROC curve using 10-fold cross validation
n <- nrow(linear.data)
n.folds=10
idx <- split(sample(seq(n)), seq(n.folds))
scores <- rep(0, n)
for(i in seq(n.folds)) {
  model <- ksvm(y ~ ., data=linear.data[-idx[[i]], ], kernel='van
  illadot', C=100 )
  scores[idx[[i]]] <- predict( model, linear.data[idx[[i]],], typ
  e='decision' )
}

plot(performance(prediction(scores, linear.data$y), measure='tpr',
  x.measure='fpr' ),
  lwd = 2, col = "steelblue2",
  main = "Validation of the linear SVM model using 10-fold c
ross validation")
```

It can be observed that the ROC curve presented in Figure 7.12 approximates the ROC curve presented in Figure 7.11 well, showing that the ROC curve by 10-folder cross-validation is a good estimation of the ROC curve obtained on a testing dataset.

Since the 10-folder cross-validation could be used to obtain the prediction performance of any given model, it gives rise to the possibility that we could use it to compare different model formulations and decide on which model is the best. To do so, the R package “**caret**” could be used that has an automatic procedure dedicated for this.

```
# Cross-validation using caret pacakge
# install.packages("caret")
# install.packages("pROC")
# Training SVM Models
require(caret)

require(kernlab)      # support vector machine
require(pROC)          # plot the ROC curves

# Setup for cross validation
ctrl <- trainControl(method="repeatedcv",    # 10fold cross validation
                      repeats=5,           # do 5 repititions of cv
                      summaryFunction=twoClassSummary,   # Use AUC
                      to pick the best model
                      classProbs=TRUE)

#Train and Tune the SVM
linear.train <- data.frame(linear.train)
trainX <- linear.train[,1:2]
trainy= linear.train[,3]
trainy[which(trainy==1)] = rep("T",length(which(trainy==1)))
trainy[which(trainy== -1)] = rep("F",length(which(trainy== -1)))
svm.tune <- train(x = trainX,
                    y = trainy,
                    method = "svmLinear",   # Linear kernel
                    tuneLength = 9,          # 9 values of
                    the cost function
                    preProc = c("center","scale"), # Center and scale data
                    metric="ROC",
                    trControl=ctrl)

svm.tune
```

Then we can obtain that:

```
svm.tune

## Support Vector Machines with Linear Kernel
##
## 160 samples
## 2 predictor
## 2 classes: 'F', 'T'
##
## Pre-processing: centered (2), scaled (2)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 144, 144, 144, 145, 143, 144, ...
## Resampling results:
##
##      ROC      Sens      Spec
## 0.9625198 0.8964286 0.9055556
##
## Tuning parameter 'C' was held constant at a value of 1
```

While “**caret**” provides an automatic but sealed process to help us directly arrive the final end, the R package “**manipulate**” could be used to visualize the intermediate process. Here, we take some snapshots of this dynamic and interactive process and present these snapshots in Figures 7.13 and 7.14. It can be seen that, with larger value of C , a tighter margin could be obtained with less support vectors.

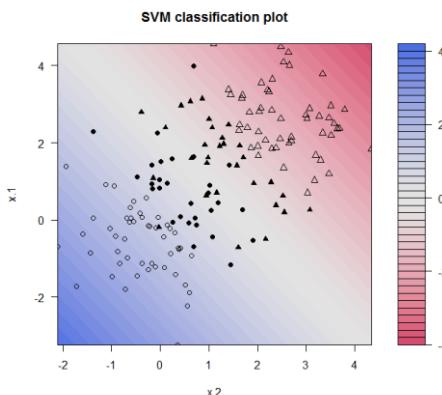


Figure 7.13: Visualization of the linear SVM model with $C = 0.01$.



Figure 7.14: Visualization of the linear SVM model with $C = 10$.

Let's further consider a nonlinear dataset.

```
# Generate a dataset with nonlinear boundary
n = 100
p = 2
bottom.left <- matrix(rnorm( n*p, mean=0, sd=1 ),n, p)
upper.right <- matrix(rnorm( n*p, mean=4, sd=1 ),n, p)
tmp1 <- matrix(rnorm( n*p, mean=0, sd=1 ),n, p)
tmp2 <- matrix(rnorm( n*p, mean=4, sd=1 ),n, p)
upper.left <- cbind( tmp1[,1], tmp2[,2] )
bottom.right <- cbind( tmp2[,1], tmp1[,2] )
y <- c( rep( 1, 2 * n ), rep( -1, 2 * n ) )
idx.train <- sample( 4 * n, floor( 3.5 * n ) )
is.train <- rep( 0, 4 * n )
is.train[idx.train] <- 1
nonlinear.data <- data.frame( x=rbind( bottom.left, upper.right,
upper.left, bottom.right ), y=y, train=is.train )

# Visualize the distribution of data points of two classes
require( 'ggplot2' )
p <- qplot( data=nonlinear.data, x.1, x.2, colour=factor(y) )
p <- p + labs(title = "Scatterplot of data points of two classes")
print(p)
```

As shown in Figure 7.15, this nonlinear dataset is similar to the XOR problem we have shown in the beginning of this section, in the sense that a similar style of classification boundary is needed.

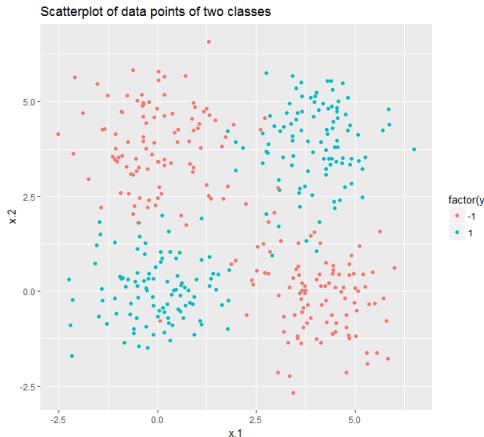


Figure 7.15: A randomly generated dataset with nonlinear boundary

Then, let's use the `kernlab` with Gaussian kernel (denoted as “`svmRadial`”) and the 10-folder cross-validation procedure in “`caret`” to train a nonlinear SVM model.

```
# Use cross-validation to choose C
# install.packages("caret")
# install.packages("pROC")
# Training SVM Models
require(caret)
require(kernlab)      # support vector machine
require(pROC)          # plot the ROC curves
# Setup for cross validation
ctrl <- trainControl(method="repeatedcv",    # 10fold cross validation
                      repeats=1,           # do 5 repititions of cv
                      summaryFunction=twoClassSummary,  # Use AUC
                      to pick the best model
                      classProbs=TRUE)

#Train and Tune the SVM
nonlinear.train <- data.frame(nonlinear.train)
trainX <- nonlinear.train[,1:2]
trainy= nonlinear.train[,3]
trainy[which(trainy==1)] = rep("T",length(which(trainy==1)))
trainy[which(trainy== -1)] = rep("F",length(which(trainy== -1)))
svm.tune <- train(x = trainX,
```

```

        y = trainy,
        method = "svmRadial",   # Radial kernel
        tuneLength = 9,          # 9 values of
the cost function
        preProc = c("center","scale"), # Center and sc
ale data
        metric="ROC",
        trControl=ctrl)

svm.tune

```

Details of the model tuning by the cross-validation process is shown in below:

```

## Support Vector Machines with Radial Basis Function Kernel
##
## 350 samples
## 2 predictor
## 2 classes: 'F', 'T'
##
## Pre-processing: centered (2), scaled (2)
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 315, 315, 315, 315, 315, ...
## Resampling results across tuning parameters:
##
##     C      ROC      Sens      Spec
## 0.25  0.9878913  0.9584559  0.9502924
## 0.50  0.9869174  0.9584559  0.9502924
## 1.00  0.9872463  0.9643382  0.9502924
## 2.00  0.9826561  0.9525735  0.9558480
## 4.00  0.9797171  0.9584559  0.9502924
## 8.00  0.9754708  0.9584559  0.9558480
## 16.00 0.9735144  0.9525735  0.9447368
## 32.00 0.9709086  0.9466912  0.9502924
## 64.00 0.9659980  0.9290441  0.9391813
##
## Tuning parameter 'sigma' was held constant at a value of 1.834
54
## ROC was used to select the optimal model using the largest va
lue.
## The final values used for the model were sigma = 1.83454 and C
= 0.25.

```

Again, we can use the package “**manipulate**” to see how the model changes according to the parameters such as C and even kernel types.

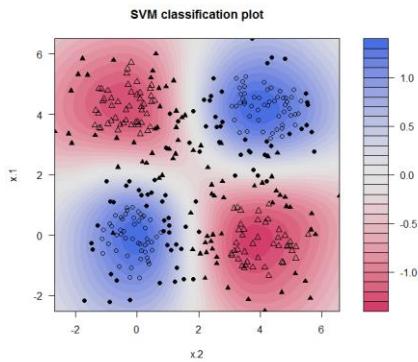


Figure 7.16: SVM model with $C = 0.01$ and Gaussian kernel

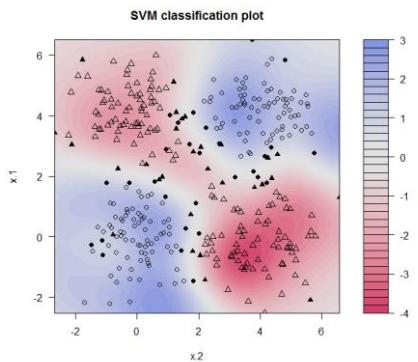


Figure 7.17: SVM model with $C = 10$ and Gaussian kernel

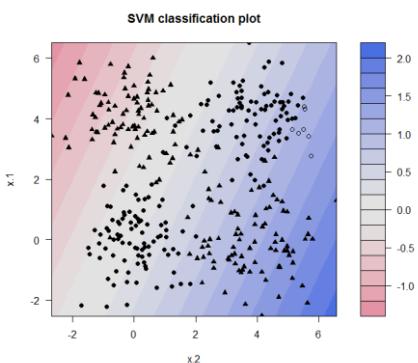


Figure 7.18: SVM model with $C = 0.1$ and Laplacian kernel

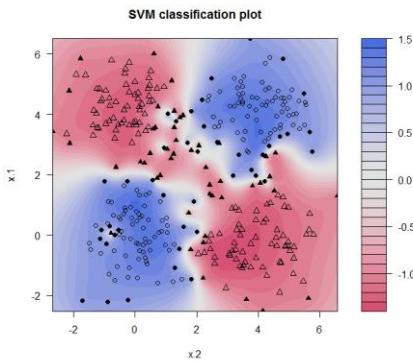


Figure 7.19: SVM model with $C = 10$ and Laplacian kernel

Finally, let's implement the above process on the AD dataset.

```
#### Dataset of Alzheimer's Disease
#### Objective: prediction of diagnosis
# filename
AD <- read.csv('AD_b1.csv', header = TRUE)
str(AD)

#Train and Tune the SVM
n = dim(AD)[1]
n.train <- floor(0.8 * n)
idx.train <- sample(n, n.train)
AD[which(AD[,1]==0),1] = rep("Normal",length(which(AD[,1]==0)))
AD[which(AD[,1]==1),1] = rep("Diseased",length(which(AD[,1]==1)))
AD.train <- AD[idx.train,c(1:16)]
AD.test <- AD[-idx.train,c(1:16)]
trainX <- AD.train[,c(2:16)]
trainy= AD.train[,1]

# Setup for cross validation
ctrl <- trainControl(method="repeatedcv",    # 10fold cross validation
                      repeats=1,           # do 5 repetitions of cv
                      summaryFunction=twoClassSummary,   # Use AUC
                      to pick the best model
                      classProbs=TRUE)

# Use the expand.grid to specify the search space
grid <- expand.grid(sigma = c(0.002, 0.005, 0.01, 0.012, 0.015),
                     C = c(0.3,0.4,0.5,0.6)
)
```

```

svm.tune <- train(x = trainX,
                    y = trainy,
                    method = "svmRadial",    # Radial kernel
                    tuneLength = 9,           # 9 values of
the cost function
                    preProc = c("center", "scale"), # Center and scale data
                    metric="ROC",
                    tuneGrid = grid,
                    trControl=ctrl)

svm.tune

```

Then we can obtain the following results.

```

## Support Vector Machines with Radial Basis Function Kernel
##
## 413 samples
## 15 predictor
## 2 classes: 'Diseased', 'Normal'
##
## Pre-processing: centered (15), scaled (15)
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 371, 372, 372, 371, 372, 372, ...
## Resampling results across tuning parameters:
##
##     sigma   C    ROC      Sens      Spec
## 0.002  0.3  0.8929523  0.9121053  0.5932900
## 0.002  0.4  0.8927130  0.8757895  0.6619048
## 0.002  0.5  0.8956402  0.8452632  0.7627706
## 0.002  0.6  0.8953759  0.8192105  0.7991342
## 0.005  0.3  0.8965129  0.8036842  0.8036797
## 0.005  0.4  0.8996565  0.7989474  0.8357143
## 0.005  0.5  0.9020830  0.7936842  0.8448052
## 0.005  0.6  0.9032422  0.7836842  0.8450216
## 0.010  0.3  0.9030514  0.7889474  0.8541126
## 0.010  0.4  0.9058248  0.7886842  0.8495671
## 0.010  0.5  0.9060999  0.8044737  0.8541126
## 0.010  0.6  0.9077848  0.8094737  0.8450216
## 0.012  0.3  0.9032308  0.7781579  0.8538961
## 0.012  0.4  0.9049043  0.7989474  0.8538961
## 0.012  0.5  0.9063505  0.8094737  0.8495671
## 0.012  0.6  0.9104511  0.8042105  0.8586580
## 0.015  0.3  0.9060412  0.7886842  0.8493506
## 0.015  0.4  0.9068165  0.8094737  0.8495671
## 0.015  0.5  0.9109051  0.8042105  0.8541126
## 0.015  0.6  0.9118615  0.8042105  0.8632035

```

```
##  
## ROC was used to select the optimal model using the largest value.  
## The final values used for the model were sigma = 0.015 and C = 0.6.
```

It can be seen that, by 10-folder cross-validation, the best model parameters are `sigma = 0.015` and `C = 0.6`, which achieve a prediction performance as 90.49%.

II.4 Remarks

Is SVM a more complex model? Here, we need to look at the term “complexity” carefully. What is a complex model? Comparing with linear regression model or logistic regression model, the idea of SVM and formulation of SVM indeed seems more complex. This is probably true. The inventor of SVM, Vladimir Vapnik, once said in the preface of his seminar book¹ that delineates the theory of SVM, that he often heard peers talked in conferences that complex models don’t work but simple models work. He thought, SVM, commonly perceived as a more complex model, is essentially simpler than some simple model. This is true, since some models that look simple are only because they presuppose stronger conditions, which make them essentially more complex!

Thus, a model is more complex than another model doesn’t necessary stem from the fact that the more complex one employs a more sophisticated mathematical representation. At least, in our current context, it doesn’t mean in this way when we say a model is complex. Here, we say that a model is more complex if it provides more capacity to represent the statistical phenomena in the training data. In other words, a more complex model is more flexible of responding to any patterns in the data by adjusting itself. Now, look at Figure 7.20, which model is simpler?

¹ Vapnik, V. *The nature of statistical learning theory*. Springer, 2000.

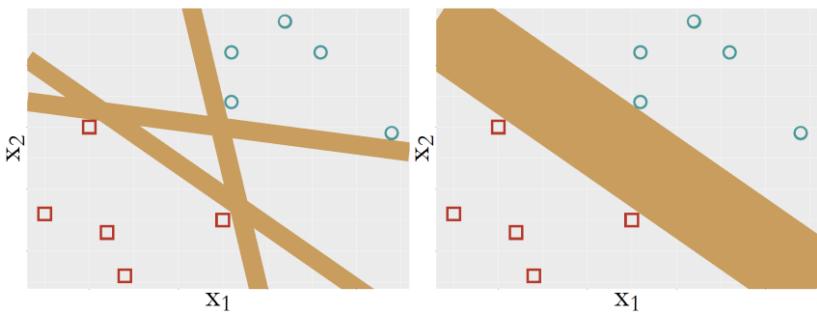


Figure 7.20: SVM is actually a simpler model

Is SVM a neural network model? Another interesting fact about SVM is that, when it was developed, it was named as “support vector network”. In other words, it has a connection with the artificial neural network. This is revealed in the Figure 7.21. Readers who know neural network models are encouraged to write up the mathematical model of the SVM model following the neural network format as shown in Figure 7.21.

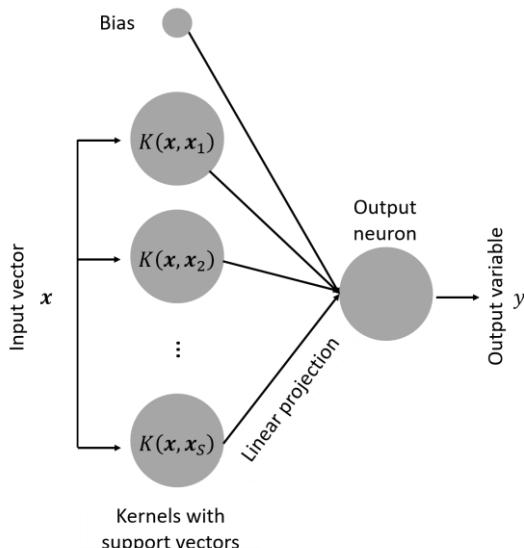


Figure 7.21: SVM as a neural network model

III. Ensemble Learning

III.1 Rationale and Formulation

As we mentioned in the begining of this chapter, the random forest model is a particular case of the more general category of models that are called ensemble models. Ensemble models consist of multiple base models, denoted as, h_1, h_2, \dots, h_K , where K is the total number of base models. Each model can be considered as a **hypothesis** in the space of \mathcal{H} that includes all the possible models. A general framework of ensemble learning is illustrated in Figure 7.22. Each model is built on a sample that is created from the original dataset. Recall that, in random forest models, each tree is built on an independently bootstrapped sample (also referred to as bagging), but this is not the only approach we can create a new dataset from the original dataset. For instance, in Adaboosting, the sample for a base model is not independently created. Rather, it also depends on the error rates from previous base models. In other words, it takes an adapative and sequential approach to grow its base models, while later models more focus on the hard data points that present challenges for previous base models to achieve good prediction performance.

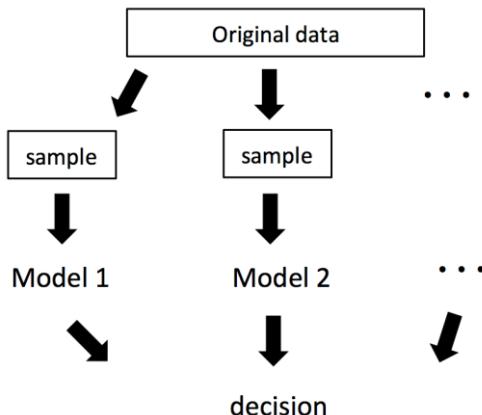


Figure 7.22: A general framework of ensemble learning

III.2 Theory/Method

As shown in Figure 7.22, the ensemble learning is very flexible as different approaches could be combined to create new samples and build new base models. It has been known that ensemble learning is very powerful in practices and has been reported as the winner methods in numerous data science competitions. Just like SVM, it is another main approach to handle the risk of overfitting in practice. Here, we use the framework proposed by Dietterich (2000)¹, where three perspectives (statistical, computational, and representational) were articulated to explain why ensemble methods could lead to excellent accuracy performance. Each perspective is described in detail below.

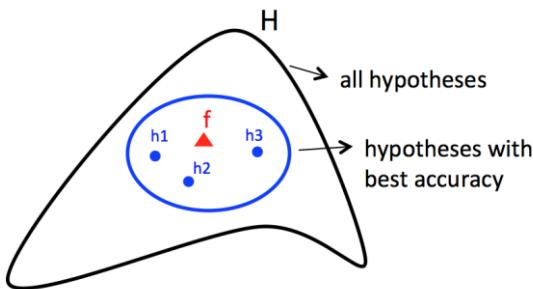


Figure 7.23: Ensemble learning approximates the true model with a combination of good models (statistical perspective)

Statistical perspective: The statistical reason is illustrated in Figure 7.23. \mathcal{H} is the hypothesis space where a learning algorithm searches for the best one guided by the training data. f is the true function. The statistical problem occurs when there are limited data, and there are multiple best hypotheses. In other words, multiple models can achieve the best accuracy on the training data. This is illustrated by the inner circle in Figure 7.23. By

¹ Dietterich, T.G. *Ensemble methods in machine learning. Multiple classifier systems*, 2000.

building an ensemble of multiple learners, e.g., h_1 , h_2 , and h_3 , the average of the hypotheses is a good approximation to the true hypothesis f . Therefore, the average of the multiple hypotheses essentially approximates a solution with the minimum variance in the inner circle.

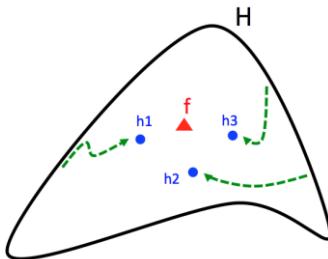


Figure 7.24: Ensemble learning approximates the true model with a combination of good models (computational perspective)

Computational perspective: A computational perspective is shown in Figure 7.24. This is related to the way we build base models, which is usually a greedy approach such as the recursive splitting procedure we have shown in decision tree models. Many machine learning models are complex models that present NP-hard optimization problems in training these models including neural networks and decision tree models. E.g., in decision trees, at each node, the node is split according to the maximum information gain. However, only the current node is evaluated, and it may result in suboptimal situations for further splitting of descendant nodes. Growing an optimal tree model is thus NP-hard and computationally expensive. This computational prospective is shown in Figure 7.24, while the learning algorithm of greedy and heuristic nature with a certain parameter setting searches for the best hypothesis in the hypothesis space. The search paths of three hypotheses are illustrated in Figure 7.24. The differences of the three hypotheses can be attributed to different parameter settings or different input data (e.g., bootstrap samples). However, growing

multiple learning algorithms and averaging them in joint decision makings, may reasonably approximate the true hypothesis f .

Representational perspective: Due to the size of the dataset or the limitations of a learning algorithm, sometimes the hypothesis space \mathcal{H} does not cover the true function, as shown in Figure 7.25. For example, linear models cannot learn non-linear patterns, and decision trees with limited data have difficulty learning linear patterns. Using a weighted sum of the outcomes from the base learners may be able to approximate a function outside \mathcal{H} . This is shown in Figure 7.25, while the true function f is outside the space, but a combination of h_1 , h_2 and h_3 can approximate the true function.

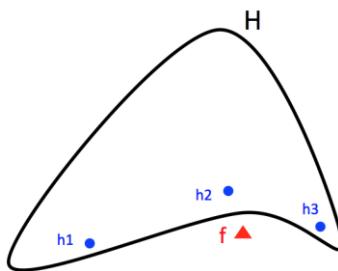


Figure 7.25: Ensemble learning approximates the true model with a combination of good models (representational perspective)

Now we discuss three methods, single decision trees, random forests, and AdaBoost (adaptive boosting) under the framework of ensemble learning.

Single decision tree: A single decision tree lacks capability to overcome overfitting in terms of all the three perspectives. From the statistical perspective, a decision tree algorithm constructs each node using the maximum information gain and is sensitive to the training data. While the training dataset is limited, the possible models achieving the best

accuracy can be large. Consequently, the learning algorithm may end up with any one of these models, which maybe far away from the true hypothesis.

Single decision trees also have the computational issue. Decision trees are greedy implementations, seeking for maximum impurity gain at each node. Decisions made in upstream nodes would affect downstream nodes very much. And it is NP-hard to find an optimal decision tree which achieves the maximum impurity gain at all nodes.

Representational perspective also shows limitations of the decision tree model. Although decision trees can approximate a wide range of functions, it needs sufficient data to achieve an accurate approximation. Given limited training data, the possible hypothesis space of a single decision tree may not be able to cover the true function, e.g., if the true function is a linear or any smooth function.

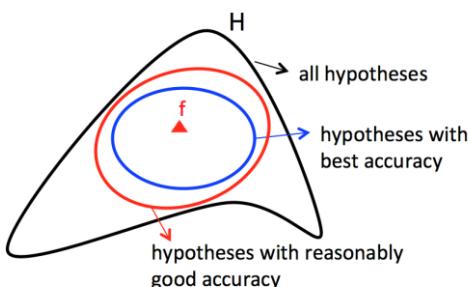


Figure 7.26: Analysis of the random forest in terms of the statistical perspective

Random forests: Random forests construct multiple hypotheses by two ways. First, each tree in random forests is built on a different bootstrapped sample. Actually, this framework that builds each base learner based on a bootstrapped sample is referred to as bagging in general. Second, at each node, a subset of variables is randomly selected and the best variable from the subset is used for splitting. It addresses the statistical issue. Note that, each tree is injected with randomness, and therefore, is not necessarily in

the best-accuracy hypothesis circle, but lies in the hypothesis space with reasonably good accuracy. Averaging the outcomes from all trees (or hypotheses) would achieve the minimum variance in the reasonably-good-accuracy space. Assuming the best-accuracy space has a similar shape with the reasonably-good space just with a smaller size, the solution may also achieve a reasonably small variance in the best-accuracy space. This is illustrated in Figure 7.26.

Random forests can also address the computational issue. As shown in Figure 7.27, the inner circle represents the space that is computationally difficult to reach. However, averaging multiple hypotheses could get into the inner space.

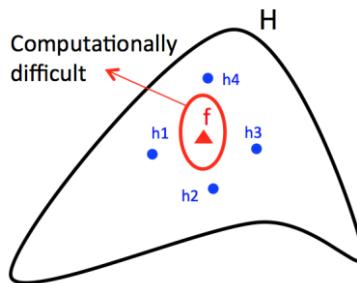


Figure 7.27: Analysis of the random forest in terms of the computational perspective

Random forests do not necessarily, or actively, solve the representational issue. If the true function lies outside \mathcal{H} , averaging the outcomes from all trees won't necessarily approximate the true function. Figure 7.28 shows that random forests would construct multiple hypotheses that randomly spread over the \mathcal{H} space. Averaging them won't reach the true function f .

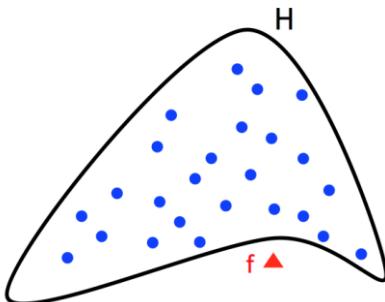


Figure 7.28: Analysis of the random forest in terms of the representational perspective

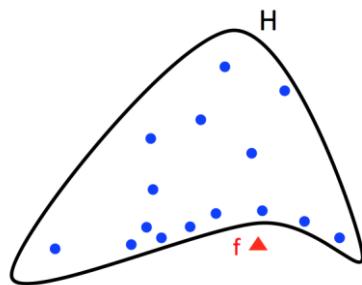


Figure 7.29: Analysis of the AdaBoost in terms of the representational perspective

AdaBoost: Unlike random forests that build each tree independently, AdaBoost builds trees sequentially. For each tree, the training dataset is sampled not by bootstrap, but by a weight determined by the error rates from previous trees. Data points that are difficult to be correctly predicted by the previous trees will be given more weights in the new training dataset for new trees. When all the base learners are trained, the aggregation of these models in predicting on a data instance \mathbf{x} is a weighted sum of base learners

$$h(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x}),$$

where the weight w_i is determined by the accuracy of learner $h_i(\mathbf{x})$.

Similar to random forests, AdaBoost solves the computational issue by generating many base learners that are built on randomly generated datasets. Different from random forests, AdaBoost actively solves the representational issue as it tries to reduce the residual errors coming from previous trees. Figure 7.29 shows that, AdaBoost could construct more hypotheses around the true function, and also could put more weight to the hypotheses that are closer to the true function by using the weighted sum of base learners in aggregation of the base learners.

But AdaBoost is not as good as random forests in terms of addressing the statistical issue. AdaBoost handles the overfitting issue through the concept of margin. Suppose each data point is labeled as -1 or 1, then the margin of a classifier of a data point (x_i, y_i) is defined as

$$m_i = y_i h(x_i).$$

It can be shown that AdaBoost tries to minimize

$$\sum_i \exp(-y_i \sum_j w_j h_j(x_j)),$$

which can be considered as an effort to minimize the margin on the training data. However, as AdaBoost aggressively solves the representational issue, and optimizes for the training data, it is more likely to overfit, and may be less stable than random forests that place more emphasis on addressing the statistical issue.

III.3 R Lab

A single decision tree (`rpart`), random forests (`randomForest`), and AdaBoost (`gbm`) are applied to the AD dataset `AD_b1.csv`. First, we change the percentage of training data, and 50 replicates of data are generated. The boxplots of the classification error rates for single decision tree, random forests, and AdaBoost are plotted at different percentages of training data by the following R code, which is shown in Figure 7.30.

```
library(dplyr)
library(tidyverse)
library(ggplot2)
require(randomForest)
require(gbm)
require(rpart)
```

```

set.seed(1)

theme_set(theme_gray(base_size = 15))

path <- "../../data/AD_b1.csv"
data <- read.csv(path, header = TRUE)
rm_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))
data <- data[, -rm_indx]
data$DX_b1 <- as.factor(data$DX_b1)
# target_indx <- which( colnames(data) == 'DX_b1' ) target <-
# data[,target_indx] rm_indx <- which( colnames(data) %in%
# c('DX_b1', 'ID', 'TOTAL13', 'MMSCORE') ) X <- data X <- X[,-rm_indx]

set.seed(1)

err.mat <- NULL
for (K in c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7)) {

  testing.indices <- NULL
  for (i in 1:50) {
    testing.indices <- rbind(testing.indices, sample(nrow(data),
a), floor((1 -
      K) * nrow(data))))
  }

  for (i in 1:nrow(testing.indices)) {

    testing.ix <- testing.indices[i, ]
    target.testing <- data$DX_b1[testing.ix]

    tree <- rpart(DX_b1 ~ ., data[-testing.ix, ])
    pred <- predict(tree, data[testing.ix, ], type = "class")
    error <- length(which(as.character(pred) != target.testing))/length(target.testing)
    err.mat <- rbind(err.mat, c("tree", K, error))

    rf <- randomForest(DX_b1 ~ ., data[-testing.ix, ])
    pred <- predict(rf, data[testing.ix, ])
    error <- length(which(as.character(pred) != target.testing))/length(target.testing)
    err.mat <- rbind(err.mat, c("RF", K, error))

    data1 <- data
    data1$DX_b1 <- as.numeric(as.character(data1$DX_b1))
    boost <- gbm(DX_b1 ~ ., data = data1[-testing.ix, ], dist
  }
}

```

```

= "adaboost",
    interaction.depth = 6, n.tree = 2000) #cv.folds = 5,
    # best.iter <- gbm.perf(boost, method='cv')
    pred <- predict(boost, data1[testing.ix, ], n.tree = 2000,
type = "response") # best.iter n.tree = 400,
    pred[pred > 0.5] <- 1
    pred[pred <= 0.5] <- 0
    error <- length(which(as.character(pred) != target.testing))/length(target.testing)
    err.mat <- rbind(err.mat, c("AdaBoost", K, error))
}
err.mat <- as.data.frame(err.mat)
colnames(err.mat) <- c("method", "training_percent", "error")
err.mat <- err.mat %>% mutate(training_percent = as.numeric(as.character(training_percent)),
error = as.numeric(as.character(error)))

ggplot() + geom_boxplot(data = err.mat %>% mutate(training_percent = as.factor(training_percent)),
aes(y = error, x = training_percent, color = method)) + geom_point(size = 3)

```

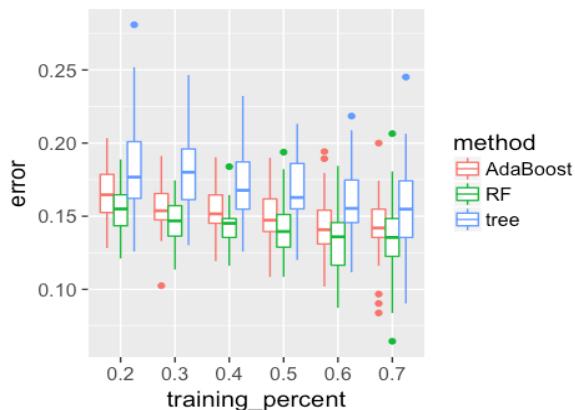


Figure 7.30: Boxplots of the classification error rates for single decision tree, random forests, and AdaBoost

It can be seen in Figure 7.30 that, all error rates are reduced as the percentage of the training data increases. The single decision tree is clearly less accurate than the other two ensemble methods. RF has lower error rates than AdaBoost in general. However, as the training data size increases, the gap between RF and AdaBoost seems to decrease slightly. This may indicate that when the training data size is small, RF is more stable due to its advantage of addressing the statistical issue.

Now we add model complexity to each model to see its impacts on the models' performance. First, we change the complexity parameter (`cp`) in decision tree. A smaller `cp` indicates a larger tree. It can be seen that the error rate decreases when the tree gets more complex, and only slightly increases after `cp` is greater than 0.02. This may indicate that, for this dataset, the main issue for decision tree may stem from the representational perspective, meaning that, a single decision is not able to capture enough information (as much as ensemble methods) using the training data. But the statistical issue is not severe, given that the error does not increase substantially given a complexity parameter value that could result in a large tree.

```
set.seed(1)
testing.indices <- NULL
for (i in 1:50) {
  testing.indices <- rbind(testing.indices, sample(nrow(data),
floor((0.3) *
      nrow(data))))
}

err.mat <- NULL
for (i in 1:nrow(testing.indices)) {
  testing.ix <- testing.indices[i, ]
  target.testing <- data$DX_bl[testing.ix]

  cp.v <- rev(c(0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.
08, 0.09, 0.1))
  for (j in cp.v) {
    tree <- rpart(DX_bl ~ ., data[-testing.ix, ], cp = j)
    pred <- predict(tree, data[testing.ix, ], type = "class")
    error <- length(which(as.character(pred) != target.testin
g))/length(target.testing)
```

```

        err.mat <- rbind(err.mat, c("Tree", j, error))
    }

err.mat <- as.data.frame(err.mat)
colnames(err.mat) <- c("method", "cp", "error")
err.mat <- err.mat %>% mutate(cp = as.numeric(as.character(cp)),
error = as.numeric(as.character(error)))
err.mat$cp <- factor(err.mat$cp, levels = sort(cp.v, decreasing =
TRUE))

ggplot() + geom_boxplot(data = err.mat, aes(y = error, x = cp, co
lor = method)) +
    geom_point(size = 3)

```

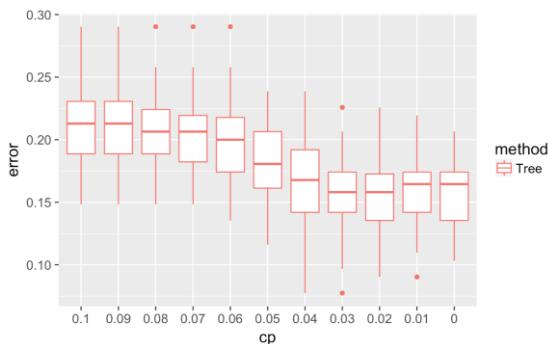


Figure 7.31: Boxplots of the classification error rates for single decision tree models with different model complexity (by controlling cp)

We also adjust the number of trees in AdaBoost and show the results in Figure 7.32. It can be seen that the error rates first go down as the number of trees increases to 400. However, the error rates increase after that, and decrease again. This may indicate that AdaBoost still have a statistical issue as the error rates are not stable.

```

err.mat <- NULL
set.seed(1)
for (i in 1:nrow(testing.indices)) {

```

```

data1 <- data
data1$DX_b1 <- as.numeric(as.character(data1$DX_b1))
ntree.v <- c(200, 300, 400, 500, 600, 800, 1000, 1200, 1400,
1600, 1800,
2000)
for (j in ntree.v) {
  boost <- gbm(DX_b1 ~ ., data = data1[-testing.ix, ], dist
= "adaboost",
    interaction.depth = 6, n.tree = j)
  # best.iter <- gbm.perf(boost,method='cv')
  pred <- predict(boost, data1[testing.ix, ], n.tree = j, t
ype = "response")
  pred[pred > 0.5] <- 1
  pred[pred <= 0.5] <- 0
  error <- length(which(as.character(pred) != target.testing))/length(target.testing)
  err.mat <- rbind(err.mat, c("AdaBoost", j, error))
}
err.mat <- as.data.frame(err.mat)
colnames(err.mat) <- c("method", "num_trees", "error")
err.mat <- err.mat %>% mutate(num_trees = as.numeric(as.character
(num_trees)),
  error = as.numeric(as.character(error)))

ggplot() + geom_boxplot(data = err.mat %>% mutate(num_trees = as.
factor(num_trees)),
  aes(y = error, x = num_trees, color = method)) + geom_point(s
ize = 3)

```

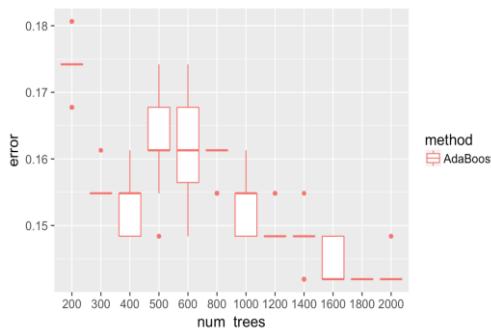


Figure 7.32: Boxplots of the classification error rates for AdaBoost with different number of trees

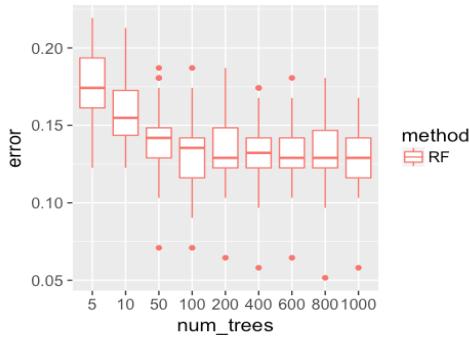


Figure 7.33: Boxplots of the classification error rates for random forest with different number of trees

Now let's look at random forests with different number of trees. Results are shown in Figure 7.33. Similar to AdaBoost, RF has high error rates initially at a small number of trees. Then, the error rates are reduced as more trees are added. However, the error rates become stable when more trees are added. This may indicate that RF handles the statistical issue well.

```

err.mat <- NULL
set.seed(1)
for (i in 1:nrow(testing.indices)) {
  testing.ix <- testing.indices[i, ]
  target.testing <- data$DX_bl[testing.ix]

  ntree.v <- c(5, 10, 50, 100, 200, 400, 600, 800, 1000)
  for (j in ntree.v) {
    rf <- randomForest(DX_bl ~ ., data[-testing.ix, ], ntree
= j)
    pred <- predict(rf, data[testing.ix, ])
    error <- length(which(as.character(pred) != target.testin
g))/length(target.testing)
    err.mat <- rbind(err.mat, c("RF", j, error))
  }
}
err.mat <- as.data.frame(err.mat)
colnames(err.mat) <- c("method", "num_trees", "error")
err.mat <- err.mat %>% mutate(num_trees = as.numeric(as.character
(num_trees)),
  error = as.numeric(as.character(error)))

```

```
ggplot() + geom_boxplot(data = err.mat %>% mutate(num_trees = as.factor(num_trees)),
  aes(y = error, x = num_trees, color = method)) + geom_point(size = 3)
```

Recall that, a requirement to solve the statistical issue in random forests is that a diverse set of learners need to be built. As we have mentioned, in random forests, there are two approaches to increase diversity, one is to bootstrap samples for each tree while another one is to conduct random feature selection for splitting each node. First, we investigate the effectiveness of using randomly bootstrapped samples. We change sampling strategy from sampling with replacement to sampling without replacement, and change the sampling size from 10% to 100% (with the number of features tested at each node being the default value of \sqrt{p} where p is the number of features). The results as shown in Figure 7.34 do not show that the increased sample size has an impact on the error rates on this particular dataset.

```
err.mat <- NULL
set.seed(1)
for (i in 1:nrow(testing.indices)) {
  testing.ix <- testing.indices[i, ]
  target.testing <- data$DX_bl[testing.ix]

  sample.size.v <- seq(0.1, 1, by = 0.1)
  for (j in sample.size.v) {
    sample.size <- floor(nrow(data[-testing.ix, ]) * j)
    rf <- randomForest(DX_bl ~ ., data[-testing.ix, ], sample.size = sample.size,
      replace = FALSE)
    pred <- predict(rf, data[testing.ix, ])
    error <- length(which(as.character(pred) != target.testing))/length(target.testing)
    err.mat <- rbind(err.mat, c("RF", j, error))
  }
}
err.mat <- as.data.frame(err.mat)
colnames(err.mat) <- c("method", "sample_size", "error")
err.mat <- err.mat %>% mutate(sample_size = as.numeric(as.character(sample_size)),
  error = as.numeric(as.character(error)))
```

```
ggplot() + geom_boxplot(data = err.mat %>% mutate(sample_size = as.factor(sample_size)),
  aes(y = error, x = sample_size, color = method)) + geom_point(size = 3)
```

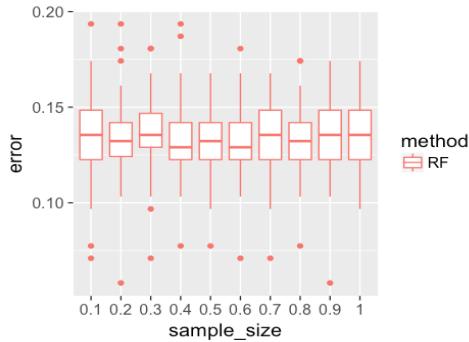


Figure 7.34: Boxplots of the classification error rates for random forest with different sample sizes

We then set the number of samples used at each tree the same as the size of the original training data set, and change the number of features in growing the random forest models. As shown in Figure 7.35, we can see that when the number of features is sufficiently large, the error rates start to increase, which could be due to a low diversity and indicate its compromised capability to address the statistical issue.

```
err.mat <- NULL
set.seed(1)
for (i in 1:nrow(testing.indices)) {
  testing.ix <- testing.indices[i, ]
  target.testing <- data$DX_bl[testing.ix]

  numfea.v <- 1:(ncol(data) - 1)
  for (j in numfea.v) {
    sample.size <- nrow(data[-testing.ix, ])
    rf <- randomForest(DX_bl ~ ., data[-testing.ix, ], mtry = j,
      sampsize = sample.size,
      replace = FALSE)
    pred <- predict(rf, data[testing.ix, ])
    error <- length(which(as.character(pred) != target.testin
```

```

g))/length(target.testing)
    err.mat <- rbind(err.mat, c("RF", j, error))
}
}
err.mat <- as.data.frame(err.mat)
colnames(err.mat) <- c("method", "num_fea", "error")
err.mat <- err.mat %>% mutate(num_fea = as.numeric(as.character(n
um_fea)), error = as.numeric(as.character(error)))

ggplot() + geom_boxplot(data = err.mat %>% mutate(num_fea = as.fa
ctor(num_fea)),
    aes(y = error, x = num_fea, color = method)) + geom_point(siz
e = 3)

```

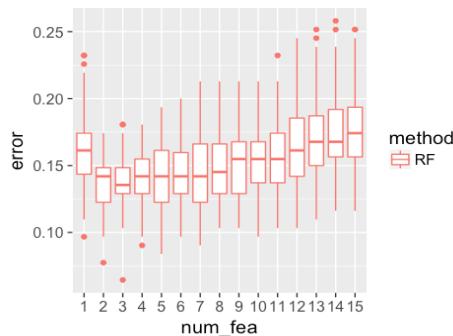


Figure 7.35: Boxplots of the classification error rates for random forest with different number of features

In the next experiment, we fix the number of tested features to be the total number of variables, and change the percentage of samples to be used at each tree. As shown in Figure 7.36, the error rate is relatively large when the percentage of samples is small (10%). But as more samples are used, the error rates increase, and reach the highest error rate when 100% of the samples are used (i.e., which leads to least diversity among the base learners).

```

err.mat <- NULL
set.seed(1)
for (i in 1:nrow(testing.indices)) {

```

```

testing.ix <- testing.indices[i, ]
target.testing <- data$DX_bl[testing.ix]

sample.size.v <- seq(0.1, 1, by = 0.1)
for (j in sample.size.v) {
  traing.data <- data[-testing.ix, ]
  sample.size <- floor(nrow(traing.data) * j)
  rf <- randomForest(DX_bl ~ ., traing.data, mtry = ncol(traing.data) -
    1, sampsize = sample.size, replace = FALSE)
  pred <- predict(rf, data[testing.ix, ])
  error <- length(which(as.character(pred) != target.testing))/length(target.testing)
  err.mat <- rbind(err.mat, c("RF", j, error))
}
err.mat <- as.data.frame(err.mat)
colnames(err.mat) <- c("method", "num_fea", "error")
err.mat <- err.mat %>% mutate(num_fea = as.numeric(as.character(num_fea)), error = as.numeric(as.character(error)))

ggplot() + geom_boxplot(data = err.mat %>% mutate(num_fea = as.factor(num_fea)),
  aes(y = error, x = num_fea, color = method)) + geom_point(size = 3)

```

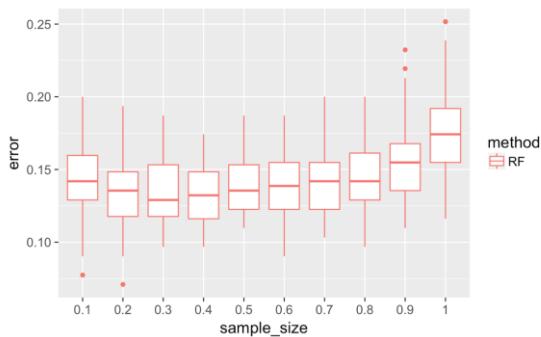


Figure 7.36: Boxplots of the classification error rates for random forest with different sample sizes

IV. Exercises

Data analysis

1. Find ten classification datasets from the UCI data repository or R datasets. Conduct a detailed analysis using the logistic regression model, SVM, decision tree, random forest, and AdaBoost. Conduct model selection and validation. Use cross-validation to select the best models. Conduct residual analysis of your final models, and comment on your results.

Programming

2. Write your own R script to implement the linear SVM model.
3. Extend your SVM code to nonlinear SVM models with Gaussian kernel and polynomial kernel. Compare your results with `ksvm()`.
4. Write your own R script to implement the AdaBoost model. Compare your results with `gbm()`.

CHAPTER 8: SCALABILITY

LASSO AND PCA

I. Overview

Chapter 7 is about “**Scalability**”. It is to enhance our capacity to deal with large-scale problems – strength to be scalable. LASSO and PCA will be introduced in this chapter. LASSO stands for the Least Absolute Shrinkage and Selection Operator, which is a main representative method for feature selection. PCA stands for the Principle Component Analysis, which is a main representative method for dimension reduction. Both methods can reduce the dimensionality of the dataset, but follow different styles. LASSO, as a feature selection method, focuses on deletion of irrelevant or redundant features in the dataset. PCA, as a dimension reduction method, keeps all the features but combine them into a smaller number of aggregated new features.

II. LASSO

II.1 Rationale and Formulation

LASSO was invented in 1996¹ that was used to sparsify the linear regression model and allowed the regression model to select significant predictors automatically. The formulation of LASSO is

$$\widehat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} \{ \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \},$$

where $\mathbf{y} \in \mathbb{R}^{N \times 1}$ is the measurement vector of the response, $\mathbf{X} \in \mathbb{R}^{N \times p}$ is the data matrix of the N measurement vectors of the p predictors, $\boldsymbol{\beta} \in \mathbb{R}^{p \times 1}$ is the regression coefficient vector. Here, $\|\boldsymbol{\beta}\|_1 = \sum_{i=1}^p |\beta_i|$. Note that, here, the intercept (i.e., β_0) is not included, since we assume that the data is normalized (i.e., $\sum_{n=1}^N x_{nj}/N = 0$, $\sum_{n=1}^N x_{nj}^2/N = 1$ for $j = 1, 2, \dots, p$ and $\sum_{n=1}^N y_n/N = 0$), and thus, the intercept is not needed.

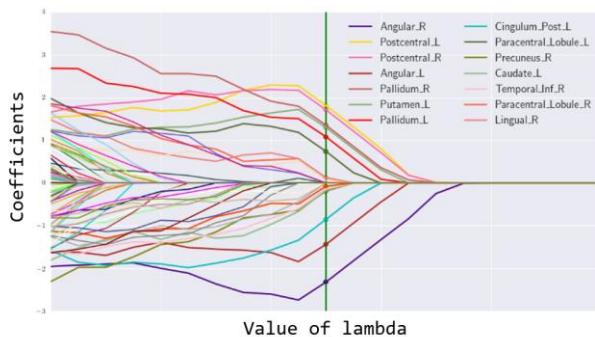


Figure 8.1: Path solution trajectory of the coefficients of LASSO, identifying brain regions that show longitudinal declines that can separate early-stage Alzheimer's patients from normal elderly.

¹ Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 1996.

It could be seen that LASSO embodies two major components in its formulation. The 1st term is the least squares loss function from linear regression, that is used to measure the goodness-of-fit of the model. The 2nd term is the sum of absolute values of the elements in β , representing the model complexity, i.e., the smaller the $\|\beta\|_1$, more zeros in β , leading to a simpler model. The parameter, λ , is called the penalty parameter that is specified by user of LASSO. In other words, LASSO suggests the best model by an optimal balance between model fit and model complexity, and this balance could be flexibly tuned by tuning the parameter λ . Furthermore, as shown in Figure 8.1, LASSO can generate the path solution trajectory that visualizes the solutions of β for a continuum of values of λ , giving us a global sense of the relationships between variables. Also, model selection criteria such as Akaike Information Criteria (AIC) or cross-validation can be used to identify the best λ .

II.2 Theory and Method

Why LASSO uses the L_1 norm: The popularity of LASSO and its enormous impact on statistical/machine learning research in the last decade needs no exaggeration. Some researchers in optimization and operations research often found puzzling is why all of sudden LASSO was invented and gave birth to the area of sparse learning. To answer this question, LASSO is often compared with another similar model, called **Ridge regression**¹ that has been developed almost half a century ago.

The formulation of Ridge regression is

$$\hat{\beta} = \arg \min_{\beta} \{ \|y - X\beta\|_2^2 + \lambda \|\beta\|_2 \},$$

where $\|\beta\|_2 = \sum_{i=1}^p |\beta_i|^2$ is called the L_2 norm.

¹ Hoerl, A.E. and Kennard, R.W. Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, 1970.

At the first glance, it seems that the Ridge regression bears the same spirit of LASSO – they both penalize the magnitudes of the regression parameters. However, it has been noticed that, in the Ridge regression model, the regression parameters in β will not achieve exactly zero. Even if you impose a very large λ , many elements in β may be close to zero with a very tiny numerical magnitude, but not zero. Although the numerical magnitudes of these elements are close to zero, and thus, seems to be insignificant, they are still in the estimation system and generate impacts on the estimation of other regression parameters. Thus, it is often reported that when you run Ridge regression and LASSO regression, you may not observe the same set of predictors that are selected by both methods. Ridge regression is more often used as a stabilization strategy to handle the multicollinearity issue or any other issues that result in numerical instability in parameter estimation, while LASSO is used as a variable selection strategy.

Shooting algorithm to solve the optimization problem of LASSO:

We could denote the objective function of LASSO as

$$L(\beta) = \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda\|\beta\|_1.$$

To see how the LASSO can be iteratively solved, let's first consider a simple case where there is only one predictor. Then, the objective function becomes

$$L(\beta) = \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda|\beta|.$$

To find the optimal solution, we can solve the equation as

$$\frac{\partial L(\beta)}{\partial \beta} = 0.$$

The complication is the L1-norm term, $|\beta|$, which has no gradient when $\beta = 0$. Thus, we can discuss different scenarios and identify the solutions.

- If $\beta > 0$, then $\frac{\partial L(\beta)}{\partial \beta} = 2\beta - 2\mathbf{X}^T \mathbf{y} + \lambda$. Thus, $\frac{\partial L(\beta)}{\partial \beta} = 0$ will lead to the solution that $\beta = \frac{(2\mathbf{X}^T \mathbf{y} - \lambda)}{2}$. But if $2\mathbf{X}^T \mathbf{y} - \lambda < 0$, this will result in a contradiction, and thereby, $\beta = 0$.
- If $\beta < 0$, then $\frac{\partial L(\beta)}{\partial \beta} = 2\beta - 2\mathbf{X}^T \mathbf{y} - \lambda$. Similarly as above, we can conclude that $\beta = \frac{(2\mathbf{X}^T \mathbf{y} + \lambda)}{2}$. But if $2\mathbf{X}^T \mathbf{y} + \lambda > 0$, this will result in a contradiction, and thereby, $\beta = 0$.
- If $\beta = 0$, then we have had the solution and no longer need the calculate the gradient.

In summary, we can derive the solution of β as

$$\hat{\beta} = \begin{cases} \frac{(2\mathbf{X}^T \mathbf{y} - \lambda)}{2}, & \text{if } 2\mathbf{X}^T \mathbf{y} - \lambda > 0 \\ \frac{(2\mathbf{X}^T \mathbf{y} + \lambda)}{2}, & \text{if } 2\mathbf{X}^T \mathbf{y} + \lambda < 0 \\ 0, & \text{if } \lambda \geq |2\mathbf{X}^T \mathbf{y}| \end{cases}$$

Now we are ready to generalize this practice to general case with more predictors.

The spirit is to keep as much the easiness of solving for one predictor as we derived above as possible. Thus, revealing the resemblance of the general problem with our one-predictor special problem is important. Particularly, we decide to follow an iterative structure that updates each β_j at a time when fixing all the other parameters as their latest values. Thus, suppose that we are now at the t th iteration and we are trying to optimize for β_j , we can rewrite the general optimization problem's objective function as a function of β_j

$$L(\beta_j) = \left\| \mathbf{y} - \sum_{k \neq j} \mathbf{X}_{(:,k)} \beta_k^{(t-1)} - \mathbf{X}_{(:,j)} \beta_j \right\|_2^2 + \lambda \sum_{k \neq j} |\beta_k^{(t-1)}| + \lambda |\beta_j|.$$

Here, $\beta_k^{(t)}$ is the value of β_k in the t th iteration. The objective function above can be simplified as

$$L(\beta_j) = \|\mathbf{y} - \mathbf{X}_{(:,j)}\beta_j\|_2^2 + \lambda|\beta_j|,$$

which just resembles the structure as the one-predictor special case we discussed. Thus, we can readily derive that

$$\hat{\beta}_j^{(t)} = \begin{cases} q_j - \lambda/2, & \text{if } q_j - \lambda/2 > 0 \\ q_j + \lambda/2, & \text{if } q_j + \lambda/2 < 0, \\ 0, & \text{if } \lambda \geq |2q_j| \end{cases}$$

$$\text{where } q_j = \mathbf{X}_{(:,j)}^T (\mathbf{y} - \sum_{k \neq j} \mathbf{X}_{(:,k)} \beta_k^{(t-1)}).$$

An Example to implement the Shooting algorithm: Here let's consider one exemplary data as shown in below.

Table 8.1: A dataset example for LASSO

X_1	X_2	Y
-0.707	0	-0.77
0	0.707	-0.33
0.707	-0.707	0.62

The dataset of Y is actually randomly sampled from the true model,

$$Y = 0.8X_1 + \varepsilon, \text{ where } \varepsilon \sim N(0, 0.5).$$

Thus, it can be seen that the variable X_2 is irrelevant.

Now let's implement the Shooting algorithm for LASSO on this data. The objective function of LASSO on this case is

$$\sum_{n=1}^N [y_n - (\beta_1 x_{n,1} + \beta_2 x_{n,2})]^2 + \lambda(|\beta_1| + |\beta_2|).$$

Note that, here, for simplicity, we don't need to include the offset parameter β_0 in the model as the predictors are standardized with mean as zero.

Suppose that we choose $\lambda = 0.88$. First, we initiate the regression parameters as $\hat{\beta}_1^{(0)} = 0$ and $\hat{\beta}_2^{(0)} = 0$.

In the first iteration, we aim to update $\hat{\beta}_1$. We can obtain that

$$\mathbf{y} - \mathbf{X}_{(:,2)}\hat{\beta}_2^{(0)} = \begin{bmatrix} -0.71 \\ -1.037 \\ 1.327 \end{bmatrix}.$$

Thus,

$$q_1 = \mathbf{X}_{(:,1)}^T (\mathbf{y} - \mathbf{X}_{(:,2)}\hat{\beta}_2^{(0)}) = 1.44.$$

As

$$q_1 - \lambda/2 = 1 > 0,$$

we know that

$$\hat{\beta}_1^{(1)} = q_1 - \lambda/2 = 1.$$

Similarly, we can update $\hat{\beta}_2$. We can obtain that

$$\mathbf{y} - \mathbf{X}_{(:,1)}\hat{\beta}_1^{(0)} = \begin{bmatrix} -1.477 \\ -0.33 \\ -0.087 \end{bmatrix}.$$

Thus,

$$q_2 = \mathbf{X}_{(:,1)}^T (\mathbf{y} - \mathbf{X}_{(:,1)}\hat{\beta}_1^{(0)}) = -0.178.$$

As

$$\lambda \geq |2q_2|,$$

we know that

$$\hat{\beta}_2^{(1)} = 0.$$

Thus, on this simple example, with only one iteration, the LASSO method can identify the irrelevant variable and delete it from the model.

II.3 R Lab

In what follows, we apply LASSO on an extended AD dataset that has 329 variables. It includes 313 variables that are derived from the MRI images of the subjects, corresponding to the grey matter volumes of 313 brain regions. We have known that many of these variables are correlated with each other as our prior knowledge. Also, depending on the outcome to predict, not all the brain regions are useful. Thus, this extended AD dataset provides a good example for us to showcase the use of LASSO and PCA.

First, let's load the data into the R workspace:

```
# Chapter 8 Dataset of Alzheimer's Disease Objective: prediction
# of
# diagnosis filename
AD <- read.csv("AD_hd.csv", header = TRUE)
```

This time, let's formulate an interesting prediction question: can we use the MRI readings to predict the age of the subject. Using the following R code, we may get a sense of the relationship of the variables by drawing the scatterplots of variables that correlate with the outcome variable “AGE” most strongly according to the Pearson correlation.

```
# Supplement the model with some visualization of the statistical
# patterns
# Scatterplot matrix to visualize the relationship between outcome
# variable
# with continuous predictors
require(ggplot2)
# install.packages('GGally')
require(GGally)
# draw the scatterplots and also empirical shapes of the distributions of
# the variables
tempRank <- sort(abs(cor(AD[, 5], AD[, 17:329])), decreasing = TRUE,
index.return = TRUE)
p <- ggpairs(AD[, c(5, 16 + tempRank$ix[1:8])], upper = list(
continuous = "points"),
lower = list(continuous = "cor"))
print(p)
```

Then we can see that, the correlations between the MRI variables with **AGE** are significant, and the correlations among the MRI variables are also strong, which confirms with our prior knowledge and also indicates significant redundancy in the variables.

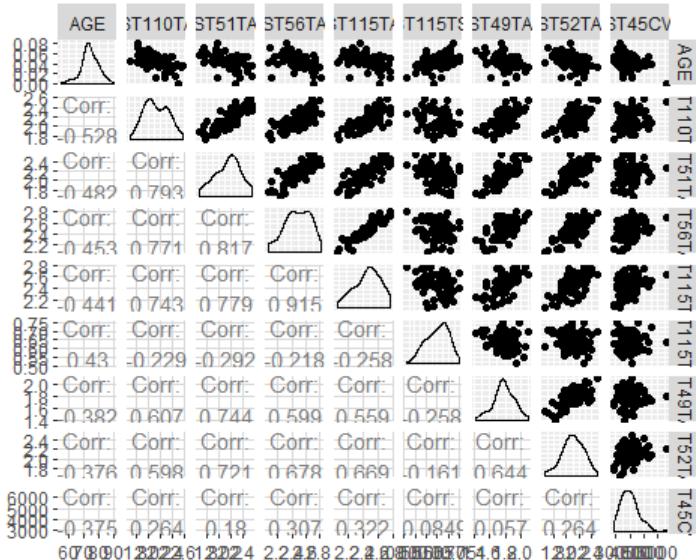


Figure 8.2: Scatterplots of some MRI variables

Now let's split the data into training and testing datasets.

```
AD[, 17:dim(AD)[2]] <- scale(AD[, 17:dim(AD)[2]])
# Use the glmnet R pacakge to build LASSO model split into training and test
# sets
AD$train <- ifelse(runif(nrow(AD)) < 0.8, 1, 0)
# separate training and test sets
trainset <- AD[AD$train == 1, -grep("train", names(AD))]
testset <- AD[AD$train == 0, -grep("train", names(AD))]
trainX <- as.matrix(trainset[, 17:dim(trainset)[2]])
testX <- as.matrix(testset[, 17:dim(testset)[2]])
trainY <- as.matrix(trainset[, 5])
testY <- as.matrix(testset[, 5])
```

The `glmnet` package can be used to implement the LASSO method:

```
# build model
install.packages('glmnet')
require(glmnet)
fit = glmnet(trainX, trainY, nlambda = 100)
```

We can use the `plot()` function to see the path solution trajectory of the regression coefficients by LASSO for different values of lambda, as shown in Figure 8.3.

```
plot(fit, label = TRUE)
```

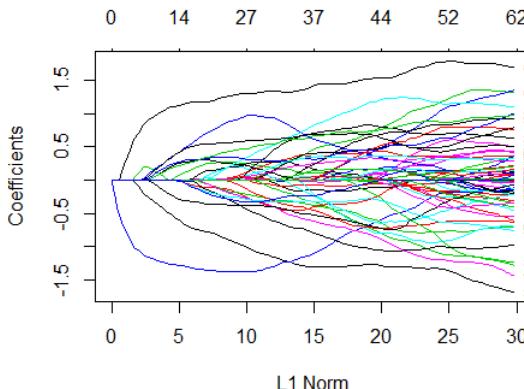


Figure 8.3: Path solution trajectory of the regression coefficients by LASSO

The numerical details of the trained LASSO regression models can also be seen by `print(fit)`. Here, we skip the output due to space limit. As we have seen, the `glmnet` trained not only one LASSO model, but actually 100 models by default. We could use `coef()` to query details of each of the models. For example, `coef(fit, s = 0.05)` queries the model that has `lambda = 0.05`.

We draw the histogram of the Pearson correlations of the selected variables in this model, which is shown in Figure 8.4.

```
# Check out the marginal correlations between the selected variables with  
# the outcome  
idx.var <- which(coef(fit, s = 0.05) != 0) - 1  
tempData <- as.numeric(abs(cor(trainY, trainX[, idx.var])))  
qplot(tempData, geom = "histogram")
```

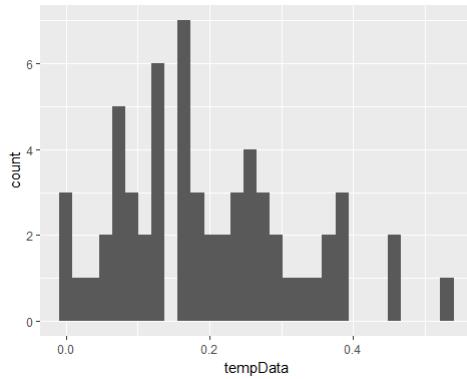


Figure 8.4: Histogram of the Pearson correlations of the selected variables with **AGE**

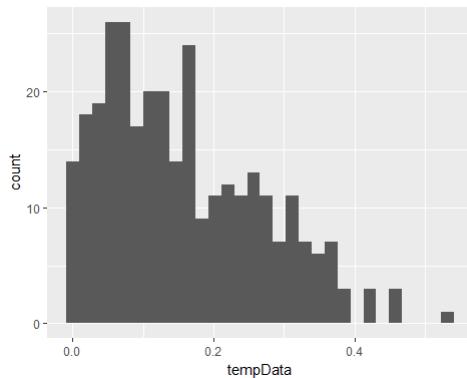


Figure 8.5: Histogram of the Pearson correlations of all variables with **AGE**

We can see that, the LASSO model is a multivariate method that selects variables not only based on their marginal correlations with the outcome, but also their syngeneic effects. Figure 8.5 shows the histogram of the Pearson correlations of all variables for a contrast.

```
# Compare with the overview of the correlations between variables
# with the
# outcome
tempData <- as.numeric(abs(cor(trainY, trainX)))
qplot(tempData, geom = "histogram")
```

The `predict()` function can be used to predict on the testing dataset using different models.

```
# Predict on the testing data
predict(fit, newx = testX, s = c(0.1, 0.2, 0.4))

##           1         2         3
## 3  81.24935 81.23730 78.97272
## 9  69.19876 69.64454 70.43511
## 17 69.89315 68.66627 67.34304
## 19 72.01627 70.19499 71.69000
## 29 73.75759 71.94729 71.05652
## 30 67.57181 67.22865 67.62802
## 31 69.72086 69.79635 71.41110
## 38 70.65147 72.04312 73.13276
## 39 84.74255 84.62814 84.28301
## 45 71.88253 71.29639 69.67876
## 48 74.83788 74.96710 72.61464
## 52 66.70134 69.83361 71.03180
## 53 72.42933 71.19085 72.60041
## 61 73.11982 73.33213 75.97945
## 62 75.87737 74.92556 75.84209
## 73 74.98227 75.70788 74.91021
```

On the other hand, to decide on the best model, `glmnet` implements 10-fold cross-validation procedure using `cv.glmnet()`.

```
# Use cross-validation to decide which model is best
cv.fit = cv.glmnet(trainX, trainY)
plot(cv.fit)
```

The result is shown in Figure 8.6, which reveals a certain optimal point on which we can decide the best model. It is also noticeable that the

optimality of the model is not very statistically significant, probably due to the small sample size, or enormous heterogeneity of the AD population, or other data issues.

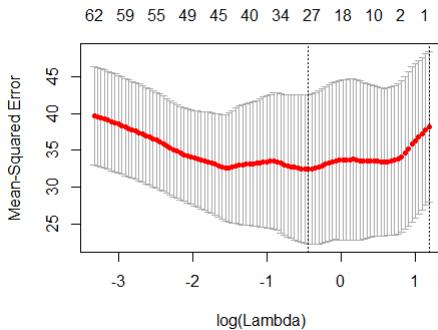


Figure 8.6: MSEs by cross-validation with different values of `lambda`

The best model can be queried by calling on the function `cv.glmnet` as shown in below (results omitted due to the space limit):

```
# To view the selected variables and the corresponding coefficients
cv.fit$lambda.min

## [1] 0.6437308

coef(cv.fit, s = "lambda.min")
```

As LASSO has picked up the variables, we can further fit a regression model using these variables. Here, the reason that we need to re-fit the regression model is that, since LASSO uses the L_1 norm to push those insignificant predictors out of the model, it also results in shrinkage of the magnitudes of the predictors that are significant. This shrinkage is bias that makes the model less accurate in prediction. Usually, people tend to use LASSO for model selection only, e.g., to identify the predictors that are

significant. Then, with the identified predictors, a classic regression model is further applied on this reduced set of predictors to build the final model.

We follow this two-step strategy. As shown in below, it can be seen that these variables could lead to a model that has the R-squared as large as **0.9117**. It is also possible, from the results shown in below, that the model can be further simplified by deleting more insignificant variables.

```
# fit a Linear regression model
trainX.reduced <- data.frame(trainX[, which(coef(cv.fit, s = "lambda.min") != 0) - 1])
tempData <- cbind(trainY, trainX.reduced)
lm.AD <- lm(trainY ~ ., data = tempData)
summary(lm.AD)

##
## Call:
## lm(formula = trainY ~ ., data = tempData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -5.4283 -1.4383  0.4106  1.0739  3.3860 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 73.48975   0.42384 173.389 < 2e-16 ***
## ST103CV     0.68829   0.86621   0.795  0.43404    
## ST103TA     0.82101   0.86085   0.954  0.34901    
## ST106TA    -1.73112   0.63784  -2.714  0.01164 *  
## ST106TS    -0.34535   0.60115  -0.574  0.57057    
## ST110CV    -0.99903   0.75223  -1.328  0.19569    
## ST110TA    -0.79230   0.64168  -1.235  0.22797    
## ST113SA     0.06535   0.62399   0.105  0.91740    
## ST118SA    -0.59835   0.69504  -0.861  0.39717    
## ST119CV    -1.05435   0.83192  -1.267  0.21626    
## ST119SA    -0.77869   0.77768  -1.001  0.32591    
## ST128SV     1.63101   0.69748   2.338  0.02733 *  
## ST129TS     0.98078   0.66285   1.480  0.15098    
## ST130TS    -0.11623   0.58658  -0.198  0.84446    
## ST14TS     -1.04771   0.55252  -1.896  0.06909 .  
## ST16SV      0.34340   0.72793   0.472  0.64104    
## ST17SV     -1.14047   0.57236  -1.993  0.05690 .  
## ST26TS    -0.47550   0.56765  -0.838  0.40985    
## ST35TS    -1.38430   0.53229  -2.601  0.01515 *
```

```

## ST39TS      1.34252   0.46659   2.877  0.00791 ** 
## ST42SV      0.96528   0.56439   1.710  0.09912 .  
## ST44TS      0.61458   0.58486   1.051  0.30301 
## ST45CV     -0.59449   0.55985   -1.062  0.29806 
## ST59CV      2.02101   0.70023   2.886  0.00774 ** 
## ST62TS     -0.54468   0.46893   -1.162  0.25597 
## ST74TS      0.10354   0.60269   0.172  0.86493 
## ST7SV       0.81011   0.54650   1.482  0.15027 
## ST83CV      0.80073   0.51498   1.555  0.13207 
## ST83TA     -0.20721   0.79645   -0.260  0.79678 
## ST85TS      0.84103   0.68633   1.225  0.23141 
## ST98CV     -0.03397   0.71845   -0.047  0.96265 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 2.707 on 26 degrees of freedom 
## Multiple R-squared:  0.9117, Adjusted R-squared:  0.8099 
## F-statistic:  8.95 on 30 and 26 DF,  p-value: 1.203e-07

```

Note that, the `glmnet` package not only implements LASSO, but also other related models such as Ridge regression. For instance, via the R code below we can implement the Ridge regression by setting `alpha = 0`:

```

# Do a ridge regression instead
fit.ridge = glmnet(trainX, trainY, alpha = 0, nlambda = 100)
print(fit.ridge)

```

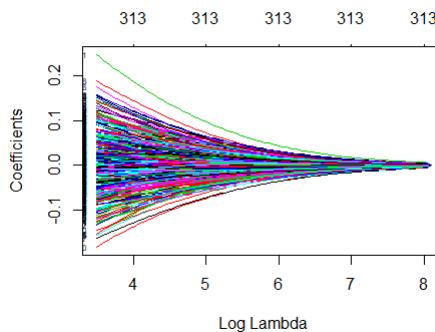


Figure 8.7: Path solution trajectory of the regression coefficients by Ridge regression

As shown in Figure 8.7, the path solution trajectory of the regression coefficients by Ridge regression has quite a different shape from the path solution trajectory of LASSO regression shown in Figure 8.3.

```
plot(fit.ridge, xvar = "lambda", label = TRUE)
```

We can also implement the idea of LASSO on logistic regression model. Here, we use the “`DX_b1`” as our outcome variable that has two classes, “`NC`” and “`LMCI`”, denoting for the normal aging and mild cognitive impairment, respectively. Note that, classifying between `NC` and `LMCI` is very challenging, much more challenging than the classification between `NC` and `AD` that has been discussed in previous chapters, since the `LMCIs` are to certain degrees still normal individuals and are not clinically diagnosed with dementia yet.

```
# Fit a LASSO model for Logistic regression
trainY <- as.matrix(trainset[, 2])
testY <- as.matrix(testset[, 2])
fit = glmnet(trainX, trainY, nlambda = 100, family = "binomial")
plot(fit, label = TRUE)
```

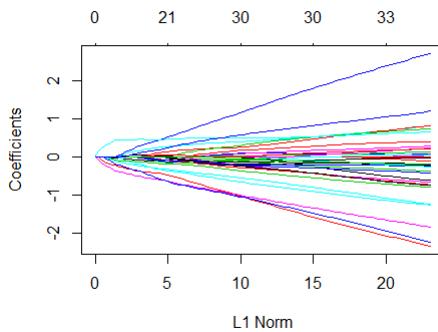


Figure 8.8: Path solution trajectory of the coefficients by logistic LASSO

Also, the cross-validation can be used to decide on the best model:

```
# Use cross-validation to decide which model is best
cv.fit = cv.glmnet(trainX, trainY, family = "binomial", type.measure = "class")
plot(cv.fit)
```

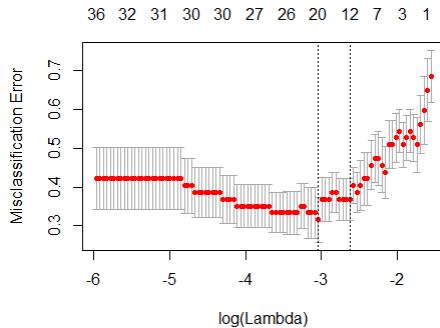


Figure 8.9: Classification errors by cross-validation with different values of lambda

As shown in Figure 8.9, a certain optimal point can be identified to decide on the best model. We can further output the corresponding coefficients of this optimal model via the R code below (results omitted due to space limit).

```
# To view the selected variables and the corresponding coefficients
coef(cv.fit, s = "lambda.min")
```

Similarly in LASSO, we could use `predict()` to predict on the testing dataset using any model.

```
predict(cv.fit, newx = testX, s = "lambda.min")

##                      1
## 3   -1.23155042
## 9   -0.32381239
## 17  -5.49829437
## 19  -0.47832582
## 29  -0.35677823
## 30  -0.82189141
## 31   0.62296595
## 38   0.52778295
## 39  -1.49146275
## 45  -2.54049855
## 48   1.07123484
## 52  -1.27364645
```

```
## 53  1.46493378
## 61 -1.21799365
## 62 -3.32616327
## 73  0.03668668
```

II.4 Remark

The shooting algorithm¹ has been widely used in many extension models of LASSO in the statistics community. The shooting algorithm is easy to use and has nice interpretation of each iteration. But it could be slow in very high-dimensional situations. Also, with more complex penalty terms such as those L₂₁-norm regularization or group regularization terms, the shooting algorithm may not work anymore. In machine learning community where the computational efficiency is of particular interest, many scalable algorithms such as the projection operator based methods have been developed. Interested readers can read more of these works² in this direction that provided closed form iterative updating rules by projection operator on a variety of regularization terms.

On the other hand, regarding why LASSO can produce sparse estimates of the regression parameters while Ridge regression could not, a deep reason was revealed in the “bible” book of statistical learning³. Here, we adopt an easy and more common sense explanation (which has also been a very famous example) as shown in the Figure 8.10. It shows the application of LASSO and Ridge regression models on a problem with 2 predictors. The contour plot corresponds to the least squares loss function which is shared by classic regression model, LASSO, and Ridge models. $\hat{\beta}$ in the

¹ Fu, WJ. Penalized regressions: the bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 1998.

² <http://www.yelab.net/software/SLEP/>

³ Hastie, T., Tibshirani R. and Friedman, J. *The elements of statistical learning*, 2nd edition. Springer, 2009.

center of the contour is the least squares estimator of the regression parameters.

Figure 8.10 shows that why LASSO can generate sparse estimation of the model. As the objective function of LASSO consists of two terms, the optimal solution lies on the intersection of the geometric areas corresponding to the two terms. As LASSO uses L_1 norm, it results in those “sharp” corner points that mostly like to be the point of contacts of the two geometric areas. Those point of contacts are themselves sparse solutions, e.g., in Figure 8.10, the point of contact implies that $\beta_1 = 0$.

As a comparison, in Ridge regression, as the geometric area corresponding to the L_2 norm has no such “sharp” corner points, the model has no strong incentive for where to allocate the point of contacts of the two geometric areas. Thus, given the infinite number of potential point of contacts of the two geometric areas, it is expected that Ridge regression will not result in sparse solutions with exact zeros in $\hat{\beta}$.

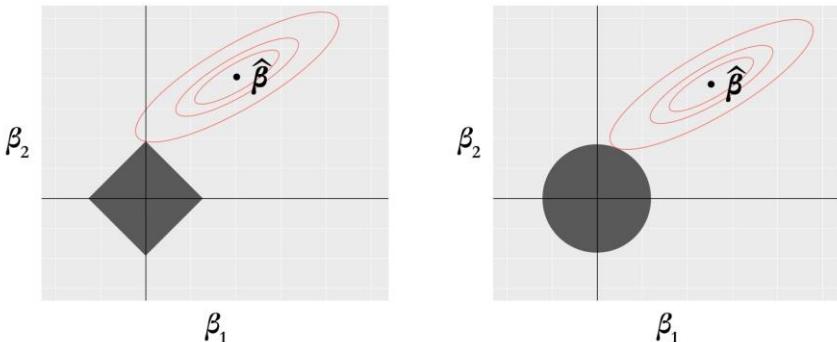


Figure 8.10: Why LASSO could generate sparse estimates while Ridge tends to not

Following this idea, the L_1 norm is later extended to L_q norm for $q \leq 1$. For any $q \leq 1$, we could generate those “sharp” corner points to enable sparse solutions of $\hat{\beta}$. The advantage of using $q < 1$ is to reduce bias in the model. Recall that, we have mentioned earlier, that LASSO will lead to bias in parameter estimation. Using $q < 1$ is a good approach to reduce this bias, while it still produces “sharp” corner points but penalizes less on the significant predictors. The cost of using $q < 1$ is that it will result in nonconcave penalty terms, making the overall objective function of the sparse model nonconcave.

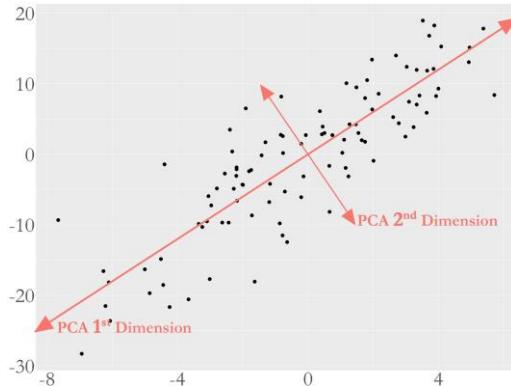


Figure 8.11: Illustration of the principal components in a dataset with 2 variables; the main variation source is represented by the 1st PC dimension

III. Principal Component Analysis

III.1 Rationale and Formulation

The PCA method is built on the assumption that, for a multivariate dataset that has many variables, the dimensionality of the dataset is smaller than it appears to be. In other words, for example, for one dataset that has 10 variables, we may have the impression that there are ten independent sources of variation that infuse uncertainty into the data. But, PCA is built

on the idea that the underlying independent sources of variations are only a few (e.g., 2 or 3 for 10 variables could be usual). The question is how to identify the intrinsic sources of the variation.

As shown in Figure 8.11, PCA pursues this idea of identifying the intrinsic sources of the variation in the framework of linear models. The characteristic shape of the dataset shown in Figure 8.11 indicates that, although the data points are located in a two-dimensional space, the data points are not totally randomly scattered all over the place. Rather, there is a force that orients these data points towards one direction (or, in the same effect, you may say there is a force that pushes the data points towards one narrow zone). To recover these forces, the PCA seeks linear combinations of the original variables to pinpoint the directions towards which the underlying forces are pushing the data points. In other words, another assumption of PCA is that the relationship between the underlying dimensions and the variables (surface dimensions) is linear.

III.2 Theory and Method

The idea shown in Figure 8.11 reveals the principle to guide the estimation of the linear weights to combine the variables. This leads to the following formulation:

$$\mathbf{w}_{(1)} = \arg \max_{\mathbf{w}_{(1)}^T \mathbf{w}_{(1)} = 1} \left\{ \sum_{n=1}^N \mathbf{x}_{(n)} \cdot \mathbf{w}_{(1)} \right\},$$

where there are N samples and p variables, $\mathbf{x}_{(n)} \in R^{1 \times p}$ is the n th sample, and $\mathbf{w}_{(1)} \in R^{p \times 1}$ is the linear weights vector of the first PC. Note that the constraint $\mathbf{w}_{(1)}^T \mathbf{w}_{(1)} = 1$ is to control the scale of the vector – without which an infinite number of solutions would exist. This also indicates that the absolute magnitudes of the weights are meaningless. Only the relative magnitudes are useful.

A more succinct form could be:

$$\mathbf{w}_{(1)} = \arg \max_{\mathbf{w}_{(1)}^T \mathbf{w}_{(1)} = 1} \left\{ \mathbf{w}_{(1)}^T \mathbf{X}^T \mathbf{X} \mathbf{w}_{(1)} \right\},$$

where $\mathbf{X} \in R^{N \times p}$ is usually called the data matrix that concatenate all the N samples into a matrix. $\mathbf{X}^T \mathbf{X}$ is actually the sample covariance matrix.

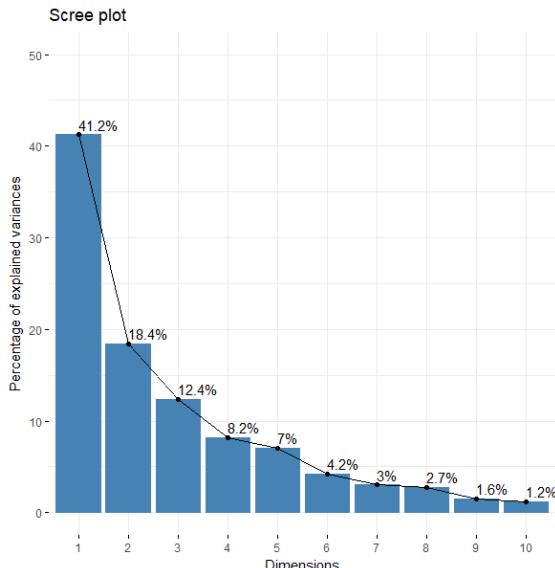


Figure 8.12: Scree plot that shows the first 5 PCs maybe significant, while the first PC is definitely a major variation source, together with the second and third PCs as other main variations sources

To identify the second PC, we could follow the principle of iteration. The idea is rather simple. As the first PC represents one variance source, and the original data \mathbf{X} contains a linear aggregation of multiple variance sources, why not remove the first variance source from \mathbf{X} , then create a new data that contains the remaining variance sources? Then, the procedure for finding $\mathbf{w}_{(1)}$ could be readily used for finding $\mathbf{w}_{(2)}$, since with $\mathbf{w}_{(1)}$ removed, $\mathbf{w}_{(2)}$ is the largest variance source now.

This process could be generalized as:

- In order to find the k th PC, we could create a dataset as $\mathbf{X}_{(k)} = \mathbf{X} - \sum_{s=1}^{k-1} \mathbf{X} \mathbf{w}_{(s)} \mathbf{w}_{(s)}^T$.
- Then, we solve $\mathbf{w}_{(k)} = \arg \max_{\mathbf{w}_{(k)}^T \mathbf{w}_{(k)} = 1} \{ \mathbf{w}_{(k)}^T \mathbf{X}_{(k)}^T \mathbf{X}_{(k)} \mathbf{w}_{(k)} \}$ for identifying $\mathbf{w}_{(k)}$.

In practice, we need to decide how many PCs are needed to represent the dataset. In theory, for a dataset with p variables, there are p PCs that could be extracted if the dataset has more sample size than the number of variables. But it is often the case that only the first few PCs are needed since these few PCs could explain away majority of the variation in the data. The scree plot as shown in Figure 8.12 is a common tool in practice, that draws the eigenvalues of the PCs to look for the change point beyond which the PCs maybe statistically insignificant.

We could use the following example to practice this procedure. The dataset is shown in the Table below:

Table 8.2: A dataset example for PCA

X_1	X_2
-1	0
3	3
3	5
-3	-2
3	4
5	6
7	6
2	2

First, we can calculate the sample covariance matrix as

$$\mathbf{S} = \mathbf{X}^T \mathbf{X} = \begin{bmatrix} 115 & 118 \\ 118 & 130 \end{bmatrix}.$$

We can obtain the $\mathbf{w}_{(1)}$ by

$$\mathbf{w}_{(1)} = \arg \max_{\mathbf{w}_{(1)}^T \mathbf{w}_{(1)} = 1} \{\mathbf{w}_{(1)}^T \mathbf{S} \mathbf{w}_{(1)}\}.$$

The lagrangian form is

$$\mathbf{w}_{(1)}^T \mathbf{S} \mathbf{w}_{(1)} - \lambda_1 \mathbf{w}_{(1)}^T \mathbf{w}_{(1)}.$$

By taking the derivative of the lagrangian form with regards to $\mathbf{w}_{(1)}$, it is not hard to arrive at the equation:

$$\mathbf{S} \mathbf{w}_{(1)} - \lambda_1 \mathbf{w}_{(1)} = 0.$$

Thus, this is an eigenvalue problem of the matrix \mathbf{S} . We can solve it as $\lambda_1 = 240.74$ and $\mathbf{w}_{(1)} = [0.68, 0.73]$. Further, we can get that $\lambda_2 = 4.26$ and $\mathbf{w}_{(2)} = [-0.73, 0.68]$.

III.3 R Lab

We apply PCA on the AD data that has been used in the R lab of LASSO. There are many packages in R that can conduct PCA. Here, we use the function `PCA()` in the “**FactoMineR**” package.

```
# Implement principal component analysis on the AD data
# install.packages('factoextra')
require(factoextra)
require(FactoMineR)
require(ggfortify)
tempData <- AD[, c(17:dim(AD)[2])]
# Conduct the PCA analysis
pca.AD <- PCA(tempData, graph = FALSE, ncp = 10)
```

The construct `pca.AD` has the eigenvalues and the principal components. We can use `fviz_screenplot()` to visualize the contributions of the principal components, as shown in Figure 8.13.

```
# Examine the contributions of the PCs in explaining the variation in data
fviz_screenplot(pca.AD, addlabels = TRUE, ylim = c(0, 50))
```

It can be seen from Figure 8.13 that the first PC could explain away 16.7% of the total variation and the second PC could explain away 12.7% of the total variation. It seems that there is a change point at the third PC, showing that the following PCs *could* be insignificant.

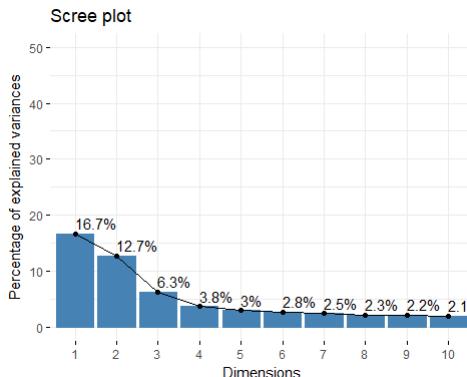


Figure 8.13: Scree plot of the PCA analysis on the AD dataset

We could also show the numerical details of the loadings of the variables in the PCs.

```
# Examine the Loadings of the variables in the PCs
var <- get_pca_var(pca.AD)
head(var$contrib)

##           Dim.1      Dim.2      Dim.3      Dim.4      Dim.5
## ST101SV 5.217488e-01 0.00543192 0.022513018 0.00695649 0.55635835
## ST102CV 6.331461e-01 0.26061413 0.167089523 0.02972375 0.11789451
## ST102SA 1.029105e+00 0.00198550 0.011535559 0.22360153 0.17723377
## ST102TA 1.080058e-02 1.06804755 0.237858049 0.38424318 0.00483804
## ST102TS 5.979293e-05 0.06038710 0.299593782 0.52436240 0.02805072
## ST103CV 8.089453e-02 0.05503171 0.001267605 0.54330434 3.08447026
##           Dim.6      Dim.7      Dim.8      Dim.9      Dim.10
## ST101SV 1.835299776 0.498169844 0.055960987 0.018902630 0.0517726064
## ST102CV 0.037225860 0.182728808 0.039477823 0.029123146 0.1602108140
```

```

## ST102SA 0.336057060 0.006782785 0.008738713 0.009108956 0.0002884433
## ST102TA 0.167160010 0.221663964 0.273441045 0.003510227 0.2088207321
## ST102TS 0.008515869 0.167631015 0.074972768 0.022843632 1.9489628313
## ST103CV 0.175682030 0.822144415 1.592278638 0.251205409 0.1727957251

```

It is also helpful to visualize the contributions of the variables to the PCs by figures. For example, Figures 8.14 and 8.15 show the contributions of the variables to the first and second PC, respectively.

```

fviz_contrib(pca.AD, choice = "var", axes = 1, top = 20)
fviz_contrib(pca.AD, choice = "var", axes = 2, top = 20)

```

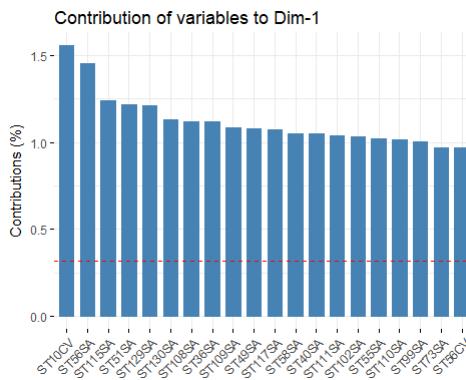


Figure 8.14: Contributions of the variables for the first PC

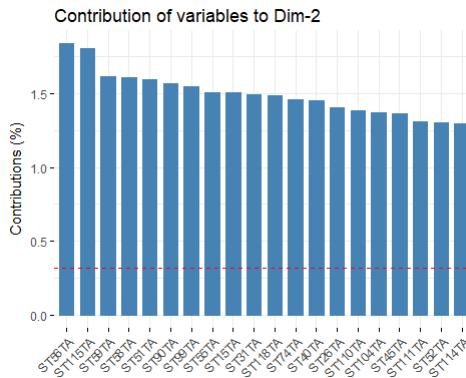


Figure 8.15: Contributions of the variables for the second PC

Figure 8.16 is another way to visualize the contributions of the variables to the PCs.

```
fviz_pca_var(pca.AD, col.var = "contrib", select.var = list(contrib
ib = 20), gradient.cols = c("#00AFBB",
 "#E7B800", "#FC4E07"), repel = TRUE # Avoid text overlapping
)
```

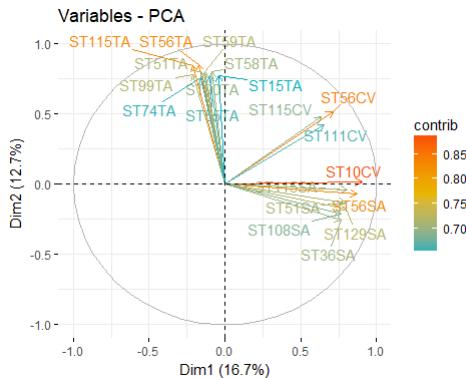


Figure 8.16: Loadings of the top variables in the first and second PCs

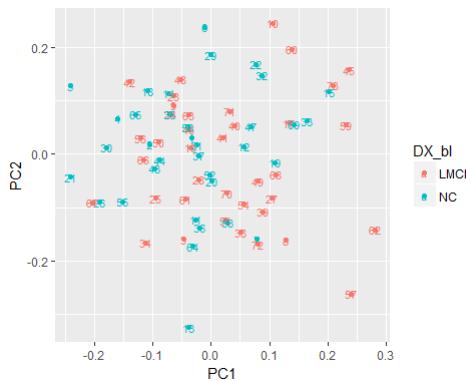


Figure 8.17: Scatterplot of the subjects in the space defined by the first and second PCs

With the identified PCs, we can visualize the distribution of the data points in this new space spanned by the PCs. Sometimes, it may reveal some inherent structure of the dataset. For example, for a classification problem, it is hoped that the data points from different classes would cluster around different centers in the space spanned by the PCs. In our case, as shown in Figure 8.17, this cluster structure is not perfect but seems to be on the borderline of significance, as it is not entirely like a pure random pattern.

```
# Examine the projection of data points in the new space defined  
# by PCs  
autoplot(prcomp(tempData), data = AD, colour = "DX_b1", label = T  
RUE, label.size = 3)
```

The PCs can be taken as new variables. For example, we can build regression models using the PCs to predict outcome variables. Here, we use AGE as the outcome, and first fit a regression model with 10 PCs.

```
# fit a regression model using the PCs  
tempData <- data.frame(cbind(AD[, 5], pca.AD$ind$coord))  
names(tempData) <- c("AGE", "PC1", "PC2", "PC3", "PC4", "PC5", "P  
C6", "PC7",  
"PC8", "PC9", "PC10")  
lm.AD <- lm(AGE ~ ., data = tempData)  
summary(lm.AD)
```

It seems that the first PC is not significant, while the second, the third, and the fifth PCs are significant. It is not unusual to see that the first PC is insignificant, as the first PC sometimes may embody a variation source that is not correlated with the outcome variable. It is always a challenge to interpret the results of PCA, particularly, to interpret the physical correspondence of the PCs. On the other hand, we can see that the R-squared by this model is 0.3672, and the p-value is as small as 0.0008235, indicating the overall model is significant. Further variable selection would be conducted to prune the model and reduce its complexity.

```

## 
## Call:
## lm(formula = AGE ~ ., data = tempData)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.3377  -2.5627   0.0518   2.6820  11.1772
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 73.68767  0.59939 122.938 < 2e-16 ***
## PC1         0.04011  0.08275  0.485  0.629580    
## PC2        -0.31556  0.09490 -3.325  0.001488 **  
## PC3         0.50022  0.13510  3.702  0.000456 *** 
## PC4         0.14812  0.17462  0.848  0.399578    
## PC5         0.47954  0.19404  2.471  0.016219 *   
## PC6        -0.29760  0.20134 -1.478  0.144444    
## PC7         0.10160  0.21388  0.475  0.636440    
## PC8        -0.25015  0.22527 -1.110  0.271100    
## PC9        -0.02837  0.22932 -0.124  0.901949    
## PC10        0.16326  0.23282  0.701  0.485794    
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.121 on 62 degrees of freedom
## Multiple R-squared:  0.3672, Adjusted R-squared:  0.2651 
## F-statistic: 3.598 on 10 and 62 DF,  p-value: 0.0008235

```

III.4 Remark

While PCA has been widely used, it is often criticized as a black box model or lack interpretability since it is always not easy to connect the identified principal components with physical entities. But, probably, we may stop worry about validity of the PCA method in many applications and focus on the significance it can reveal. After all, when studying real-world systems that we haven't known what we don't know yet, we have to make bold hypothesis and make the leap over the gaps. This is probably one of the reasons why PCA has been applied in many real world applications. The massive practices of PCA in many areas have formed a convention, or a myth – some critical statisticians may say – that formulistic rubrics have been invented to help beginners to quickly jump in to the vehicle of PCA

and start to convert their very challenging data into PCA patterns, then further convert these patterns into formulated sentences such as “the variables that have larger magnitudes in the first 3 PCs correspond to the brain regions in hippocampus areas, indicating that these brain regions manifest significant functional connectivity to deliver the verbal function”, or “we have identified 5 significant PCs, and the genes that show dominant magnitudes in the linear weights vector are all related to T-cell production and immune functions – thereby each of the PC indicates a biological pathway that consists of these constitutional genes working together to produce specific types of proteins”. You may also hear from some financial analysts who presented such a result: “through PCA on 100 stocks, we found that the first PC consists of 10 stocks as their weights are significantly larger than the other stocks. This may indicate that there is strong correlation between these 10 stocks and you may consider this fact when you define your investment strategy”.

IV. Variable Selection by Random Forests

III.1 Rationale and Formulation

As we have seen that, both LASSO and PCA are linear models, which are not suitable if there are nonlinear relationships in the dataset. For nonlinear variable selection, the random forests have been commonly used. Recall that the random forests consist of decision nodes that are defined by splits on some variables. This can provide information about the variables’ importance in the random forests. Also, random forests are powerful in capturing nonlinear and predictive information from data, and therefore, provide data-driven characterization of variable importance. Third, random forests require little data preprocessing. They can handle different scales of the continuous variables since the impurity gain is calculated based on the outcome variable, and can work with both categorical and numerical variables. Given these advantages, methods have been developed to conduct variable selection using random forest models.

III.2 Theory and Method

Variable importance scores: The importance score of a variable can be measured based on the Impurity gain. For classification problems, the Gini index for the data points at a node is defined as

$$Gini = \sum_{c=1}^C p_c(1 - p_c),$$

where C is the number the classes in the data set, and p_c is the proportion of the data instances of class c .

Assuming that a variable is used for splitting the node into n children nodes. The Gini gain of the variable can be calculated as

$$\nabla Gini = Gini - \sum_{i=1}^n w_i * Gini_i,$$

where $Gini$ is the Gini index at the node to be split; w_i and $Gini_i$ are the percentage of data instances of the entire dataset and the Gini index at the i^{th} node, respectively.

Then, the importance score of a variable can be calculated as

$$\frac{1}{ntree} \sum_{i=j_1}^{j_m} \nabla Gini_i,$$

where j_1, \dots, j_m are the nodes where the variable j is used for splitting, $\nabla Gini_i$ is the Gini gain at node i , and $ntree$ is the number of trees in the random forest model.

In what follows, we show how this can be done using a small data example.

Consider the following data example shown in Table 8.3. Note that X_1 and X_2 are identical, and thus one of them is redundant.

Table 8.3: A dataset example for RF

ID	X_1	X_2	X_3	X_4	Class
1	1	1	0	1	C0
2	0	0	0	1	C1
3	1	1	1	1	C1
4	0	0	1	1	C1

Assume that a random forest model is built on the data set with two trees, show in Figure 8.18 and Figure 8.19, respectively. The Gini index at each node is also shown in the Figures.

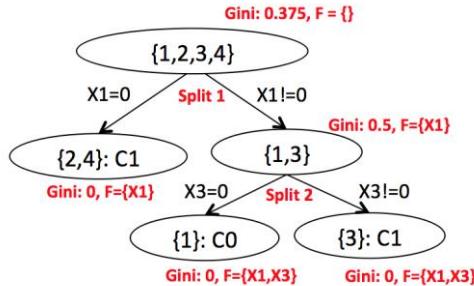


Figure 8.18: Tree 1 in a random forest model

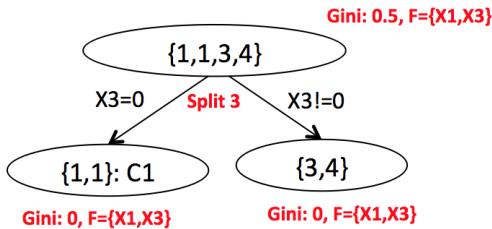


Figure 8.19: Tree 2 in a random forest model

Now, we calculate the importance score for each variable.

At split 1, the Gini gain for X_1 is calculated as

$$0.375 - 0.5*0 - 0.5*0.5 = 0.125.$$

At split 2, the Gini gain for X_3 is 0.5.

At split 3, the Gini gain for X_2 is $0.5 - 0.25*0 + 0.75*0.44 = 0.17$.

At split 4, the Gini gain for X_3 is 0.44.

Therefore, the importance score of X_1 , X_2 , X_3 and X_4 , are $0.125/2 = 0.0625$, $0.17/2=0.085$, $(0.5+0.44)/2=0.47$, and 0 (there is not split using X_4), respectively.

Note that the example above is about classification problems. For regression problems, the mean squared error can be used as the impurity measure:

$$MSE = \sum_i (y_i - \bar{y})^2,$$

where y_i is the value of the outcome variable of the i^{th} data instance at a node, and \bar{y} is the average of the outcome variable of all the data instance at the node.

Regularized random forests: We can see that, in this calculation, the Gini gains are added equally across all the nodes, and a split that perfectly separates 2 data points has the same Gini gain as a split that perfectly separates 100 data points. Thus, the variable importance score built on this Gini gain can be sensitive to noise. In addition, since the importance score of a variable depends on only the splits where the variable is used, the existence of correlation or redundancy between the variables can make this concept of importance score misleading. In the illustrative example abovementioned, since X_1 and X_2 are identical, their importance scores should be the same. However, this is not the case, as in building the trees, the random forest model randomly selects either of them to split the nodes. This results in a dilution effect in the estimation of their importance scores. Thus, the true importance score of either X_1 or X_2 should be the sum of the obtained importance scores from an established random forest model.

This is just a case of two redundant variables. As the number of highly correlated variables increases, the importance scores for each are expected to further decrease.

Thus, the importance scores of the variables provided by the random forests do not consider variables redundancy. In the illustrative example, both X_1 and X_2 are used for splitting nodes and generating impurity gain, but only one of them is needed for the prediction task as they are essentially

the same. To overcome this limitation, we introduce the Regularized random forests (RRF)¹ that can generate a relevant and non-redundant variable subset.

The RRFs are built sequentially. A key difference between RRF and ordinary random forests is that the RRF uses the regularized impurity gain for evaluating the splitting criteria. The regularized impurity gain of variable X_i at a node is calculated as

$$Gain'(X_i) = \begin{cases} \lambda \cdot Gain(X_i) & X_i \notin F \\ Gain(X_i) & X_i \in F \end{cases},$$

where $Gain(X_i)$ is an ordinary impurity gain, e.g., Gini index gain and reduction of mean square error; F is a feature subset including features used to split the previous nodes and is an empty set at the first node of the first tree; $\lambda \in (0,1]$ is referred as the **coefficient** and is used to penalize the gain if X_i is used in previous split.

In RRF, if a variable X_i that is not present in F produces more information gain than the variables in F , it will be used for splitting the node and further added into F . Also, not like in random forests where the number of features M is randomly selected and tested at each node, in RRF, all features from F and a subset of features randomly selected from \bar{F} (features that do not belong to F) are tested. The size of the subset can be set to the minimum of M and size of \bar{F} .

We illustrate how the RRF can be built with the data example shown in Table 8.3. Here, we set $\lambda = 0.8$, and $M = 1$.

First, look at the tree shown in Figure 8.18. At split 1, F is an empty set. Assuming that, still, X_1 is used for testing the split. The regularized Gini gain for $X_1 = 0$ is calculated as

$$0.8 * (0.375 - 0.5*0 - 0.5*0.5) = 0.8 * 0.125 = 0.1.$$

After split 1, $F = \{X_1\}$.

¹ Deng, H. and Runger, G. Gene selection with guided regularized random forest. *Pattern recognition*, 2013.

At split 2, suppose X_3 is selected for testing the split as it is not in F yet. The regularized Gini gain for $X_3 = 0$ is $0.8 * 0.5 = 0.4$. As all the other variables in F should be tested, we can also get that the regularized Gini gain for X_1 is 0. Therefore, X_3 is still the best variable for splitting the node.

After split 3, $F = \{X_1, X_3\}$.

It can be seen that the first tree grown by RRF is the same as the one from the earlier section. Now consider the second example as shown in Figure 8.19.

At split 3, suppose X_2 is still used for testing the node. As X_2 is not in F yet, the regularized Gini gain of $X_2 = 0$ is $0.8 * (0.5 - 0.25*0 + 0.75*0.44) = 0.8 * 0.17 = 0.136$. The regularized Gini gain for $X_1 = 0$ is $0.5 - 0.25*0 + 0.75*0.44 = 0.17$ as X_1 is in F and is not penalized. The regularized Gini gain for $X_3 = 0$ is 0.5.

Therefore, X_3 is used for splitting the node. Both children nodes have only one class and so are made as leaf nodes.

It can be seen from the second tree building process, the redundant feature, X_2 , is penalized and has less gain than its identical variable X_1 . Also, X_3 is now used for splitting the node as it has the strongest impurity gain. Thus, the variable subset selected from the two trees are $F = \{X_1, X_3\}$.

While the RRF is a remedy to overcome redundancy of variables, the guided regularized random forests (GRRF) can further enhance RRF when the sample size is small. This is because that, since a tree recursively splits the training data points, the number of data points can be small when the tree reaching a certain depth. The evaluation criterion may not be accurate when the number of data points is small and could add noise to variable selection. The GRRF can be used to reduce the chance an irrelevant or redundant variable being selected when the number of instances is small.

In GRRF, instead having one λ for all variables, each variable X_i can have its own λ_i :

$$Gain'(X_i) = \begin{cases} \lambda_i \cdot Gain(X_i) & X_i \notin F \\ Gain(X_i) & X_i \in F \end{cases},$$

where λ_i is

$$\lambda_i = (1 - \gamma)\lambda_0 + \gamma * w_i,$$

where λ_0 controls the base regularization, $w_i \in [0,1]$ is a prior of importance of each variable v_i , and $\gamma \in [0,1]$ controls the weight from the prior. Note RRF is a special case of GRRF when $\gamma = 0$. w_i can be determined by prior knowledge about the variables, or can be generated from the normalized importance scores (between 0 and 1) from random forests. The importance scores aggregate the impurities gains from all trees, and therefore, are expected to be less noisy than the impurity gain calculated only from a single node.

Suppose at the first split of the first tree shown in Figure 8.18, two variables X_1 and X_2 are selected for testing, where $w_1 = 0.6$, $w_2 = 1$, $\lambda_0 = 0.9$, and $\gamma = 0.5$. Then, the impurity gain in GRRF for X_1 is calculated as

$$(0.5*0.9 + 0.5*0.6)*0.125.$$

And the impurity gain for X_2 is

$$(0.5*0.9 + 0.5*1)*0.125.$$

Therefore, even the original impurity gain for the two variables is the same, with a prior weight, X_2 is preferred to split the node.

III.3 R Lab

We use the extended AD dataset. Further, we add redundant variables, i.e., the number of variables in this manipulated dataset are 4 times of the number of original variables. We then use all the features to predict the age as a classification problem, i.e., we discretize the variable “AGE” to create a binary variable by its mean value.

First, we apply random forests to this data set. The importance scores of the variables are plotted in Figure 8.20. The variable names are omitted due to limited space.

```
require(inTrees)
require(randomForest)
require(RRF)
set.seed(1)
theme_set(theme_gray(base_size = 18))
```

```

path <- "../../data/AD_hd.csv"
data <- read.csv(path, header = TRUE)
data$AGE <- as.factor(dicreteizeVector(data$AGE, K = 2))
target <- data$AGE
rm_indx <- which(colnames(data) %in% c("AGE", "ID", "TOTAL13", "M
MSCORE"))
X <- data[, -rm_indx]
X1 <- cbind(X, X, X)
colnames(X1) <- paste0("Y", 1:ncol(X1))
for (i in 1:ncol(X1)) {
  perc <- 0.1
  index <- sample(nrow(X1), floor(nrow(X1) * perc))
  X1[, i][sort(index)] <- (X1[, i])[index]
}
X <- cbind(X, X1)
rf <- randomForest(X, target)
imp <- as.data.frame(rf$importance)
colnames(imp)[colnames(imp) == "MeanDecreaseGini"] <- "importance"
imp <- imp[order(imp$importance, decreasing = TRUE), , drop = FALSE]
imp$variable <- rownames(imp)
imp$variable <- factor(imp$variable, levels = as.character(imp$va
riable))
ggplot(data = imp, aes(x = variable, y = importance)) + geom_bar(
  stat = "identity",
  aes(factor(variable)), fill = "red") + theme(axis.text.x = el
ement_blank())

```

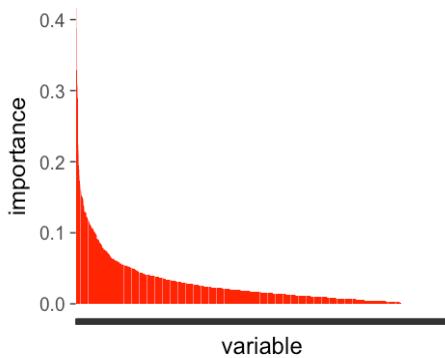


Figure 8.20: The important score of the variables by RF

From Figure 8.20 we can see a ranking of all variables in terms of their predictive powers. The top variables are ST62TA, ST59TS, ST56TA, ST58CV, and ST26TS. However, as we have demonstrated on the exemplary dataset shown in Table 8.3, the important scores of the variables are actually diluted by the redundancy of the variables. Thus, the observation that a large number of variables have non-zero importance scores in Figure 8.20 just amplifies the suspicion that there may be many redundant variables.

Now, let's apply the RRF to the data. The importance scores from the RRF are plotted in Figure 8.21.

```
rrf <- RRF(X, target)
imp <- as.data.frame(rrf$importance)
colnames(imp)[colnames(imp) == "MeanDecreaseGini"] <- "importance"
imp <- imp[order(imp$importance, decreasing = TRUE), , drop = FALSE]
imp$variable <- rownames(imp)
imp$variable <- factor(imp$variable, levels = as.character(imp$variable))
ggplot(data = imp, aes(x = variable, y = importance)) + geom_bar(
  stat = "identity",
  aes(factor(variable)), fill = "red") + theme(axis.text.x = element_blank())
```

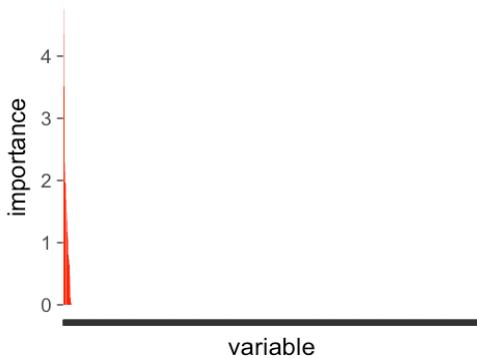


Figure 8.21: The important score of the variables by RRF

Clearly, as shown in Figure 8.21, a much smaller number of variables have non-zero importance scores, compared to ordinary random forests.

This demonstrates the superior capacity of RRF to deal with redundant variables than regular RF models.

Now, let's apply the GRRF to this dataset. The importance scores from the GRRF are shown in Figure 8.22. Similarly, the number of variables with non-zero importance scores is much smaller than ordinary random forests.

```
rf <- randomForest(X, target)
impRF <- rf$importance
impRF <- impRF[, "MeanDecreaseGini"]
imp <- impRF/(max(impRF)) #normalize the importance scores into [0,1]
gamma <- 0.1
coefReg <- (1 - gamma) * 1 + gamma * imp # each variable has a coefficient, which depends on the importance score from the ordinary RF and the parameter: gamma
grrf <- RRF(X, target, flagReg = 1, coefReg = coefReg)

imp <- as.data.frame(grrf$importance)
colnames(imp)[colnames(imp) == "MeanDecreaseGini"] <- "importance"
imp <- imp[order(imp$importance, decreasing = TRUE), , drop = FALSE]
imp$variable <- rownames(imp)
imp$variable <- factor(imp$variable, levels = as.character(imp$variable))
ggplot(data = imp, aes(x = variable, y = importance)) + geom_bar(stat = "identity",
  aes(factor(variable)), fill = "red") + theme(axis.text.x = element_blank())
```

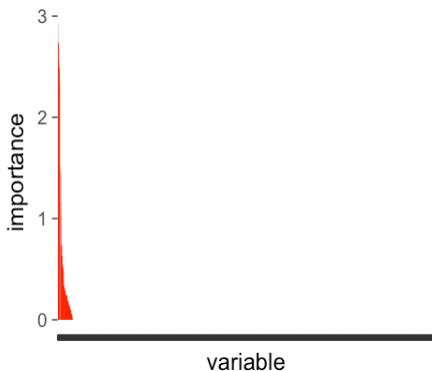


Figure 8.22: The important score of the variables by RRF

The previous figures illustrate that both RRF and GRRF use a much smaller number of variables to predict, compared to ordinary random forests. Now we evaluate the quality of the variable subset by the classification error.

Here is the procedure of evaluating the variable selection method. The dataset is split into training and testing sets with a 2:1 ratio. Different variable selection methods (e.g., by RF, RRF, and GRRF) are applied to the training set, and then, variable subsets are selected. Then, ordinary random forests are trained on the reduced training set, and applied to the testing set such that we can obtain the classification error. Each of the following experiments is run 50 times to get a robust estimate of the error rate. Note that, here, we first create the indices for the testing set (50 replicates), so that the later experiments can all consistently use the same indices.

The first variable selection method is to select the top K variables that have the top K importance scores from random forests. We study this method by changing K from 1 to 200. Results are shown in Figure 8.23.

```
set.seed(1)
testing.indices <- NULL
for (i in 1:50) {
  testing.indices <- rbind(testing.indices, sample(nrow(data),
floor(1 * nrow(data)/3)))
}

err.mat.rf <- NULL
for (K in c(1, (1:10) * 10, 150, 200)) {
  pred <- NULL
  for (i in 1:nrow(testing.indices)) {

    testing.ix <- testing.indices[i, ]
    X.training <- X[-testing.ix, ]
    target.training <- target[-testing.ix]
    X.testing <- X[testing.ix, , drop = FALSE]
    target.testing <- target[testing.ix]

    rf <- randomForest(X.training, target.training)
    impRF <- rf$importance
    impRF <- impRF[, "MeanDecreaseGini"]
```

```

ix <- order(importance, decreasing = TRUE)

X.training.new <- X.training[, ix[1:K], drop = FALSE]
rf <- randomForest(X.training.new, target.training)

target.pred <- predict(rf, X.testing)

error <- length(which(as.character(target.pred) != target.
testing))/length(target.testing)
err.mat.rf <- rbind(err.mat.rf, c(K, error))
}
}
err.mat.rf <- as.data.frame(err.mat.rf)
colnames(err.mat.rf) <- c("num_features", "error")

ggplot() + geom_boxplot(data = err.mat.rf %>% mutate(num_features =
as.factor(num_features)),
aes(y = error, x = num_features)) + geom_point(size = 3)

```

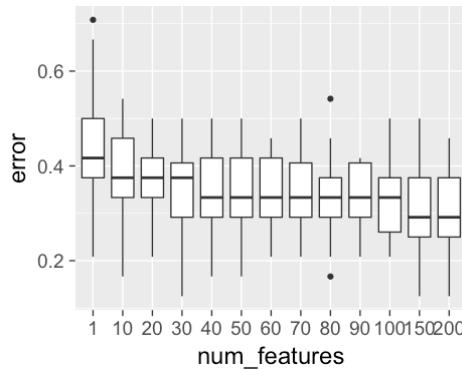


Figure 8.23: The error rates of the RF models with different number of features

From Figure 8.23, it can be seen that the error rates decrease as the number of variables increases. This makes sense as additional information is provided to the model to predict the outcome variable as more variables are added.

Next, we use RRF to conduct variable selection. This can be done by changing the coefficient parameter (`coefReg`) in RRF. The number of

selected variables should increase as the coefficient increases, and the error rates should decrease accordingly. Results are shown in Figure 8.24 and Figure 8.25.

```
set.seed(1)
err.mat.rrf <- NULL
for (coefI in c(0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 1)) {
  pred <- NULL
  for (i in 1:nrow(testing.indices)) {
    testing.ix <- testing.indices[i, ]
    X.training <- X[-testing.ix, ]
    target.training <- target[-testing.ix]
    X.testing <- X[testing.ix, , drop = FALSE]
    target.testing <- target[testing.ix]

    rrf <- RRF(X.training, target.training, coefReg = coefI)

    X.training.new <- X.training[, rrf$feaSet, drop = FALSE]
    rf <- randomForest(X.training.new, target.training)

    target.pred <- predict(rf, X.testing)
    # pred <- c(pred, as.character(target.pred) )
    error <- length(which(as.character(target.pred) != target.
testing))/length(target.testing)
    err.mat.rrf <- rbind(err.mat.rrf, c(coefI, length(rrf$fea
Set), error))
  }
  # error <- length(which(pred != target))/Length(pred) err.mat.
rrf <-
  # rbind(err.mat.rrf, c(coefI, mean(numfea.v), error))
}
err.mat.rrf <- as.data.frame(err.mat.rrf)
colnames(err.mat.rrf) <- c("coef", "num_features", "error")
# err.mat.rrf <- err.mat.rrf %>% mutate(coef=as.factor(coef))
ggplot() + geom_boxplot(data = err.mat.rrf %>% mutate(coef = as.f
actor(coef)),
aes(y = error, x = coef)) + geom_point(size = 3)
```

It is known that the maximum number of features are selected when the coefficient becomes 1. As shown in Figure 8.24, when the coefficient is around 0.95, the smallest error rates could be obtained. Also, from Figure 8.25, we can see that, indeed the number of selected variables increases when the coefficient increases. Thus, RRF provides a continuum of variable

selection controlled by the parameter `coefReg`, providing convenience in model tuning and cross-validation.

```
ggplot() + geom_boxplot(data = err.mat.rrf %>% mutate(coef = as.factor(coef)),  
aes(y = num_features, x = coef)) + geom_point(size = 3)
```

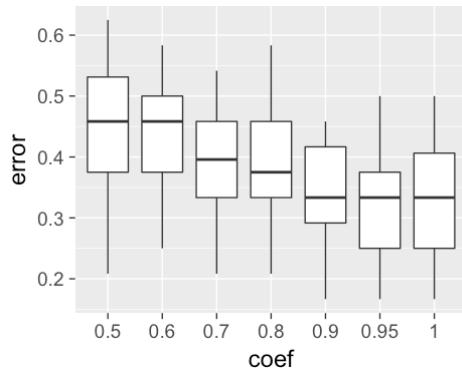


Figure 8.24: The error rates of the RF models with different values of `coef`

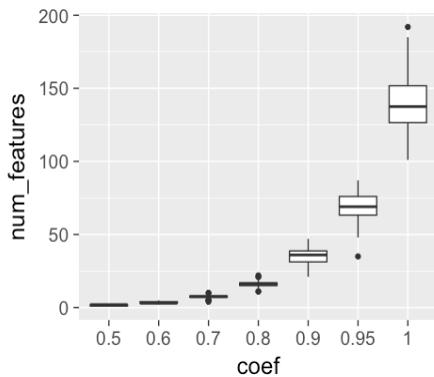


Figure 8.25: Number of features versus different values of `coef`

Furthermore, we conduct the variable selection using GRRF. We use different γ in GRRF and check the number of features and error rates. The weights come from the importance scores from random forests. As shown in Figure 8.26 and Figure 8.27, we can observe that the number of features decreases as γ increases, and the error rates increase accordingly.

```

set.seed(1)
err.mat.grrf <- NULL
for (gammaI in c(0.4, 0.3, 0.2, 0.1, 0.05, 0)) {
  pred <- NULL
  numfea.v <- NULL

  for (i in 1:nrow(testing.indices)) {
    testing.ix <- testing.indices[i, ]
    X.training <- X[-testing.ix, ]
    target.training <- target[-testing.ix]
    X.testing <- X[testing.ix, , drop = FALSE]
    target.testing <- target[testing.ix]

    rf <- randomForest(X.training, target.training)
    impRF <- rf$importance
    impRF <- impRF[, "MeanDecreaseGini"]
    imp <- impRF/(max(impRF))
    coefReg <- (1 - gammaI) * 1 + gammaI * imp

    grrf <- RRF(X.training, target.training, flagReg = 1, coe
fReg = coefReg)

    # numfea.v <- c(numfea.v, Length(grrf$feaSet))
    X.training.new <- X.training[, grrf$feaSet, drop = FALSE]

    rf <- randomForest(X.training.new, target.training)

    target.pred <- predict(rf, X.testing)
    # pred <- c(pred, as.character(target.pred) )
    error <- length(which(as.character(target.pred) != target.
testing))/length(target.testing)
    err.mat.grrf <- rbind(err.mat.grrf, c(gammaI, length(grrf
$feaSet), error))
  }
}

err.mat.grrf <- as.data.frame(err.mat.grrf)
colnames(err.mat.grrf) <- c("gamma", "num_features", "error")

```

```
# err.mat.grrf <- err.mat.grrf %>% mutate(gamma=as.factor(gamma))
ggplot() + geom_boxplot(data = err.mat.grrf %>% mutate(gamma = as.factor(gamma)),
aes(y = error, x = gamma)) + geom_point(size = 3)
```

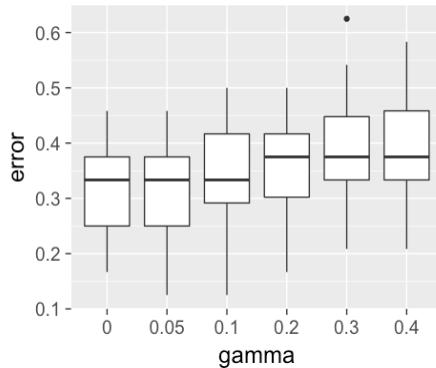


Figure 8.26: The error rates of the RF models with different values of gamma in RRF

```
ggplot() + geom_boxplot(data = err.mat.grrf %>% mutate(gamma = as.factor(gamma)),
aes(y = num_features, x = gamma)) + geom_point(size = 3)
```

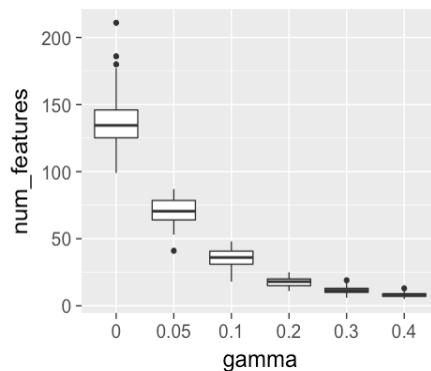


Figure 8.27: Number of features versus different values of gamma

Now, we compile the results from all the methods, and plot the average error rates at each number of features in Figure 8.28. For RRF and GRRF, the average number of features and average error rates of each parameter setting are used. It can be seen that between 40 and 80 features, the RRF and GRRF methods have lower error rates than the RF model that uses the top K features according to the importance scores generated by the RF model. As K increases, the error rate of using the top K variables according to the RF importance scores continues to decrease, lower than using the variables selected from RRF or GRRF. This means that RRF and GRRF can miss some informative variables. However, since this dataset is small, and the use of cross-validation makes it even smaller, the difference may not be as significant. To verify this hypothesis, we also plot the average errors with $+/- 1$ standard deviation in Figure 8.29. It can be seen that all the methods have similar error rate ranges when a certain number of variables is selected. However, an advantage for RRF and GRRF is that they can efficiently determine the number of variables needed by changing the coefficient or γ .

```

err.mat.rf <- as.data.frame(err.mat.rf)
err.mat.rf$method <- "RF"
err.mat.rrf <- as.data.frame(err.mat.rrf)
err.mat.rrf$method <- "RRF"
err.mat.grrf <- as.data.frame(err.mat.grrf)
err.mat.grrf$method <- "GRRF"
err.mat.rrf.summary <- err.mat.rrf %>% group_by(coef, method) %>%
  summarize(num_features = mean(num_features),
            sd = sd(error), error = mean(error), upper = error + sd, lower =
  r = error -
            sd) %>% ungroup()
err.mat.grrf.summary <- err.mat.grrf %>% group_by(gamma, method) %>%
  summarize(num_features = mean(num_features),
            sd = sd(error), error = mean(error), upper = error + sd, lower =
  r = error -
            sd) %>% ungroup()
err.mat.rf.summary <- err.mat.rf %>% group_by(num_features, method) %>%
  summarize(sd = sd(error),
            error = mean(error), upper = error + sd, lower = error - sd) %>% ungroup()
err.mat <- rbind(err.mat.rf.summary[, c("num_features", "error",
  "method", "lower",

```

```

    "upper")], err.mat.rrf.summary[, c("num_features", "error", "method",
    "lower", "upper")], err.mat.grrf.summary[, c("num_features", "error",
    "method", "lower", "upper")])

ggplot(err.mat, aes(x = num_features, y = error, group = method,
colour = method)) +
  geom_line(linetype = "dashed") + geom_point()

```

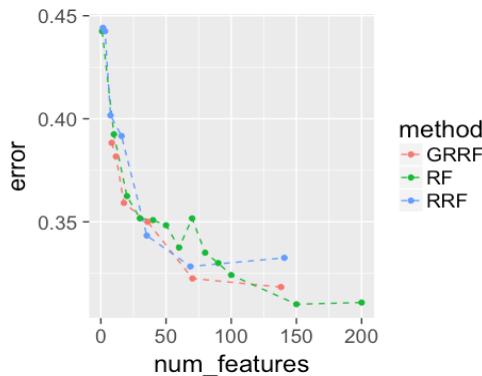


Figure 8.28: The error rates of the RF models with different number of features by RF, RRF, and GRRF

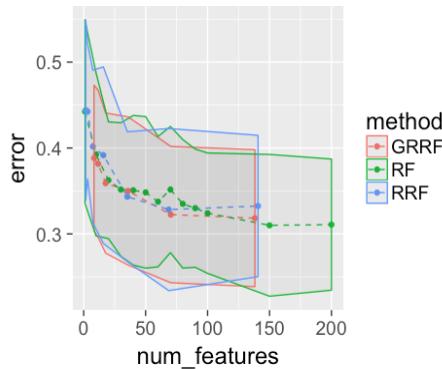


Figure 8.29: The error rates (and their upper and lower bounds) of the RF models with different number of features by RF, RRF, and GRRF

```

ggplot(err.mat, aes(x = num_features, y = error, group = method,
colour = method)) +
  geom_line(linetype = "dashed") + geom_point() + geom_ribbon(d
ata = err.mat,
  aes(ymin = lower, ymax = upper), alpha = 0.05)

```

In addition, a more extensive study on more datasets has found that the RF model with the variables selected from RRF or GRRF can significantly outperform the RF model that uses all the features. This provides evidence that the RRF and GRRF indeed can provide better variable selection results in this data, which is consistent with theoretical arguments illustrated using the exemplary dataset shown in Table 8.3, and the simulation results we have shown in Figures 8.20 – 8.22.

IV. Exercises

Data analysis

1. Find 5 classification datasets from the UCI data repository or R datasets. Use LASSO to select variables. Then, based on the reduced dataset, conduct a detailed analysis using the logistic regression model, SVM, decision tree, random forest, and AdaBoost. Conduct model selection and validation. Use cross-validation to select the best models.
2. Repeat 1, but use random forest to select variables. Compare the final models with the ones based on LASSO.
3. Repeat 1, but use PCA to identify the top PCs to replace the original variables. Compare the final models with the ones based on LASSO.
4. Find 5 regression datasets from the UCI data repository or R datasets. Repeat 1 and 3.

Programming

5. Write your own R script to implement the shooting model. Compare your results with `glm()`.

6. Write your own R script to implement the PCA model. Compare your results with `pca()`.

CHAPTER 9: CRAFTMANSHIP

MODEL EXTENSION/STACKING

I. Overview

Chapter 8 is about “**craftsmanship**”. It recognizes the complexity of real-world problems, and highlight how we can modify or combine existing methods in integrative ways to solve a problem. Three examples are given, including the Kernel regression model that generalized the idea of linear regression, the conditional variance regression model that interestingly layers one regression model into another, and the tree-based quality monitoring method that converts traditional quality monitoring into classification framework.

II. Kernel Regression Model

II.1 Rationale and Formulation

Linear regression model looks intuitive, but it is built on very strong assumptions. One strong assumption is that, the linear regression model treats any data point in a global fashion. In other words, while a data point

is located in a particular geographical area, it has impact on the model's prediction on any other location in the space. This could be irrational, as in some applications the data point collected in a local area may only tell information about that area, not easily generalizable to the whole space. This risk is shown in Figure 9.1.

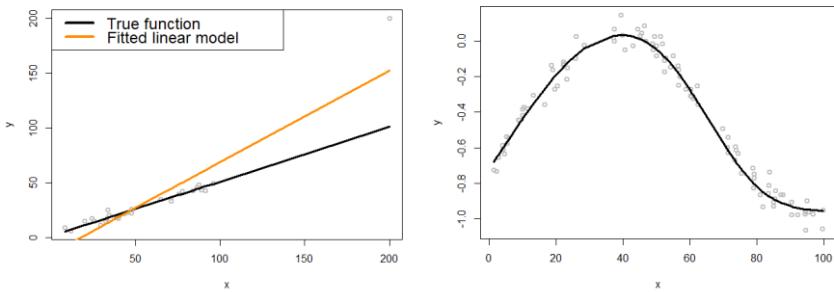


Figure 9.1: Risk of linear regression model as a global model. (left) A single outlier could impact the regression model as a whole; (Right) Many data problems call for localized regression models

The R code for conducting the experiment shown in the left figure in Figure 9.1 is shown in below.

```
# Write a nice simulator to generate dataset with one predictor and one outcome
# from a polynomial regression model
require(splines)

seed <- rnorm(1)
set.seed(seed)
gen_data <- function(n, coef, v_noise) {
  eps <- rnorm(n, 0, v_noise)
  x <- sort(runif(n, 0, 100))
  X <- cbind(1,ns(x, df = (length(coef) - 1)))
  y <- as.numeric(X %*% coef + eps)
  return(data.frame(x = x, y = y))
}
```

```

n_train <- 30
coef <- c(1,0.5)
v_noise <- 3
tempData <- gen_data(n_train, coef, v_noise)
tempData[31,] = c(200,200)
# Fit the data using Linear regression model
x <- tempData[, "x"]
y <- tempData[, "y"]
fit <- lm(y~x,data=tempData)
# Plot the data
x <- tempData$x
X <- cbind(1, x)
y <- tempData$y
plot(y ~ x, col = "gray", lwd = 2)
lines(x, X %*% coef, lwd = 3, col = "black")
lines(x, fitted(fit), lwd = 3, col = "darkorange")
legend(x = "topleft", legend = c("True function", "Fitted linear
model"), lwd = rep(4, 4), col = c("black", "darkorange"), text.wi
dth = 100, cex = 1.5)

```

So how to fix this problem, while on the other hand we don't want to derivate from the linear regression framework too far?

One approach we could utilize is to look at the linear regression model in a new perspective. Statisticians and data scientists who innovate on modeling use this approach of look-for-a-new-perspective all the time¹.

Let's look at the simple linear regression problem $y = \beta_0 + \beta_1 x$. Let's further simplify it by assuming that we know the mean of y is zero, so is the mean of x . This will lead to the model as $y = \beta_1 x$ and the estimator of β_1 as

$$\beta_1 = \frac{(\sum_{i=1}^n x_i y_i)}{\sum_{i=1}^n x_i^2}.$$

Thus, when we try to make prediction on a new data point with a given x^* , the prediction y^* will be

¹ Some examples for interested readers: Neal, R. *Bayesian learning for neural networks*, Springer Verlag 1996. Lee, K. and Kim, J. *On the equivalence of linear discriminant analysis and least squares*, AAAI 2005. Ye, J. *Least squares linear discriminant analysis*, ICML 2007. Li, F., Yang, Y. and Xing, E. *From LASSO regression to feature vector machine*, NIPS 2005.

$$y^* = x^* \frac{(\sum_{i=1}^n x_i y_i)}{\sum_{i=1}^n x_i^2}.$$

This could be further reformed as:

$$y^* = \sum_{i=1}^n y_i \frac{x_i}{\sum_{i=1}^n x_i^2} x^*,$$

which is equivalent with

$$y^* = \sum_{i=1}^n y_i \frac{x_i x^*}{n S_x^2}.$$

Now if we look closely at this formula, we can draw interesting observations how linear regression model works in prediction on a new location using its knowledge on other locations (e.g., the historical data points (x_i, y_i) for $i = 1, 2, \dots, n$). It first evaluates the similarity between the new location with each of the knowing locations, as reflected in $\frac{x_i x^*}{n S_x^2}$, where $x_i x^*$ calculates the similarity and $n S_x^2$ is a normalization factor. Then, the prediction y^* is a weighted sum of y_i for $i = 1, 2, \dots, n$ while the weight of y_i is proportional to the similarity between x_i and x^* . From this perspective, we see linear regression model as a very empirical prediction model that bears the same idea with those lazy learning methods such as k-nearest-neighbor regression model or local regression models. The difference here, in the linear regression model, is that a special similarity measure (i.e., $\frac{x_i x^*}{n S_x^2}$) is used, that means the weight of a data point depends on how far it is from the center of the data, not how far it is from the point at which we are trying to predict. Thus, for this similarity measure to work we need to hope that the underlying model is globally linear.

II.2 Theory and Method

We then pursue a generalized family of model, defined as:

$$y^* = \sum_{n=1}^N y_n w(x_n, x^*).$$

Here, $w(x_n, x^*)$ is the weight that characterizes the similarity between the point that will be predicted on (i.e., x^*) and the existing data points, x_n for $n = 1, 2, \dots, N$. Roughly speaking, there are two types of methods to define this similarity metric.

One is the **K-nearest neighbor (KNN) smoother**:

$$w(x_n, x^*) = \begin{cases} \frac{1}{k}, & \text{if } x_n \text{ is one of the } k \text{ nearest neighbors of } x^* \\ 0, & \text{if } x_n \text{ is NOT in the } k \text{ nearest neighbors of } x^*. \end{cases}$$

Here, to define the neighbors of a data point, a distance function is needed, e.g., a popular one is the Euclidean distance function.

Note that, a distinct feature of the KNN smoother is the discrete manner to define similarity between data points. Which is, a data point is either a neighbor of another data point, or not. Not like this, the **kernel smoother** is another approach that has the continuity in similarity between data points. A kernel smoother defines $w(x_n, x^*)$ in the following manner:

$$w(x_n, x^*) = \frac{K(x_n, x^*)}{\sum_{n=1}^N K(x_n, x^*)}.$$

There have been many kernel functions developed, for example, as shown in Table 9.1:

Table 9.1: Some kernel functions used in machine learning

Kernel function	Mathematical form	Parameters
Linear	$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$	null
Polynomial	$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^q$	q
Gaussian radial basis	$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \ \mathbf{x}_i - \mathbf{x}_j\ ^2}$	$\gamma \geq 0$
Laplace radial basis	$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \ \mathbf{x}_i - \mathbf{x}_j\ }$	$\gamma \geq 0$
Hyperbolic tangent	$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{x}_i^T \mathbf{x}_j + b)$	b
Sigmoid	$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(a \mathbf{x}_i^T \mathbf{x}_j + b)$	a, b
Bessel function	$K(\mathbf{x}_i, \mathbf{x}_j) = \frac{\text{bessel}_{v+1}^n(\sigma \ \mathbf{x}_i - \mathbf{x}_j\)}{(\ \mathbf{x}_i - \mathbf{x}_j\)^{-n(v+1)}}$	σ, n, v
ANOVA radial basis	$K(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^n e^{-\sigma(x_i^k - x_j^k)} \right)^d$	σ, d

II.3 R Lab

Using the established framework in Chapter 5 to generate nonlinear dataset, here, we use the following R code to implement the KNN regression model.

```
# Simulate one batch of data
n_train <- 100
coef <- c(-0.68, 0.82, -0.417, 0.32, -0.68)
v_noise <- 0.2
n_df <- 20
df <- 1:n_df
tempData <- gen_data(n_train, coef, v_noise)
# Fit different KNN models
x <- tempData$x
X <- cbind(1, ns(x, df = (length(coef) - 1)))
y <- tempData$y
# install.packages("FNN")
require(FNN)

## Loading required package: FNN

xy.knn3<- knn.reg(train = x, y = y, k=3)
xy.knn10<- knn.reg(train = x, y = y, k=10)
xy.knn50<- knn.reg(train = x, y = y, k=50)
```

Then, we draw the true model (as a black curve) and the sampled data points using the following R code. Result is shown in Figure 9.2.

```
# Plot the data
plot(y ~ x, col = "gray", lwd = 2)
```

And we further layer the fitted KNN regression models with different choices on the parameter k onto the figure.

```
lines(x, X %*% coef, lwd = 3, col = "black")
lines(x, xy.knn3$pred, lwd = 3, col = "darkorange")
lines(x, xy.knn10$pred, lwd = 3, col = "dodgerblue4")
lines(x, xy.knn50$pred, lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("True function", "KNN (k = 3)",
"KNN (k = 10)", "KNN (k = 50)"),
lwd = rep(3, 4), col = c("black", "darkorange", "dodgerblue4",
"forestgreen"),
text.width = 32, cex = 0.85)
```

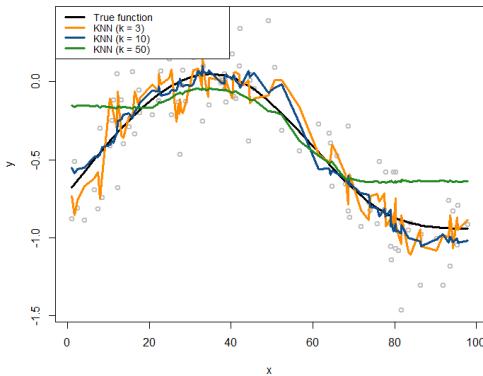


Figure 9.2: KNN regression models with different choices on the number of nearest neighbors

It can be seen that, with smaller number of nearest neighbors, the fitted curve by the KNN regression model is less smooth. That means, a KNN regression model with a smaller parameter k tends to predict on a data point by relying on only a few local data points, ignoring information provided by other data points that are far away. This is very different from the spirit of linear regression model, in which no matter how far away a data point is, it can change predictions on any other data point globally as it changes the regression line as a whole.

A related observation is, in terms of model complexity, the smaller the parameter k , the more complex the regression model. This is often taken as a counterintuitive conclusion.

Similarly, we can repeat the experiments introduced above for implementing the kernel smoother regression model. Here, we use the Gaussian radial basis kernel function in the kernel smoother regression model. Result is shown in Figure 9.3.

```
# Repeat the above experiments with kernel smoother
# Plot the data
plot(y ~ x, col = "gray", lwd = 2)
lines(x, X %*% coef, lwd = 3, col = "black")
lines(ksmooth(x,y, "normal", bandwidth=2), lwd = 3, col = "darkora")
```

```

nge")
lines(ksmooth(x,y, "normal", bandwidth=5), lwd = 3, col = "dodgerblue4")
lines(ksmooth(x,y, "normal", bandwidth=15), lwd = 3, col = "forest green")
legend(x = "topright", legend = c("True function", "Kernel Reg (bw = 2)", "Kernel Reg (bw = 5)", "Kernel Reg (bw = 15)"),
       lwd = rep(3, 4), col = c("black", "darkorange", "dodgerblue4", "forestgreen"),
       text.width = 32, cex = 0.85)

```

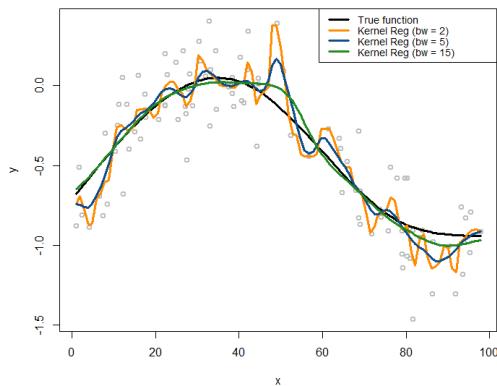


Figure 9.3: Kernel regression models with different choices on the `bandwidth` parameter of the Gaussian radial basis kernel function

As shown in Figure 9.3, the `bandwidth` parameter in the kernel smoother regression plays a similar role as the parameter k in KNN regression. The larger the bandwidth, the smoother of the regression curve. On the other hand, it can be seen that the curve of kernel smoother is smoother than the KNN curves. This observation corresponds to the note we mentioned in the beginning of this subsection that KNN is discretely parameterized while kernel smoother introduces smoothness and continuity into the definition of the neighbors of a data point (thus no hard thresholding is needed to classify whether a data point is a neighbor of another data point).

III. Conditional Variance Regression Model

II.1 Rationale and Formulation

Another common complication when applying linear regression model in real-world applications is that the variance of the response variable may also change. This phenomenon is called as **heteroscedasticity** in regression analysis. This complication can be taken care of by a conditional variance regression model that allows the variance of the response variable to be a (usually implicit) function of the input variables. This leads to the following model:

$$y = \beta^T x + \epsilon_x,$$

and ϵ_x is the error term that is a normal distribution with varying variance:

$$\epsilon_x \sim N(0, \sigma_x^2).$$

The remaining issue is how to estimate the regression parameters.

II.2 Theory and Method

Known σ_x^2 : If we have known the σ_x^2 , this will lead to the following scheme for parameter estimation of the unknown regression parameters. The likelihood function is:

$$-\frac{n}{2} \ln 2\pi - \frac{1}{2} \sum_{n=1}^N \log \sigma_{x_n}^2 - \frac{1}{2} \sum_{n=1}^N \frac{(y_n - \beta^T x_n)^2}{\sigma_{x_n}^2}.$$

As we have known σ_x^2 , the parameters to be estimated only involve the last part of the likelihood function. Thus, we estimate the parameters that minimize

$$\frac{1}{2} \sum_{n=1}^N \frac{(y_n - \beta^T x_n)^2}{\sigma_{x_n}^2}.$$

This could be written in the matrix form as

$$\min_{\beta} (\mathbf{Y} - \mathbf{X}\beta)^T \mathbf{W} (\mathbf{Y} - \mathbf{X}\beta),$$

where \mathbf{W} is a diagonal matrix with its diagonal elements as $\mathbf{W}_{nn} = \frac{1}{\sigma_{x_n}^2}$.

To solve this optimization problem, we can take the gradient of the objective function and set it to be zero:

$$\frac{\partial (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T \mathbf{W} (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0,$$

which gives rise to the equation:

$$\mathbf{X}^T \mathbf{W} (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) = 0.$$

This leads to the weighted least square estimator of $\boldsymbol{\beta}$ as

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{Y}.$$

Unknown σ_x^2 : A more complicated situation, also more realistic situation, is that we don't know σ_x^2 . This means that we need to estimate σ_x^2 . To do so, it is important to recognize that this problem bears a regression core in its formulation. It is to use the input variables \mathbf{x} to predict a new outcome variable, σ_x^2 . The only complication here is that, we don't have the "natural measurements" of the new outcome variable that is needed to apply the regression method. Since, here, the outcome variable σ_x^2 is not directly measurable. This is a **latent variable** in statistics.

To overcome this problem, we can estimate the measurements of the latent variable, denoted as $\hat{\sigma}_{x_n}^2$ for $n = 1, 2, \dots, N$. This philosophy of taking some variables as latent variables and further using statistical estimation/inference to fill in the unseen measurements is popular and fertile in statistics that underlies many models such as the latent factor models, structural equation models, missing values imputation, EM algorithm, Gaussian mixture model, graphical models with latent variables, etc.

Thus, we propose the following steps:

1. Initialize $\hat{\sigma}_{x_n}^2$ for $n = 1, 2, \dots, N$, by any reasonable approach including the random generation of values.
2. Build a regression model for the mean of the response variable using the weighted LS estimator. Estimate $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{Y}$ and get $\hat{y}_n = \hat{\boldsymbol{\beta}}^T \mathbf{x}_n$.
3. Derive the residuals $\hat{\epsilon}_n = y_n - \hat{y}_n$.

4. Build a regression model, e.g., using the kernel regression which is a nonparametric method, to fit $\hat{\varepsilon}_n^2$ using \mathbf{x}_n for $n = 1, 2, \dots, N$.
5. Predict $\hat{\sigma}_{\mathbf{x}_n}^2$ for $n = 1, 2, \dots, N$ using the fitted regression model in Step 3.
6. Repeat Step 2 – Step 5 until convergence or satisfaction of a stopping criteria (could be a fixed number of iterations or small change of parameters).

We can see that the proposed conditional variance regression model is a composition of two regular linear regression models to work out the heteroscedasticity. This is a typical model stacking strategy to create new models based on existing models.

II.3 R Lab

We first simulate dataset to see how the proposed iterative procedure of the conditional variance regression model can work out the heteroscedasticity. The simulated data has one predictor and one outcome. The model parameters (including the intercept and regression coefficient) are assigned values as `coef <- c(1,0.5)`. The variance is a function of the predictor x , i.e., which equals to $0.5+0.8*x^2$.

```
# Conditional variance function
# Simulate a regression model with heterogeneous variance
gen_data <- function(n, coef, v_noise) {
  x <- rnorm(100,0,2)
  eps <- rnorm(100,0,sapply(x,function(x){0.5+0.8*x^2}))
  X <- cbind(1,x)
  y <- as.numeric(X %*% coef + eps)
  return(data.frame(x = x, y = y))
}
n_train <- 100
coef <- c(1,0.5)
v_noise <- 2.5
tempData <- gen_data(n_train, coef, v_noise)
```

While this data presents a typical heteroscedasticity problem, in what follows we apply a regular linear regression model with assumption of

homogenous variance. The fitted line is shown in Figure 9.4, indicating a significant derivation from the true regression model.

```
# Fit the data using linear regression model (OLS)
x <- tempData[, "x"]
y <- tempData[, "y"]
fit.ols <- lm(y~x,data=tempData)
# Plot the data and the models
x <- tempData$x
X <- cbind(1, x)
y <- tempData$y
plot(y ~ x, col = "gray", lwd = 2)
# Plot the true model
lines(x, X %*% coef, lwd = 3, col = "black")
# Plot the linear regression model (OLS)
lines(x, fitted(fit.ols), lwd = 3, col = "darkorange")
legend(x = "topleft", legend = c("True function", "Linear model
(OLS)"),
lwd = rep(4, 4), col = c("black", "darkorange"), text.widt
h = 4, cex = 1)
```

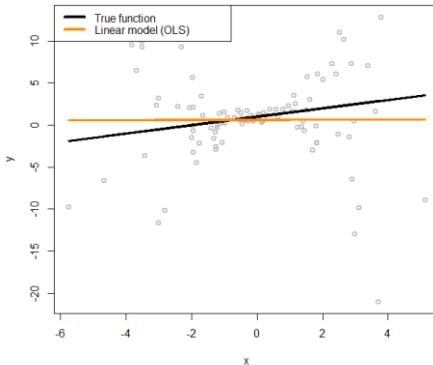


Figure 9.4: Linear regression model to fit a heteroscedastic dataset

We can generate the residuals based on the fitted regular linear regression model, which are plotted in Figure 9.5. A nonlinear regression model, the kernel smoother regression model implemented by `npreg()`, is fitted on these residuals. The true function of the variance is also shown as

the black line in Figure 9.5. It can be seen that the residuals encode the information for us to approximate the underlying true variance function.

```
# Plot the residual estimated from the Linear regression model (OLS)
plot(x,residuals(fit.ols)^2,ylab="squared residuals",col = "gray",
lwd = 2)
# Plot the true model underlying the variance of the error term
curve((1+0.8*x^2)^2,col = "black", lwd = 3, add=TRUE)
# Fit a nonlinear regression model for residuals
# install.packages("np")
require(np)

var1 <- npreg(residuals(fit.ols)^2 ~ x)

grid.x <- seq(from=min(x),to=max(x),length.out=300)
lines(grid.x,predict(var1,exdat=grid.x), lwd = 3, col = "darkorange")
legend(x = "topleft", legend = c("True function", "Fitted nonlinear model (1st iter)"),
lwd = rep(4, 4), col = c("black", "darkorange"), text.width = 5, cex = 1.2)
```

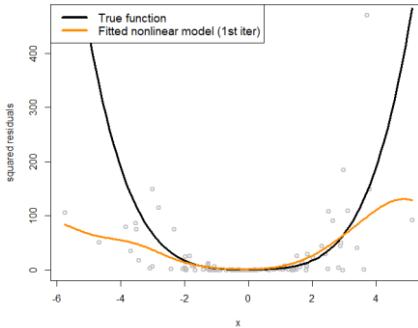


Figure 9.5: Nonlinear regression model to fit the residuals

Thus, we fit another linear regression model with weights of the data points assigned according to the inverse of the variance, i.e., `weights=1/fitted(var1)`, to penalize the influence of the data points that have larger variances on the fit of the regression line. The new regression model is added into Figure 9.4, which generates Figure 9.6. It can be seen

that, with this strategy, the new regression model (the green line) is much more closer with the true model than the regular linear regression model.

```
# Fit a Linear regression model (WLS) with the weights specified
# by the fitted nonlinear model of the residuals
fit.wls <- lm(y~x,weights=1/fitted(var1))
plot(y ~ x, col = "gray", lwd = 2,ylim = c(-20,20))
# Plot the true model
lines(x, X %*% coef, lwd = 3, col = "black")
# Plot the Linear regression model (OLS)
lines(x, fitted(fit.ols), lwd = 3, col = "darkorange")
# Plot the Linear regression model (WLS) with estimated variance
# function
lines(x, fitted(fit.wls), lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("True function", "Linear (OLS)", "Linear (WLS) + estimated variance"),
       lwd = rep(4, 4), col = c("black", "darkorange", "forestgreen"), text.width = 5, cex = 1)
```

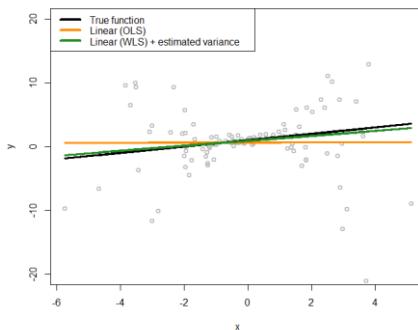


Figure 9.6: Fit the heteroscedastic dataset with two linear regression models using OLS and WLS (that accounts for the heteroscedastic effects with a nonlinear regression model to model the variance regression)

This process could proceed with updating the fitted variance function on the new residuals, as shown in Figure 9.7. Here, it seems that the use of the kernel smoother by `npreg` hits its limit as a local and data-driven model, that could not correctly fit the curve in the two ends. If we have known the form of the variance function as a second order polynomial function, we could use parametric regression model to attain this fitting.

```

# Plot the residual estimated from the Linear regression model (OLS)
plot(x,residuals(fit.ols)^2,ylab="squared residuals",col = "gray",
lwd = 2)
# Plot the true model underlying the variance of the error term
curve((1+0.8*x^2)^2,col = "black", lwd = 3, add=TRUE)
# Fit a nonlinear regression model for residuals
# install.packages("np")
require(np)
var2 <- npreg(residuals(fit.wls)^2 ~ x)

grid.x <- seq(from=min(x),to=max(x),length.out=300)
lines(grid.x,predict(var1,exdat=grid.x), lwd = 3, col = "darkorange")
lines(grid.x,predict(var2,exdat=grid.x), lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("True function", "Fitted nonlinear model (1st iter)", "Fitted nonlinear model (2nd iter)"),
lwd = rep(4, 4), col = c("black", "darkorange", "forestgreen"), text.width = 6, cex = 1.2)

```

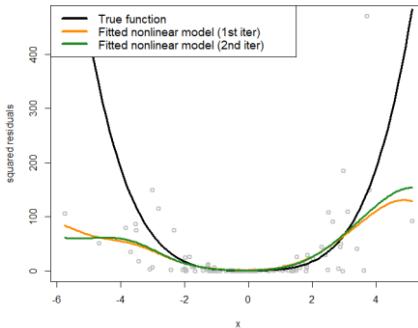


Figure 9.7: Nonlinear regression model to fit the residuals in the 2nd iteration

Now let's apply the conditional variance regression model on the AD dataset. As what we did in the simulated dataset, we first fit a regular linear regression model, then, use the kernel smoother regression model to fit the residuals, and further fit a weighted linear regression model with weights of the data points being assigned according to the inverse of the variance, i.e., `weights=1/fitted(var1)`, to penalize the influence of the data points that

have larger variances on the fit of the regression line. Results are shown in Figure 9.8.

```
AD <- read.csv('AD_b1.csv', header = TRUE)
str(AD)

# Fit the data using linear regression model (OLS)
x <- AD$HippoNV
y <- AD$MMSCORE
fit.ols <- lm(y~x,data=AD)

# Fit a Linear regression model (WLS) with the weights specified
# by the fitted nonlinear model of the residuals
var1 <- npreg(residuals(fit.ols)^2 ~ HippoNV, data = AD)

fit.wls <- lm(y~x,weights=1/fitted(var1))

plot(y ~ x, col = "gray", lwd = 2)
# Plot the Linear regression model (OLS)
lines(x, fitted(fit.ols), lwd = 3, col = "darkorange")
# Plot the Linear regression model (WLS) with estimated variance
# function
lines(x, fitted(fit.wls), lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("Linear (OLS)", "Linear (WLS) +
estimated variance"),
lwd = rep(4, 4), col = c("darkorange","forestgreen"), text.
width = 0.2, cex = 1)
```

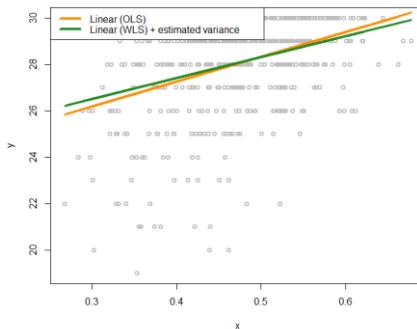


Figure 9.8: Fit the heteroscedastic AD dataset with two linear regression models using OLS and WLS (that accounts for the heteroscedastic effects with a nonlinear regression model to model the variance regression)

We can also visualize the fitted variance functions in Figure 9.9 via the following R code.

```
# Plot the residual estimated from the Linear regression model (OLS)
plot(x,residuals(fit.ols)^2,ylab="squared residuals",col = "gray",
lwd = 2)
# Fit a nonlinear regression model for residuals
# install.packages("np")
require(np)
var2 <- npreg(residuals(fit.wls)^2 ~ x)

grid.x <- seq(from=min(x),to=max(x),length.out=300)
lines(grid.x,predict(var1,exdat=grid.x), lwd = 3, col = "darkorange")
lines(grid.x,predict(var2,exdat=grid.x), lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("Fitted nonlinear model (1st iter)",
" Fitted nonlinear model (2nd iter)"),
lwd = rep(4, 4), col = c( "darkorange", "forestgreen"), text.width = 0.25, cex = 1.2)
```

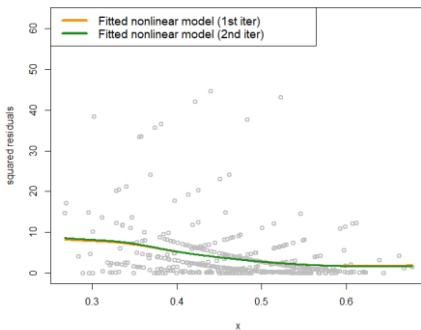


Figure 9.9: Nonlinear regression model to fit the residuals in the 2nd iteration for the AD data

It can be seen that, the heteroscedasticity problem is significant in this problem. Learning the variance function is helpful in this context in terms of at least two aspects. First, in terms of the statistical aspect, it improves the fitting of the regression line. Second, knowing the variance function

itself is important knowledge in healthcare, e.g., as variance implies unpredictability or low quality in healthcare operations.

II.4 Remark

For regression problems, the interest is usually on the modeling of the relationship between the mean of the outcome variable with the input variables. Thus, when there is heteroscedasticity in the data, a nonparametric regression method is recommended to estimate the latent variance information, more from a curve fitting perspective which is to smooth and estimate, rather than a modeling perspective to study the relationship between the outcome variable with input variables. But, of course, this usual tendency doesn't exclude the possibility that we can still study how the input variables affect the variance of the response variable explicitly. Specifically, as the linear regression as we know is a model to link the mean of Y with the input variables X , we can develop an analogical linear regression model to link the variance of Y with the input variables X . The iterative procedure developed above is still applicable here.

IV. System Monitoring as a Decision Tree Model

IV.1 Rationale and Formulation

Another method we'd like to introduce is an interesting framework that was proposed¹ to convert quality monitoring problem in statistical quality control into a classification problem. This is built on the following insight about quality monitoring as a statistical problem that does things with data. In a quality monitoring problem, we often collect the so-called "reference data" from the process in normal conditions. The objective of quality monitoring of this process is to trigger alerts if the new data that come in real time deviate from the reference data. It is hoped that the alerts could

¹ Deng, H., Runger, G. and Tuv, E. *System monitoring with real-time contrasts*. *Journal of quality technology*, 2012

correspond to real anomaly happening in the process, with a small rate of false positive (i.e., alerts triggered when the process is actually normal).

Figure 9.10 shows a monitoring problem. At time 1 to time 40, the reference data are collected from a normal process. From time 41, we monitor the process with real-time data. As the process from time 41 to time 80 is under normal condition, we should not trigger any alert. From time 81, the process is abnormal and therefore it is expected that the quality monitoring system should trigger an alert as soon as possible.

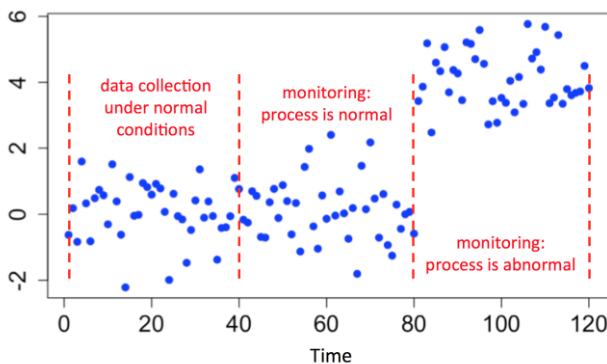


Figure 9.10: Illustration of the quality monitoring problem

In this 1-dimensional example, i.e., monitoring a process using one variable, it is easy to see that the data has an increased mean. However, when there are multiple dimensions, it is desirable to know which variables are causing the process abnormal. This so-called **fault diagnosis** problem is crucial. In this section, we discuss how we convert the problem to a classification problem where decision trees can be used for both monitoring and diagnosis.

IV.2 Theory/Method

Let \mathbf{x} to be a p dimensional vector of the process. First, we collect a few data points under normal condition which form the **reference data**. Let $f_0(\mathbf{x})$ denote the distribution of the p process variables when the system is

under normal condition. Let $f_1(\mathbf{x})$ denote the distribution of the data points in monitoring. The goal of monitoring is to trigger an alert as quick as possible if $f_1(\mathbf{x})$ differs from $f_0(\mathbf{x})$, while at the same time, to reduce false alert when $f_1(\mathbf{x})$ is the same as $f_0(\mathbf{x})$.

One may notice that this is a typical scenario considered in multivariate quality monitoring literature. Indeed, if the variables are continuous as considered in most existing multivariate quality control charts, one may use traditional control charts like Hoteling's T^2 chart. However, these methods have difficulty handling more complex datasets, e.g., having both categorical and continuous variables. With the transformation of the monitoring problem into a stack of classification problems, we can overcome these data challenges as described in what follows.

Table 9.2: An exemplary time series dataset with 4 time points

Time	1	2	3	4
Value	2	1	3	3

Here we introduce the real-time contrasts method (RTC). The key idea of RTC method is to have a sliding window, with length of L , that includes the most recent data points to be compared with the reference data. Specifically, we label the reference data as one class, and the data points in the sliding window as another class, formulating a classification problem. The intuition is that, if the two data sets come from the same distribution, it is difficult to classify the two data sets and will result in a large classification error. But, if the training error is small, the real-time data may be different from the reference data, and will result in a small classification error. Therefore, the training error in building the classification model can be used as a metric indicating the difficulty classifying the two datasets.

Here we illustrate this intuition through a 1-dimensionaional problem where the variable takes values either as 1 or 2. We also assume that, the reference data have been collected as $\{1,2\}$. The collected data for monitoring is shown in the Table 9.2.

To monitor the process, we use a window size of 2. That is say, the first monitoring action takes place at time 2 since we can collect two data points to compare with the reference data. At time 2, the 2 most recent data points are 1 and 2. The reference data set $\{1,2\}$ is labeled as class 0, and the data points captured by the window with size of 2, i.e., $\{2,1\}$, are labeled as class 1. It can be seen that these two data sets are identical. Thus, the classification error rate is 0.5, which is very large.

At time 3, the sliding window now includes data points $\{1,3\}$. A classification rule “value ≤ 2 , then class 0; else class 1” would achieve the best classification error rate as 0.25.

At time 4, the sliding window includes data points $\{3,3\}$. The same classification rule “value ≤ 2 , then class 0; else class 1” can classify all example correctly with error rate of 0.

Through this example we can see that the classification error rate could be a **monitoring statistic** to guide the triggering of alerts. While we use a simple classification rule in this example since we only have one process variable with very simple value domains, in more complex problems, we can use other classification models such as Random forest models.

Actually, through in-depth research into this idea of directly using classification error rate as the monitoring statistic, a limitation soon reveals itself. Considering the number of data points in the monitoring window, which is L . The number of possible distinct classification error rate values are actually limited to be $L + 1$. This suggests that, while the monitoring statistic should be a continuum, the resolution of the classification error rate to reflect the continuum maybe limited if the window size is too small. This will result in gaps between the monitoring statistics, failing to capture changes that happen in the gaps which are blind zones.

As a remedy, the probability estimates of the data points can be used to replace the errors of the data points. The probability estimates are continuous indicators, while the errors are binary indicators. Then, the sum of the probability estimates from all data points in the sliding window can be used for monitoring, which is defined as:

$$p_t = \frac{\sum_{i=1}^w \hat{p}_1(\mathbf{x}_i)}{w}.$$

Here, \mathbf{x}_i is the i^{th} data point in the sliding window, w is the window size, and $\hat{p}_1(\mathbf{x}_i)$ is the probability estimate of \mathbf{x}_i belonging to $f_1(\mathbf{x})$. At each time point in monitoring, we can obtain a p_t . Following the tradition of control chart, we could chart the time series of p_t and observe the patterns to see if alerts should be triggered.

Besides this monitoring capacity, on the other hand, we could also use the classification model for fault diagnosis. Specifically, when the random forest is used for classification, the importance scores from random forests can be used for fault diagnosis. When process is under normal conditions and the classification errors are expected to be high, the importance scores are expected to be equal among process variables as none of them contribute to the classification problem. When process is abnormal, classification errors should be reduced, and the variables responsible for the process abnormality should now have larger importance scores. This gives us the foundation for using random forest for fault diagnosis.

Note that, under the RTC framework, the size of the sliding window is an important parameter. When the window is too long, the method requires a large number of real-time data in each monitoring epoch, which can delay the identification of abnormal patterns. In contrast, if the window is too short, the classifiers built on the small data sets may be unstable and are prone to more false positives. In our R lab, we will explore this phenomenon further.

IV.3 R Lab

We have written up the RTC method into the R function, **Monitoring()**, as shown in below. The Monitoring function takes two datasets as input, the first one being the reference data, and the second one being the real-time data points. The window size also should be provided. The function returns a few monitoring statistics for each real-time data point, and the importance score of each variable.

```
library(dplyr)
library(tidyr)
library(randomForest)
library(ggplot2)

theme_set(theme_gray(base_size = 15) )

# define monitoring function. data0: reference data; data.real.time: real-time data; wsz: window size
Monitoring <- function( data0, data.real.time, wsz ){
  num.data.points <- nrow(data.real.time)
  stat.mat <- NULL
  importance.mat <- NULL

  for( i in 1:num.data.points ){
    # at the start of monitoring, when real-time data size is smaller than the window size, combine the real-time data points and random samples from the reference data to form a data set of wsz
    if(i<wsz){
      sample.size.from.reference <- wsz - i
      sample.reference <- data0[ sample(nrow(data0),sample.size.from.reference,replace = TRUE), ]
      current.real.time.data <- rbind( sample.reference, data.real.time[1:i,,drop=FALSE] )
    }else{
      current.real.time.data <- data.real.time[(i-wsz+1):i,,drop=FALSE]
    }
    current.real.time.data$class <- 1
    data <- rbind( data0, current.real.time.data )
    colnames(data) <- c(paste0("X",1:(ncol(data)-1)), "Class")
    data$Class <- as.factor(data$Class)

    # apply random forests to the data
    my.rf <- randomForest(Class ~.,sampsiz=c(wsz,wsz), data=data)

    # get importance score
  }
}
```

```

importance.mat <- rbind( importance.mat, t( my.rf$importance
) )
# get monitoring statistics
ooblist <- my.rf[5]
oobcolumn=matrix(c(ooblist[[1]]),2:3)
ooberrornormal= (oobcolumn[,3])[1]
ooberrorabnormal=(oobcolumn[,3])[2]

temp=my.rf[6]
p1vote <- mean( temp$votes[,2][ (nrow(data0)+1) : nrow(data) ]
)

this.stat <- c(ooberrornormal,ooberrorabnormal,p1vote)
stat.mat <- rbind(stat.mat, this.stat)
}
result <- list(importance.mat = importance.mat, stat.mat = sta
t.mat)
return(result)
}

```

First, let's consider a 2-dimesional data. The reference data follow a normal distribution with mean of 0 and standard deviation of 1. The real-time data come from two distributions. The first 100 data points have the same distribution as the reference data, while the second 100 data points have the second variable changed with mean of 2. Note that, here we label the reference data with class 0 and the real-time data with class 1.

```

# data generation
# sizes of reference data, real-time data without change, and rea
l-time data with changes
length0 <- 100
length1 <- 100
length2 <- 100

# 2-dimension
dimension <- 2

# reference data
data0 <- rnorm( dimension * length0, mean = 0, sd = 1)
# real-time data with no change
data1 <- rnorm( dimension * length2, mean = 0, sd = 1)
# real-time data different from the reference data in the second
the variable
data2 <- cbind( V1 = rnorm( 1 * length1, mean = 0, sd = 1), V2 =
rnorm( 1 * length1, mean = 2, sd = 1) )

```

```

# convert to data frame
data0 <- matrix(data0, nrow = length0, byrow = TRUE) %>% as.data.frame()
data1 <- matrix(data1, nrow = length2, byrow = TRUE) %>% as.data.frame()
data2 <- data2 %>% as.data.frame()

# assign variable names
colnames( data0 ) <- paste0("X",1:ncol(data0))
colnames( data1 ) <- paste0("X",1:ncol(data1))
colnames( data2 ) <- paste0("X",1:ncol(data2))

# assign reference data with class 0 and real-time data with class 1
data0 <- data0 %>% mutate(class = 0)
data1 <- data1 %>% mutate(class = 1)
data2 <- data2 %>% mutate(class = 1)

# real-time data consists of normal data and abnormal data
data.real.time <- rbind(data1,data2)

```

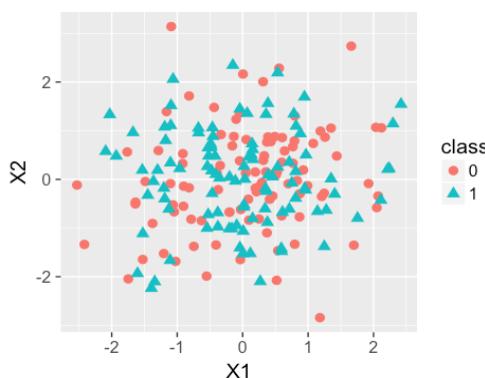


Figure 9.11: Scatterplot of the generated data points in the first 100 time points that come from the process under normal condition

Before we run the Monitor function, we show the scatterplot of the reference dataset and the dataset from the first 100 time points to obtain a visual sense of the data.

```
data.plot <- rbind( data0, data1 ) %>% mutate(class = factor(class))
ggplot(data.plot, aes(x=X1, y=X2, shape = class, color=class)) +
  geom_point(size=3)
```

Then we can obtain Figure 9.11. It can be seen that the two sets of data points are similar.

We also show the scatterplot of the reference dataset and the dataset from the second 100 time points.

```
data.plot <- rbind( data0, data2 ) %>% mutate(class = factor(class))
ggplot(data.plot, aes(x=X1, y=X2, shape = class, color=class)) +
  geom_point(size=3)
```

Then we can obtain Figure 9.12. It can be seen that for the real-time data set, X_2 has changed mean from 0 to 2.

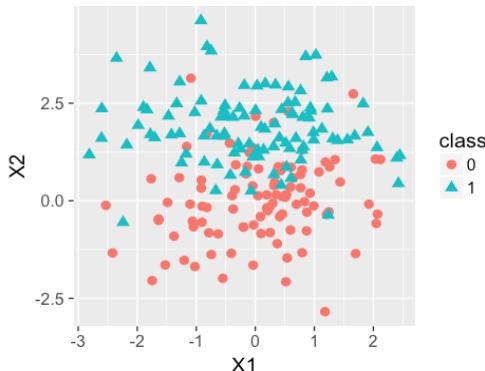


Figure 9.12: Scatterplot of the generated data points in the second 100 time points that come from the process under abnormal condition

Now we are ready to apply the RTC method. A window size of 10 is applied for monitoring. The error rates from the reference data, and the real-time data, and the probability estimates for the second class are shown in Figure 9.13 drew by the following R code.

```

wsz <- 10
result <- Monitoring( data0, data.real.time, wsz )
stat.mat <- result$stat.mat
importance.mat <- result$importance.mat

# plot different monitor statistics
stat.mat <- data.frame(stat.mat)
stat.mat$id <- 1:nrow(stat.mat)
colnames(stat.mat) <- c("error0", "error1", "prob", "id")
stat.mat <- stat.mat %>% gather(type, statistics, error0,error1,p
rob)
ggplot(stat.mat,aes(x=id,y=statistics,color=type)) + geom_line(lin
etyp
e = "dashed") + geom_point() + geom_point(size=2)

```

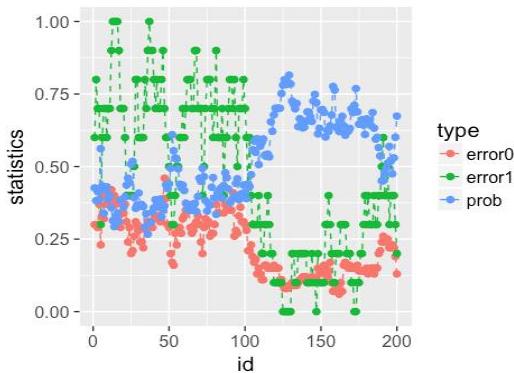


Figure 9.13: Chart of the monitoring statistics from time 1 to time 200.

Three monitoring statistics are shown: **error0** denotes for the error rate in Class 0, **error1** denotes for the error rate in Class 1, and **prob** denotes for the probability estimates of the data points

As we have known that a process shift happened on X2 after the 100th data point, a good monitor statistic should significantly signal the process change after the 100th data point, the sooner the better. As we can see, there is a slight decrease for the error rates from the reference dataset, but the decrease is not substantial. The probability estimates of the data points in the reference data have more obvious increase. Similar observation can be made for the error rates from the dataset captured by the sliding window. However, the error rates from the reference data jump among a small

number of distinct values. As mentioned earlier, the number of distinct values would further reduce with a smaller sliding window. Thus, this experiment confirms that the probability estimates lead to smoother monitoring statistic and have a significant change during the process transition phase.

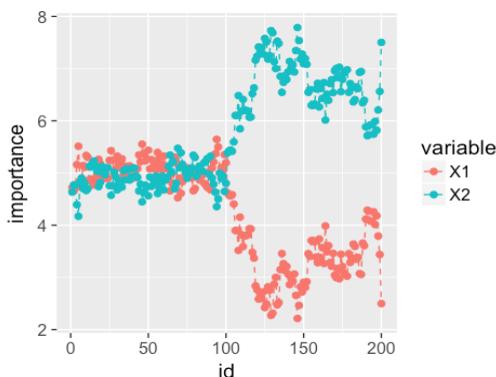


Figure 9.14: Chart of the importance score of the two process variables from time 1 to time 200

Next, let's consider fault diagnosis. Variable importance scores from the two variables from the random forests are shown in Figure 9.14 drew by the following R code.

```
# plot importance scores for diagnosis
importance.mat <- data.frame(importance.mat)
importance.mat$id <- 1:nrow(importance.mat)
colnames(importance.mat) <- c("X1", "X2", "id")
importance.mat <- importance.mat %>% gather(variable, importance, X1, X2)

ggplot(importance.mat, aes(x=id, y=importance, color=variable)) + geom_line(linetype = "dashed") + geom_point(size=2)
```

From Figure 9.14, we can see that the importance scores of X2 significantly increase after the 100th point. This indicates that X2 plays an

important role in improving the classification and may be responsible for the process change.

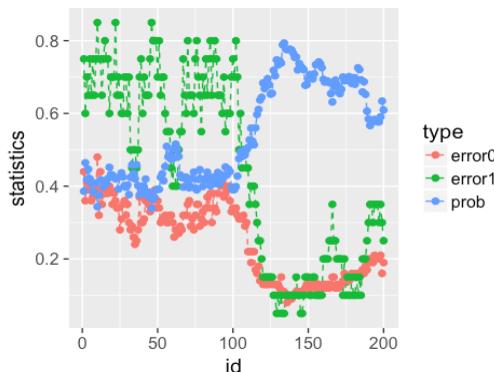


Figure 9.15: Chart of the monitoring statistics from time 1 to time 200
(window size increases to 20)

As we have mentioned, the size of the window for monitoring is an important parameter. Here, to see its effect, the window size is increased to 20. The monitoring statistics and importance scores are re-plotted in Figure 9.15 and Figure 9.16.

```
# change window size to 20
wsz <- 20
result <- Monitoring( data0, data.real.time, wsz )
stat.mat <- result$stat.mat
importance.mat <- result$importance.mat

# plot different monitor statistics
stat.mat <- data.frame(stat.mat)
stat.mat$id <- 1:nrow(stat.mat)
colnames(stat.mat) <- c("error0","error1","prob","id")
stat.mat <- stat.mat %>% gather(type, statistics, error0,error1,p
rob)
ggplot(stat.mat,aes(x=id,y=statistics,color=type))+ geom_line(lin
etype = "dashed") + geom_point() + geom_point(size=2)
```

```
# plot importance scores for diagnosis
importance.mat <- data.frame(importance.mat)
importance.mat$id <- 1:nrow(importance.mat)
colnames(importance.mat) <- c("X1", "X2", "id")
importance.mat <- importance.mat %>% gather(variable, importance, X1, X2)

ggplot(importance.mat, aes(x=id, y=importance, color=variable)) + geom_line(linetype = "dashed") + geom_point(size=2)
```

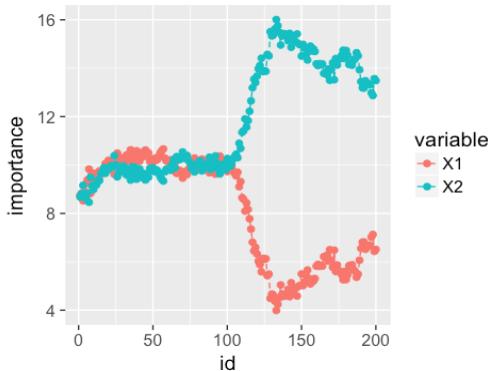


Figure 9.16: Chart of the importance score of the two process variables from time 1 to time 200 (window size increases to 20)

Compared to the previous results with window size of 10, the monitoring statistics on the changed real-time data points have a clearer increase from the un-changed real-time data. Similarly, the increase of the importance score of X2 is stronger. However, the change of the monitoring statistics and importance scores is slightly slower than the change with a smaller window. Therefore, a large window size can lead to more confident alert, but with a slower speed.

Now, let's change the window size to 5. The monitoring statistics and importance scores are re-plotted in Figure 9.17 and Figure 9.18.

```

# change window size to 5
wsz <- 5
result <- Monitoring( data0, data.real.time, wsz )
stat.mat <- result$stat.mat
importance.mat <- result$importance.mat

# plot different monitor statistics
stat.mat <- data.frame(stat.mat)
stat.mat$id <- 1:nrow(stat.mat)
colnames(stat.mat) <- c("error0","error1","prob","id")
stat.mat <- stat.mat %>% gather(type, statistics, error0,error1,p
rob)
ggplot(stat.mat,aes(x=id,y=statistics,color=type))+ geom_line(lin
etype = "dashed") + geom_point() + geom_point(size=2)

```

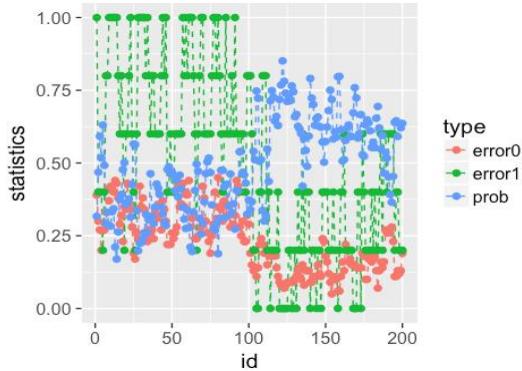


Figure 9.17: Chart of the monitoring statistics from time 1 to time 200
(window size decreases to 5)

```

# plot importance scores for diagnosis
importance.mat <- data.frame(importance.mat)
importance.mat$id <- 1:nrow(importance.mat)
colnames(importance.mat) <- c("X1","X2","id")
importance.mat <- importance.mat %>% gather(variable, importance,
X1,X2)

ggplot(importance.mat,aes(x=id,y=importance,color=variable)) + ge
om_line(linetype = "dashed") + geom_point(size=2)

```

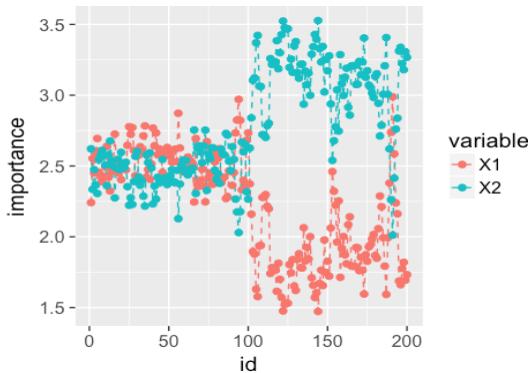


Figure 9.18: Chart of the importance score of the two process variables from time 1 to time 200 (window size decreases to 5)

Obviously, the monitoring statistic seems to be more noisy, producing less confident patterns, but it raises quicker alert at the 101th time point.

Now, let's consider a 10-dimensional dataset. In this example, two variables' means change from 0 to 2 in the second 100 data points.

```
# 10-dimensions, with 2 variables being changed from the normal condition
dimension <- 10
wsz <- 5
# reference data
data0 <- rnorm( dimension * length0, mean = 0, sd = 1)
# real-time data with no change
data1 <- rnorm( dimension * length1, mean = 0, sd = 1)
# real-time data different from the reference data in the second the variable
data2 <- c( rnorm( (dimension - 2) * length2, mean = 0, sd = 1),
          rnorm( (2) * length2, mean = 20, sd = 1))

# convert to data frame
data0 <- matrix(data0, nrow = length0, byrow = TRUE) %>% as.data.frame()
data1 <- matrix(data1, nrow = length1, byrow = TRUE) %>% as.data.frame()
data2 <- matrix(data2, ncol = 10, byrow = FALSE) %>% as.data.frame()
```

```

# assign reference data with class 0 and real-time data with class 1
data0 <- data0 %>% mutate(class = 0)
data1 <- data1 %>% mutate(class = 1)
data2 <- data2 %>% mutate(class = 1)

# real-time data consists of normal data and abnormal data
data.real.time <- rbind(data1,data2)

```

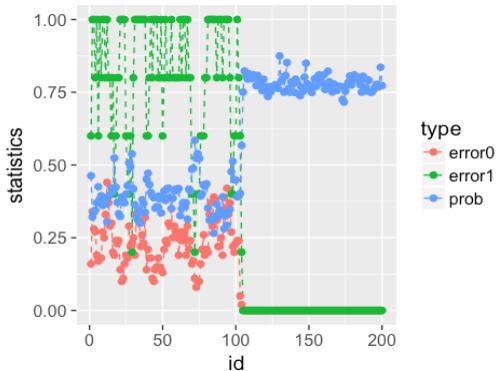


Figure 9.19: Chart of the monitoring statistics from time 1 to time 200
(window size is 10)

10 dimensions are difficult to visualize and monitor. Encouragingly, the monitoring statistics shown in Figure 9.19 shows that the RTC method is still capable of capturing the changes. It is clear in Figure 9.19 that all the monitoring statistics change after the 101th time point, and the importance scores in Figure 9.20 also indicate the change is due to X9 and X10. The following R codes generated Figure 9.19.

```

result <- Monitoring( data0, data.real.time, wsz )
stat.mat <- result$stat.mat
importance.mat <- result$importance.mat

# plot different monitor statistics
stat.mat <- data.frame(stat.mat)
stat.mat$id <- 1:nrow(stat.mat)

```

```
colnames(stat.mat) <- c("error0","error1","prob","id")
stat.mat <- stat.mat %>% gather(type, statistics, error0,error1,p
rob)
ggplot(stat.mat,aes(x=id,y=statistics,color=type))+ geom_line(lin
etype = "dashed") + geom_point() + geom_point(size=2)
```

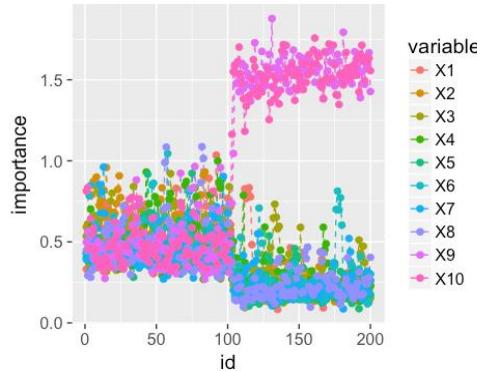


Figure 9.20: Chart of the importance score of the ten process variables from time 1 to time 200 (window size is 10)

The following R codes generated Figure 9.20.

```
# plot importance scores for diagnosis
importance.mat <- data.frame(importance.mat)
importance.mat$id <- 1:nrow(importance.mat)
# colnames(importance.mat) <- c("X1", "X2", "id")
importance.mat <- importance.mat %>% gather(variable, importance,
X1:X10)
importance.mat$variable <- factor( importance.mat$variable, level
s = paste0( "X", 1:10 ) )
# Levels(importance.mat$variable) <- paste0( "X", 1:10 )
ggplot(importance.mat,aes(x=id,y=importance,color=variable)) + ge
om_line(linetype = "dashed") + geom_point(size=2)
```

IV.4 Remarks

There is an important technical aspect to implement the RTC method. Realistically, to capture the normal conditions of a process, many data

points are needed to define the reference dataset. On the other hand, to capture process changes more sensitively, the window size for online monitoring should not be too large. Thus, comparing with the sample size in the reference dataset, an effective window size is typically substantially smaller than the size of reference data. Therefore, this usually leads to a highly imbalanced classification problem.

As a remedy, the random forest model can handle class imbalance by under-sampling the reference data to be the same size as the sliding window data for each tree. That is to say, instead of sampling uniformly the data points for each tree (as shown in the left figure in Figure 9.21), the same number of samples are selected from the reference data and the sliding window data (as shown in the right figure in Figure 9.21).

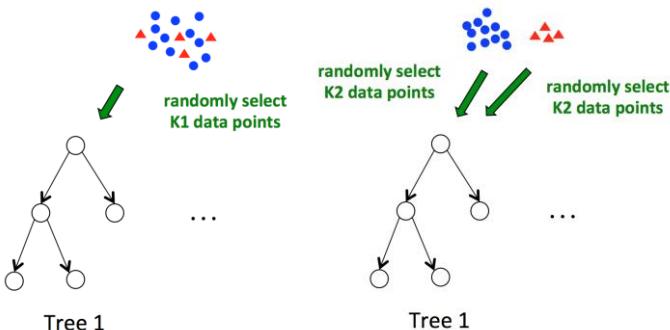


Figure 9.21: Regular sampling (left) and purposeful equal sampling (right) by random forest to grow its trees

IV. Exercises

Data analysis

1. Find 10 regression datasets from the UCI data repository or R datasets. Build kernel regression model, KNN regression model, and linear regression models. Conduct model selection and validation. Use cross-validation to select the best models. Compare the models and comment on their applications on these datasets.

2. Identify 3 datasets from the UCI data repository or R datasets that the heteroscedasticity may be a problem. Build the conditional variance regression model on these datasets. Compare the conditional variance regression model with linear regression model.

Programming

3. Use bootstrap to show the conditional variance regression model is significantly better than linear regression model on the datasets you have selected that have the problem of heteroscedasticity. Write your own R script to implement this idea. Make sure that, for datasets that have no concern of heteroscedasticity, your approach would not always advocate for the use of conditional variance regression model.
4. Implement the tree-based system monitoring method on a high-dimensional dataset with more than 100 variables. You can simulate such a dataset following the R lab in this chapter. See if the tree-based system monitoring method can lead to quick detection of process change, and accurate fault diagnosis (i.e., make sure in your data, only a few variables are responsible for the process change).

CHAPTER 10: SYNTHESIS

INTREES/PIPELINE ENGINEERING

I. Overview

Chapter 9 is about “**synthesis**”. It recognizes the practical dimension of solving real-world problems, such as building pipelines that combine and streamline a variety of models and operations. This is nonetheless a diverse topic and anticipates many possible alternatives, depending on the particular problems and contexts. To give one example, here, we introduce the method implemented in the R package, InTrees¹, which combines decision tree, random forest, and feature selection such as LASSO. Note that, as an overarching framework, its combinatory power is not limited with these models.

¹ Deng, H. *Interpreting tree ensembles with inTrees*. Manuscript available at: <https://arxiv.org/abs/1408.5456>

II. InTrees

II.1 Rationale and Formulation

As we have seen that, the linear regression models are advantageous in providing a linear characterization of a multivariate system. Although its ostensible interpretability comes with a question mark, it is undoubtable a more interpretable model than the tree models. Tree models are more like approximating the data and build on their power in capturing complex interactions between variables. Given their different advantages, it is natural to wonder if we could combine both models and get the best of both. From a pragmatic perspective, as Prof. George Box has pointed out, “*all models are wrong, some are useful*”, neither model is the truth, but we can always make the model better in terms of some quantitative criterion such as prediction accuracy on testing data or qualitative criterion such as interpretability.

To combine the two models, first, we may notice that the two models employ different semantics: the regression model uses an equation-based semantics, that is more mathematically and formative; the tree model uses a rule-based semantics, which is more intuitive and heuristic. Can we have a hybrid model that combines both semantics? Sure, we could. One approach we can use is to plug in one semantics into another.

For example, while regression model puts variables into the equation, the variables are defined by users. So, what are the variables?

This is the starting point of InTrees. It takes rules as the variables that can be put into a regression model. To get the rules, InTrees harvests the power of random forest to convert the original raw data into a new dataset whose variables are the rules. As we know, rules represent complex interactions between variables. In this way, we have the best parts of both methods, while the rules capture the variable-level patterns in the data, and the regression equation captures the global effects of these patterns in predicting the outcome variable.

II.2 Theory and Method

Consider the following dataset that has 2 predictors and 7 instances as shown in Table 10.1.

Table 10.1: An exemplary dataset with 7 instances

ID	X_1	X_2	Class
1	1	1	C0
2	1	0	C1
3	0	1	C1
4	0	0	C1
5	0	0	C0
6	0	0	C0
7	0	0	C0

Importance scores from the random forest model can provide insights regarding which variables are important, however, it is still not straightforward to understand. For example, the importance scores from random forests applied to the dataset in Table 10.1 are shown in Figure 10.1. We can only know that X_1 and X_2 have similar importance scores, but it is unclear how exactly these variables work together to predict the class.

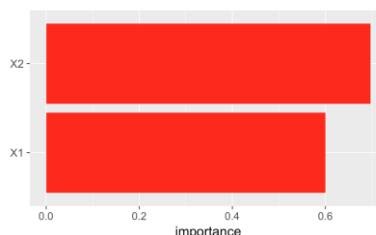


Figure 10.1: Importance score of two variables

To this end, the inTrees framework, illustrated in Figure 10.2, was proposed to extract, measure, prune, and select rules from a tree ensemble. It will further calculate frequent variable interactions, and summarize rules

into a prediction model (rules become the variables). The framework has been implemented in the ```inTrees`'' R package. In the following we introduce each functionality of the `inTrees` framework.

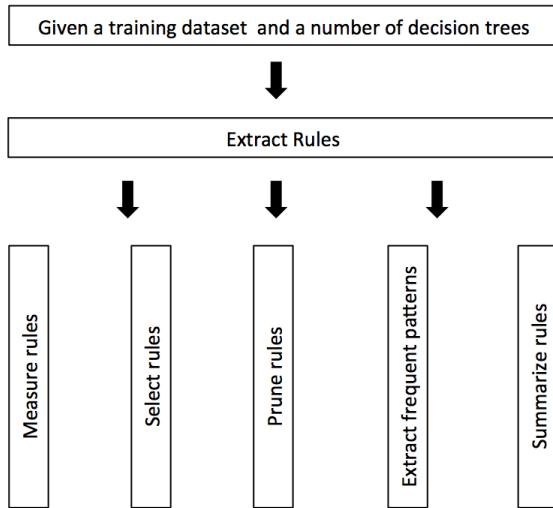


Figure 10.2: The pipeline of `inTrees`

Rule extraction and measuring: A decision tree can be dissembled into a set of rules. For example, considering the decision tree shown in Figure 10.3, which was created as one tree of the random forest model applied on the dataset shown in Table 1. It can be seen that the tree was built based on the resampled dataset that includes the instances $\{1,1,2,2,7,7,7\}$. In other words, in this resampled dataset, the instance (ID: 1) was resampled twice, the instance (ID: 2) was resampled twice, and the instance (ID: 7) was resampled three times. In the tree, the root and inner nodes are labeled with the data point IDs and leaf nodes are labeled with the data point IDs and the decisions (i.e., which class to predict). Here, three rules can be extracted: $\{X_1 = 0 \rightarrow \text{Class} = C_1\}$, $\{X_1! = 0, X_2 = 0 \rightarrow \text{Class} = C_1\}$, $\{X_1! = 0, X_2! = 0 \rightarrow \text{Class} = C_0\}$.

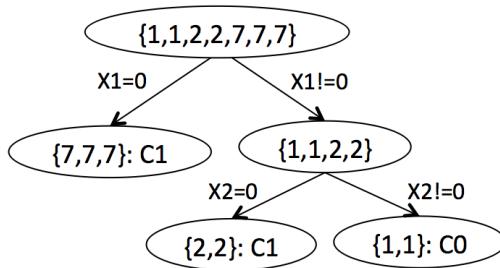


Figure 10.3: An exemplary decision tree

Generally, in a decision tree, a rule can be extracted from the root node to each leaf node as $\{X_{i_1} = a_{i_1}, \dots, X_{i_k} = a_{i_k} \rightarrow T = t\}$, where X_{i_j} represents the variable used in the path from the root node to a leaf node, $X_{i_j} = a_{i_j}$ represents the criterion for splitting at node i_j , and $T = t$ is the outcome at the leaf node. Note a_{i_j} can be a range when X_{i_j} is numerical and a set of values when it is categorical.

Thus, for a random forest model, we can extract a set of rules by dissembling all its trees. There is one complication, though, that in random forest, as we have mentioned in Chapter 4, each tree is built on a subset of samples and a subset of features in order to be a weak model. That means, we need to revise the outcome of each rule that maybe different from the original outcome associated with this rule provided by the tree. In the inTrees framework, the outcomes from the original rules are ignored. Instead, the outcomes are re-calculated using all the training data, such that the most frequent class is used as the outcome.

For example, for the rule $\{X_1 = 0 \rightarrow \text{Class} = C_1\}$ derived from the decision tree shown in Figure 10.3, the three data points (the same instance) have the class of C_1 . However, when using all the raining data, there are five data points (IDs: 3-7) satisfying $X_1 = 0$, and 3/5 of the data points have

class of C_0 . Therefore, the rule should be updated with the outcome as C_0 . The other two rules remain the same since the classes from the tree are consistent with the most frequent classes when applied to all the training data.

After we collect all the rules from the random forest model, the rules are evaluated with three criteria. The **length** of a rule is defined as the number of variable-value pairs in the rule condition. The **frequency** of a rule is the proportion of data points satisfying the rule condition, the left-hand part of the rule. The **error** is the error rate of the rule. For classification problems, it is the number of data points incorrectly identified by the rule divided by the number of data points satisfying the condition. For regression problems, it is mean squared error defined as:

$$err = \frac{1}{k} \sum_{i=1}^k (t_i - \tilde{t})^2,$$

where k is the number of data points in the leaf node, t_i is the value of the response variable of the i^{th} data point, and \tilde{t} is the prediction at the leaf node.

For the illustrative example and decision tree, the evaluation of the three rules are shown in the Table 10.2.

Table 10.2: Evaluation of the three rules extracted from the tree shown in Figure 10.3

ID	Rule	length	frequency	error
1	$\{X_1 = 0 \rightarrow Class = C_0\}$	1	5/7	2/5
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$	2	1/7	0
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$	1	1/7	0

Prune rules: As each tree in a tree ensemble can be weak, the rules we collect can include irrelevant and redundant variable-value pairs. Therefore, it may be beneficial to remove/prune irrelevant and redundant rules.

Let p_i denote the i^{th} variable-value pair of a rule condition, such that a rule can be written as $\langle p_1, \dots, p_k \rangle \rightarrow T = t$. To prune the rule, inTrees uses leave-one-out pruning, that is, at each round, removes one pair and checks how much error this removal will induce. The pair with the least error increase is removed, if it is also below a pre-specified threshold. The increase of error is referred as the **decay** in the terminology of inTrees.

Two types of decay are defined. The first one is the absolute error increase, defined as:

$$\text{decay}_i = \text{Err}_{-i} - \text{Err}.$$

The second one is the relative error increase, defined as:

$$\text{decay}_i = \frac{\text{Err}_{-i} - \text{Err}}{\max(\text{Err}, s)},$$

where Err is the error of the original rule, Err_{-i} is the error of the rule with the p_i removed, and s is a small positive constant (e.g., 0.001) that bounds the value of decay when Err is zero or close to zero.

Take rule $\{X_1! = 0, X_2 = 0 \rightarrow \text{Class} = C_1\}$ for example. The error for this rule is known to be 0 if we check the illustrative dataset aforementioned. Assume that the threshold is 0.05. Now remove $X_1! = 0$ from the rule condition, and the new rule becomes $\{X_2 = 0 \rightarrow \text{Class} = C_1\}$, which has an error of 3/5. Therefore, the absolute error increase is 3/5. Therefore, $X_1! = 0$ should not be pruned.

On the other hand, we can also see that the relative error increase is $\frac{3}{5*s}$. With a default value as $s = 0.001$, the relative error increase is also large, indicating that $X_1! = 0$ should not be pruned.

Now let's remove $X_2 = 0$. The resulting rule $\{X_1 \neq 0 \rightarrow \text{Class} = C_1\}$ has an error of 1/2. Therefore, removing $X_2 = 0$ also hurts the accuracy of the rule. The rule should not be pruned.

Let's do another example. Suppose that the rules are built with the following data set shown in Table 10.3.

Table 10.3: An exemplary dataset

ID	X_1	X_2	Class
1	1	0	C_1
2	1	0	C_1
3	1	1	C_0
4	0	1	C_0

On this dataset, the rule $\{X_1 \neq 0, X_2 = 0 \rightarrow \text{Class} = C_1\}$ has an error of 0. The error after removing $X_1 \neq 0$ is still 0, therefore, both the absolute and relative error increase are 0. The error after removing $X_2 = 0$ becomes 1/4. Therefore, the absolute error increase is 0.25, and the relative error increase is $\frac{1}{4*5}$. Thus, $X_1 = 0$ should be removed, and the new pruned rule becomes $\{X_2 = 0 \rightarrow \text{Class} = C_1\}$. This pruning process will continue. After removing $X_2 = 0$ from the rule, the rule is effectively a random guess, thus the error becomes 0.5. This indicates that no variable-value pair can be removed, the pruning should be stopped, and the final rule is $\{X_2 = 0 \rightarrow \text{Class} = C_1\}$.

Select rules: In previous sections, each rule is pruned and the pruned rules can be ranked by accuracy, frequency and complexity. However, a tree ensemble can generate numerous rules, and many of them can be redundant. If the top rules tend to be redundant rules, the ranked rule set is

again hard to interpret. Therefore, selecting a non-redundant rule set is valuable for interpretation.

The inTrees framework casts the rule selection problem into the feature selection formulation. This is built on the creation of a new dataset based on the rule set that binarizes the original dataset. For example, continue our discussion of the dataset with 7 instances as shown in Table 10.1, the three rules we have collected are shown in Table 10.4.

Table 10.4: The three rules

ID	Rule
1	$\{X_1 = 0 \rightarrow Class = C_0\}$
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$

Table 10.5: The binarized dataset of Table 10.1 by the rules in Table 10.4

ID	Z ₁	Z ₂	Z ₃	Class
1	0	0	1	C0
2	0	1	0	C1
3	1	0	0	C1
4	1	0	0	C1
5	1	0	0	C0
6	1	0	0	C0
7	1	0	0	C0

Consider $\{X_1 = 0 \rightarrow Class = C_0\}$ first. In the new dataset, a binary feature Z_1 is created for the condition. The instances satisfying the condition $X_1 = 0$ include $\{3,4,5,6,7\}$ and therefore, the binary feature values of the instances are $\{0,0,1,1,1,1\}$. For $\{X_1! = 0, X_2 = 0 \rightarrow$

$\text{Class} = C_1\}$, only instance 2 satisfies the condition, and therefore, the binary feature values of the instances for the second rule is $\{0,1,0,0,0,0,0\}$. Similarly, the feature values for the third rule is $\{1,0,0,0,0,0,0\}$. Following this process, the new converted data set is shown in Table 10.5.

Now a feature selection method can be applied on this new data set. The goal of feature selection is to select a subset of relevant but not redundant features from the original features. Feature selection methods include L_1 regularized logistics regression (or LASSO) and regularized random forests are used in the inTrees R package. Suppose a feature selection method selects the feature subset as $\{Z_1, Z_2\}$. It indicates that only the first and second rules should be used in prediction.

Note that the binary feature representation does not include the information about the length of rules. Given two rules with the same predictive power, the rule with a smaller length may be preferred in terms of interpretability. In the inTrees framework, the guided regularized random forests are used. The guided regularized random forests (GRRF) can assign a weight to each feature, so that when two features have similar predictive power, the feature with more weight is more likely to be selected. In this case, shorter rules are more likely to be selected.

Frequent variable interaction: The inTrees framework further provides extraction of variable interactions that have been important in the selected rules. A rule essentially encodes these interaction information, e.g., the rule $\{X_1 \neq 0, X_2 = 0 \rightarrow \text{Class} = C_1\}$ captures the interaction between X_1 and X_2 .

Consider the following set of rules extracted by inTrees. Association rule analysis is used for mining the frequent variable-value pairs from the rules. In particular, each variable-value pair is considered as an item.

First, let's consider all the rule conditions (left-hand side of the rules). Define the **support** of a particular interaction pattern (e.g., X_1 and X_2) as

the number of rules that encode this interaction in the rule set, divided by the size of the rule set (size = 4 for the illustrative example). For example, the support of each variable interaction pattern extracted from the rules in Table 10.6 is calculated in Table 10.7.

Table 10.6: An exemplary set of rules

ID	Rule
1	$\{X_1 = 0 \rightarrow \text{Class} = C_0\}$
2	$\{X_1! = 0, X_2 = 0 \rightarrow \text{Class} = C_1\}$
3	$\{X_1! = 0, X_2! = 0 \rightarrow \text{Class} = C_0\}$
4	$\{X_1! = 0, X_2 = 0, X_3 = 1 \rightarrow \text{Class} = C_1\}$

Table 10.7: Interaction patterns extracted from the rules in Table 5 and their supports

Variable-value pairs (interaction patterns)	Support
$X_1 = 0$	1/4
$X_1! = 0$	3/4
$X_2 = 0$	1/4
$X_3 = 1$	1/4
$X_1! = 0, X_2 = 0$	2/4
$X_1! = 0, X_2! = 0$	1/4
$X_1! = 0, X_2 = 0, X_3 = 1$	1/4

Hence, the most frequent interactions are $X_1! = 0$ and $X_1! = 0, X_2 = 0$. Now consider the right hand of each interaction. The **confidence** is defined as the accuracy of an interaction pattern predicting a particular class. For example, continuing the example mentioned above, the confidence of the interactions can be calculated as shown in Table 10.8.

Table 10.8: Interaction patterns extracted from the rules in Table 10.6 and their confidences

Variable-value pairs (interaction patterns)	Class	Confidence
$X_1 = 0$	$Class = C_0$	1/1
$X_1 \neq 0$	$Class = C_1$	2/3
$X_2 = 0$	$Class = C_1$	1/1
$X_3 = 1$	$Class = C_1$	1/4
$X_1 \neq 0, X_2 = 0$	$Class = C_1$	2/2
$X_1 \neq 0, X_2 \neq 0$	$Class = C_0$	1/1
$X_1 \neq 0, X_2 = 0, X_3 = 1$	$Class = C_1$	1/1

Combine both Tables, we can see the top variable interactions in terms of confidence and support is

$$\{X_1 \neq 0, X_2 = 0 \rightarrow Class = C_1\}.$$

with a support of 0.5 and confidence of 1. This indicates this variable interaction plays an important role in the tree ensemble.

Note that, for continuous features, it may be useful to do a discretization before inputting the data into inTrees. This is because that there can be many possible splitting points for continuous features in a tree, resulting in the possibility that fewer frequent variable interactions for continuous features could be identified.

Summarize rules: Once rules from tree ensembles are pruned and selected, we can summarize these high-quality rules into classifiers. There are multiple methods for summarizing. For example, the method RuleFit¹ used a linear model for summarizing the rules. Here, we introduce a simple method to summarize the rules into an ordered rule set for prediction.

¹ Friedman, J.H. and Popescu, B.E. Predictive learning via rule ensembles. *Annals of applied statistics*, 2008.

The method has multiple iterations. Denote r_0 as the default rule (i.e., with null condition) that classifies all data points to the most frequent class. Denote the ordered rule set as R , which is set to be empty at the beginning. Then, at each iteration, the best rule in the rule set is selected and added to R . The best rule is defined as the rule with the minimum error evaluated by the training data. If there are ties, the rule with higher frequency and smaller length is selected. Then, the data points that satisfy the condition of the best rule are removed, and the default rule r_0 is re-calculated with the data points left. This iterative process continues until no instance is left in the training dataset, or the default rule r_0 has the best accuracy comparing with other rules in the remaining rule set. Note that, the selected rules in R are ordered according to the sequence of their inclusion.

Consider the dataset shown in Table 1 and the rule set shown in Table 4. At the beginning, the default rule is $\{Class = C_0\}$ with error rate of 3/7. The error rate and frequency of each rule is shown in Table 10.9.

Table 10.9: Error rates and frequencies of the rules in Table 10.5 using dataset in Table 10.1

ID	Rule	Error	Frequency
1	$\{X_1 = 0 \rightarrow Class = C_0\}$	2/5	5/7
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$	0/1	1/7
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$	0/1	1/7
4	$Class = C_0$	3/7	

In this case, Rule 1 and Rule 2 have the least errors, and their frequency and length are also the same. Thus, we can select either of them to the ordered rule set $R = \{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$. Then, the data point (ID:2) classified by this rule is removed. The default rule is still

$\{Class = C_0\}$, and the error and frequency of each rule on the new dataset is updated in Table 10.10.

Table 10.10: Updated error rates and frequencies of the rules in Table 10.5 using the reduced dataset in Table 10.1 (data point ID:2 is removed)

ID	Rule	Error	Frequency
1	$\{X_1 = 0 \rightarrow Class = C_0\}$	2/5	5/6
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$	NA	0/6
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$	0/1	1/6
4	$Class = C_0$	2/6	

Table 10.11: Updated error rates and frequencies of the rules in Table 4 using the reduced dataset in Table 1 (data points ID:1 and ID:2 are removed)

ID	Rule	Error	Frequency
1	$\{X_1 = 0 \rightarrow Class = C_0\}$	2/5	5/5
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$	NA	0/5
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$	NA	0/5
4	$Class = C_0$	2/5	

Then, $\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$ is added to R , and the data point (ID:1) is removed. The default rule remains unchanged and the error and frequency of each rule on the new dataset is updated in Table 10.11.

Now the default rule $Class = C_0$ has the minimum error 2/5, the same as $\{X_1 = 0 \rightarrow Class = C_0\}$. Therefore, the default rule is added to R and the process stops. The final ordered rule set R is summarized in Table 10.12.

Table 10.12: Final results of R

Order	Rule
1	$\{X_1 \neq 0, X_2 = 0 \rightarrow \text{Class} = C_1\}$
2	$\{X_1 \neq 0, X_2 \neq 0 \rightarrow \text{Class} = C_0\}$
3	$\text{Class} = C_0$

When predicting on an instance, the first rule in R satisfying the data point is used for prediction. For example, for a data point $\{X_1 = 0, X_2 = 1\}$, it does not satisfy neither Rule 1 or Rule 2 in R . Therefore, the default rule is used, and the prediction is C_0 .

II.3 R Lab

Here we apply random forests to the AD dataset and use `inTrees` to extract rules. First, based on the random forest model, 4555 rules are extracted.

```
rm(list = ls(all = TRUE))
library("arules")
library("randomForest")
library("RRF")
library("inTrees")
library("reshape")
library("ggplot2")
set.seed(1)
path <- "../../data/AD_b1.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_b1")
target <- paste0("class_", as.character(data[, target_indx]))
rm_indx <- which(colnames(data) %in% c("DX_b1", "ID", "TOTAL13",
"MMSCORE"))
X <- data
X <- X[, -rm_indx]
for (i in 1:ncol(X)) X[, i] <- as.factor(dicreteizeVector(X[, i],
K = 3))

rf <- randomForest(X, as.factor(target))

treeList <- RF2List(rf) # transform rf object to an inTrees' for
```

```
mat
exec <- extractRules(treeList, X) # R-executable conditions

## 4555 rules (length<=6) were extracted from the first 100 trees.
```

Next, the rules are measured by length, error and frequency. E.g., the statistics of 5 rules are shown in the graph below.

```
class <- paste0("class_", as.character(target))
rules <- getRuleMetric(exec, X, target)
print(rules[order(as.numeric(rules[, "len"])), ][1:5, ])

##      len freq    err      condition
## [1,] "2" "0.118" "0.098" "X[,6] %in% c('L1') & X[,11] %in% c('L1')"
## [2,] "2" "0.182" "0"      "X[,4] %in% c('L1') & X[,6] %in% c('L1')"
## [3,] "2" "0.182" "0"      "X[,4] %in% c('L1') & X[,6] %in% c('L1')"
## [4,] "2" "0.081" "0.024" "X[,3] %in% c('L3') & X[,4] %in% c('L3')"
## [5,] "2" "0.043" "0.136" "X[,6] %in% c('L3') & X[,7] %in% c('L3')"
##      pred
## [1,] "class_1"
## [2,] "class_1"
## [3,] "class_1"
## [4,] "class_0"
## [5,] "class_0"
```

For rule pruning, first, we try with absolute decay using a threshold of 0.005 (`maxDecay = 0.005`) to prune the rules, that is, a variable-pair is not removed if the decay is larger than 0.005. The statistics of the rules before and after pruning are shown in Figures 4-6.

The R code below generates Figure 10.4.

```
rules.pruned <- pruneRule(rules, X, target, maxDecay = 0.005, type = "absolute")
length <- data.frame(original = as.numeric(rules[, "len"]),
                      pruned = as.numeric(rules.pruned[, "len"]))
ggplot(melt(length), aes(value, fill = variable)) + geom_histogram(position = "dodge",
                                                               binwidth = 0.4) + ggtitle("Histogram of Lengths") + theme(plot.title = element_text(hjust = 0.5))
```

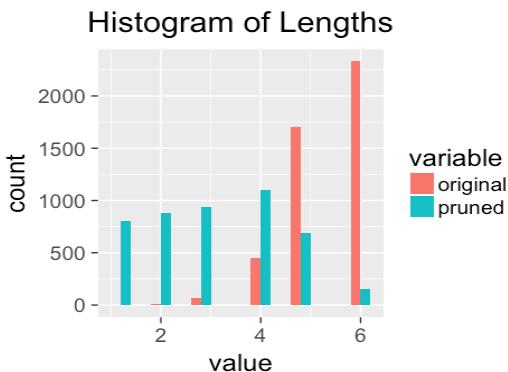


Figure 10.4: Histogram of lengths of the rules before and after the pruning

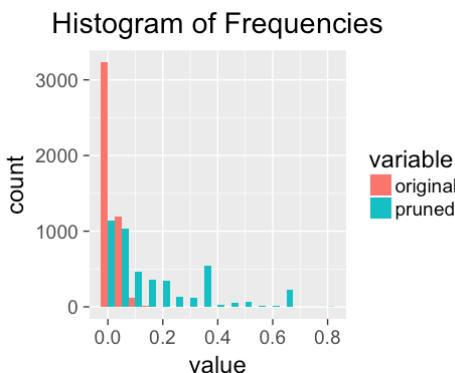


Figure 10.5: Histogram of frequencies of the rules before and after the pruning

The R code below generates Figure 10.5.

```
frequency <- data.frame(original = as.numeric(rules[, "freq"]),
  pruned = as.numeric(rules.pruned[, "freq"]))
ggplot(melt(frequency), aes(value, fill = variable)) + geom_histogram(position = "dodge",
  binwidth = 0.05) + ggtitle("Histogram of Frequencies") + theme(plot.title = element_text(hjust = 0.5))
```

The R code below generates Figure 10.6.

```
error <- data.frame(original = as.numeric(rules[, "err"]),
pruned = as.numeric(rules.pruned[, "err"]))
ggplot(melt(error), aes(value, fill = variable)) + geom_histogram(position = "dodge",
binwidth = 0.01) + ggtitle("Histogram of Errors") + theme(plot.title = element_text(hjust = 0.5))
```

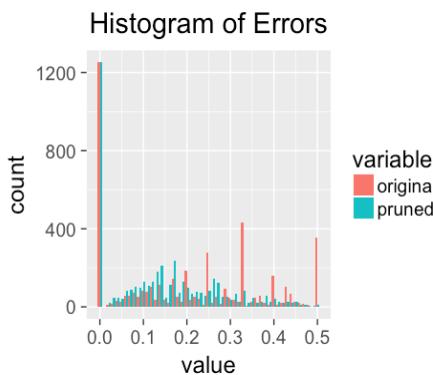


Figure 10.6: Histogram of errors of the rules before and after the pruning

It can be seen that, the lengths of rules are substantially reduced. For example, a majority of the original rules have length of 6 (as the default max length is set to be 6), while after pruning, only a slight percentage of the rules have length of 6. Also, since rules are shortened, the reduction of frequencies are also significant. In terms of errors, after pruning, the error distribution has shifted to the left. Therefore, the rules are simplified without significant sacrifice of accuracy.

For a comparison, we conduct one more experiment using the relative decay with a threshold of 0.05 (`maxDecay = 0.05`). Results are shown in Figures 7-9.

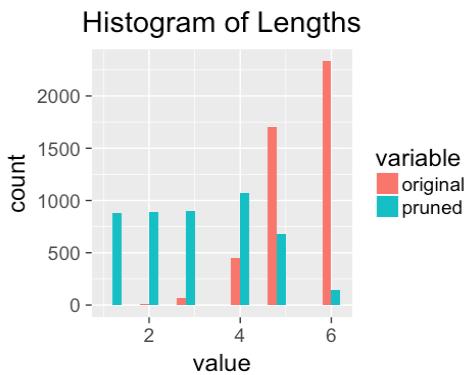


Figure 10.7: Histogram of lengths of the rules before and after the pruning

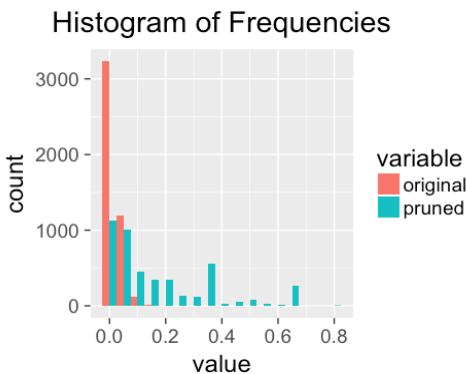


Figure 10.8: Histogram of frequencies of the rules before and after the pruning

The R code below generates Figure 10.7.

```
rules.pruned <- pruneRule(rules, X, target, maxDecay = 0.05, type
Decay = 1)

length <- data.frame(original = as.numeric(rules[, "len"]),
prune
d = as.numeric(rules.pruned[, "len"]))
ggplot(melt(length), aes(value, fill = variable)) + geom_histogra
m(position = "dodge",
```

```
binwidth = 0.4) + ggtitle("Histogram of Lengths") + theme(plot.title = element_text(hjust = 0.5))
```

The R code below generates Figure 10.8.

```
frequency <- data.frame(original = as.numeric(rules[, "freq"]),
                           pruned = as.numeric(rules.pruned[, "freq"]))
ggplot(melt(frequency), aes(value, fill = variable)) + geom_histogram(position = "dodge",
                           binwidth = 0.05) + ggtitle("Histogram of Frequencies") + theme(plot.title = element_text(hjust = 0.5))
```

The R code below generates Figure 10.9.

```
error <- data.frame(original = as.numeric(rules[, "err"]),
                      pruned = as.numeric(rules.pruned[, "err"]))
ggplot(melt(error), aes(value, fill = variable)) + geom_histogram(position = "dodge",
                           binwidth = 0.01) + ggtitle("Histogram of Errors") + theme(plot.title = element_text(hjust = 0.5))
```

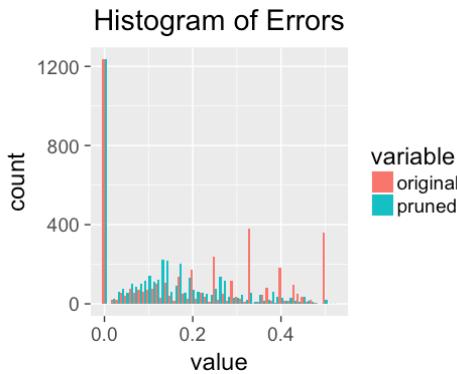


Figure 10.9: Histogram of errors of the rules before and after the pruning

The changes of lengths, frequencies and errors, look similar to the previous results using the absolute decay. An advantage of using relative decay is that one does not need to know the baseline error of a dataset. However, relative decay depends on the baseline error of each original rule,

and when the baseline error is small, e.g., 0, the relative error increase can be large even if the absolute error increase is small.

Now let's consider rule selection. The following R codes applies rule selection to the pruned rule set.

```
rules.selected <- selectRuleRRF(rules.pruned, X, target)
rules.present <- presentRules(rules.selected, colnames(X))
print(cbind(ID = 1:nrow(rules.present), rules.present[, c("condition", "pred")]))
```

After selection, only 16 rules are selected.

```
##      ID
## [1,] "1"
## [2,] "2"
## [3,] "3"
## [4,] "4"
## [5,] "5"
## [6,] "6"
## [7,] "7"
## [8,] "8"
## [9,] "9"
## [10,] "10"
## [11,] "11"
## [12,] "12"
## [13,] "13"
## [14,] "14"
## [15,] "15"
## [16,] "16"
##      condition

# [1,] "FDG %in% c('L1', 'L2') & HippoNV %in% c('L1')"

# [2,] "FDG %in% c('L1') & HippoNV %in% c('L1', 'L2')"

# [3,] "PTGENDER %in% c('L2') & FDG %in% c('L2') & AV45 %in% c('L1', 'L2') & rs3818361 %in% c('L2') & rs3851179 %in% c('L2')"
# [4,] "AGE %in% c('L3') & FDG %in% c('L1') & HippoNV %in% c('L1', 'L2')"
# [5,] "PTEDUCAT %in% c('L1') & AV45 %in% c('L2') & HippoNV %in% c('L2') & rs610932 %in% c('L2')"
# [6,] "HippoNV %in% c('L1') & rs3818361 %in% c('L1')"

# [7,] "AV45 %in% c('L3') & HippoNV %in% c('L1', 'L2')"
# [8,] "AV45 %in% c('L1', 'L2') & HippoNV %in% c('L2') && rs37646
```

```

50 %in% c('L1')"
# [9,] "AGE %in% c('L1') & PTGENDER %in% c('L2') & FDG %in% c('L
2') & AV45 %in% c('L1','L2') & HippoNV %in% c('L1')"
# [10,] "AGE %in% c('L3') & PTGENDER %in% c('L1') & PTEDUCAT %in%
c('L2') & AV45 %in% c('L1','L2')"
# [11,] "AGE %in% c('L2') & PTEDUCAT %in% c('L2','L3') & HippoNV
%in% c('L2','L3') & e4_1 %in% c('L2')"
# [12,] "AGE %in% c('L2') & PTEDUCAT %in% c('L3') & e4_1 %in% c(
'L1')"
# [13,] "PTEDUCAT %in% c('L1','L3') & e4_1 %in% c('L1') & rs11136
000 %in% c('L1') & rs610932 %in% c('L1')"
# [14,] "AGE %in% c('L2') & HippoNV %in% c('L2','L3') & rs3865444
%in% c('L1')"
# [15,] "AGE %in% c('L1','L2') & AV45 %in% c('L1')"
# [16,] "PTEDUCAT %in% c('L1','L3') & FDG %in% c('L2','L3') & Hip
poNV %in% c('L2','L3')"

##      pred
## [1,] "class_1"
## [2,] "class_1"
## [3,] "class_0"
## [4,] "class_1"
## [5,] "class_1"
## [6,] "class_1"
## [7,] "class_1"
## [8,] "class_0"
## [9,] "class_0"
## [10,] "class_0"
## [11,] "class_0"
## [12,] "class_0"
## [13,] "class_0"
## [14,] "class_0"
## [15,] "class_0"
## [16,] "class_0"

print(cbind(ID = 1:nrow(rules.present), rules.present[, c("len",
"freq", "err")])))

##      ID    len freq     err
## [1,] "1"  "2"  "0.279" "0.083"
## [2,] "2"  "2"  "0.279" "0.09"
## [3,] "3"  "5"  "0.029" "0.133"
## [4,] "4"  "3"  "0.122" "0.016"
## [5,] "5"  "4"  "0.031" "0.312"
## [6,] "6"  "2"  "0.207" "0.121"
## [7,] "7"  "3"  "0.172" "0.124"

```

```

## [8,] "8"  "4"  "0.06"  "0.194"
## [9,] "9"  "5"  "0.006"  "0"
## [10,] "10" "4"  "0.044"  "0.13"
## [11,] "11" "5"  "0.019"  "0.2"
## [12,] "12" "3"  "0.043"  "0.182"
## [13,] "13" "4"  "0.037"  "0.158"
## [14,] "14" "3"  "0.114"  "0.203"
## [15,] "15" "2"  "0.234"  "0.215"
## [16,] "16" "3"  "0.282"  "0.144"

```

Now let's extract the frequent variable interactions by the function `getFreqPattern()`. Here, we discretize the continuous features to 3 levels with equal frequency.

```

freqPattern <- getFreqPattern(rules.pruned)

top.pattern <- (freqPattern[which(as.numeric(freqPattern[, "len"])
  >= 2), ][1:5, ])
print(presentRules(top.pattern, colnames(X)))

```

And the top frequency variable interactions (with length greater than 2) are shown below.

```

##      len sup    conf
## [1,] "2" "0.038" "1"
## [2,] "2" "0.026" "1"
## [3,] "2" "0.023" "0.991"
## [4,] "2" "0.022" "0.953"
## [5,] "2" "0.021" "0.99"
##      condition

## [1,] "FDG %in% c('L2','L3') & HippoNV %in% c('L2','L3')"
## [2,] "AV45 %in% c('L1','L2') & HippoNV %in% c('L2','L3')"
## [3,] "HippoNV %in% c('L1') & rs3818361 %in% c('L1')"
## [4,] "AV45 %in% c('L3') & HippoNV %in% c('L1')"
## [5,] "rs610932 %in% c('L1') & HippoNV %in% c('L2','L3')"

##      pred
## [1,] "class_0"
## [2,] "class_0"
## [3,] "class_1"
## [4,] "class_1"
## [5,] "class_0"

```

An ordered rule set can be built using the selected rules by the function `buildLearner()`.

```

learner <- buildLearner(rules.selected, X, target)
learner.readable <- presentRules(learner, colnames(X))
print(cbind(ID = 1:nrow(learner.readable), learner.readable[, c("condition", "pred")]))
```

```

##           ID
## [1,] "1"
## [2,] "2"
## [3,] "3"
## [4,] "4"
## [5,] "5"
## [6,] "6"
## [7,] "7"
## [8,] "8"
## [9,] "9"
## [10,] "10"
## [11,] "11"
## [12,] "12"
##           condition

# [1,] "AGE %in% c('L3') & FDG %in% c('L1') & HippoNV %in% c('L1', 'L2')"
# [2,] "FDG %in% c('L1','L2') & HippoNV %in% c('L1')"
# [3,] "AGE %in% c('L2') & PTEDUCAT %in% c('L3') & e4_1 %in% c('L1')"
# [4,] "PTEDUCAT %in% c('L1','L3') & e4_1 %in% c('L1') & rs11136000 %in% c('L1') & rs610932 %in% c('L1')"
# [5,] "AGE %in% c('L3') & PTGENDER %in% c('L1') & AV45 %in% c('L1','L2')"
# [6,] "PTGENDER %in% c('L2') & FDG %in% c('L2') & AV45 %in% c('L1','L2') & rs3818361 %in% c('L2') & rs3851179 %in% c('L2')"
# [7,] "PTEDUCAT %in% c('L1','L3') & FDG %in% c('L2','L3') & HippoNV %in% c('L2','L3')"
# [8,] "AV45 %in% c('L1','L2') & HippoNV %in% c('L2') & rs3764650 %in% c('L1')"
# [9,] "FDG %in% c('L1') & HippoNV %in% c('L1','L2')"
# [10,] "AGE %in% c('L2') & HippoNV %in% c('L2','L3') & rs3865444 %in% c('L1')"
# [11,] "AGE %in% c('L1','L2') & AV45 %in% c('L1')"
# [12,] "Else"

##           pred
## [1,] "class_1"
## [2,] "class_1"
```

```

## [3,] "class_0"
## [4,] "class_0"
## [5,] "class_0"
## [6,] "class_0"
## [7,] "class_0"
## [8,] "class_0"
## [9,] "class_1"
## [10,] "class_0"
## [11,] "class_0"
## [12,] "class_0"

print(cbind(ID = 1:nrow(learner.readable), learner.readable[, c("len",
  "freq", "err")]))
```

ID	len	freq	err
[1,]	"1"	"3"	"0.121856866537718"
[2,]	"2"	"2"	"0.195357833655706"
[3,]	"3"	"3"	"0.034816247582205"
[4,]	"4"	"4"	"0.02321083172147"
[5,]	"5"	"4"	"0.0367504835589942"
[6,]	"6"	"5"	"0.0154738878143133"
[7,]	"7"	"3"	"0.2321083172147"
[8,]	"8"	"4"	"0.0212765957446809"
[9,]	"9"	"2"	"0.0328820116054159"
[10,]	"10"	"3"	"0.0425531914893617"
[11,]	"11"	"2"	"0.0851063829787234"
[12,]	"12"	"1"	"0.158607350096712"

IV. Exercises

Data analysis

1. Find 10 regression datasets from the UCI data repository or R datasets. Use inTrees to do analysis. Identify the final list of rules.

Programming

2. Simulate a dataset that has 10 variables, and design some interactions among the 10 variables (the form of the interactions is open-ended, e.g., it could be rule-based interactions, or any other statistical interactions). You can learn more from the simulation

study in this paper¹ to help you conduct this simulation. Implement inTrees to see if the interactions can be captured.

3. Increase the number of variables to be 100. Make the interaction patterns sparse, e.g., only 20 variable interactions. Implement inTrees to see if the interactions can be captured.

¹ Friedman, J.H. and Popescu, B.E. *Predictive learning via rule ensembles*. *Annals of applied statistics*, 2008.

CONCLUSION

Not to play devil's advocate, this book is named as analytics of *small data* for a reason. It doesn't mean that the methods introduced in this book could only be applied to small datasets. Rather, it is the approach of this book to introduce analytics methods through exemplary datasets as small as possible, small enough that we could grasp with perception or intuition, whatever readily accessible to us. Then, we illustrate what questions we could ask and what types of models we can build based on these small datasets. In this way, we hope to connect perceivable intuition with abstract formulations. We hope this endeavor is achieved by this book.

We also feel that we own an explanation of the cover image. It is a computer running statistical analysis using R, which is connected with a 3D printer in production. The computer analyzes the real-time measurements obtained from the 3D printer and generates results for the 3D printer such that the 3D printer can adjust its real-time production. This is what is happening in real world. The point we'd like to convey is, data analytics is

not just some cyber activities. Its impact is not confined in office. It is actually the brain device in many production systems and online operations such as search engines. Its application is everywhere.

In writing this book, we owe great debts to many people who generously share their materials and codes online. In online communities such as GitHub and stackoverflow, you can find many free resources which can launch you into a fast track of developing a project. Many insightful notes and technical reports have been posted online. And nowadays researchers post their latest results on arXiv.org so we could save much time without waiting for the manuscript to go through the entire publication period that could be lengthy. Of course, it is also our greatest gratitude to our pioneers who have made solid contributions in areas such as statistics, machine learning, and optimization, enabling us to use the data analytics tools.

To conclude this book, notice that much efforts of this book are devoted to show the development process of a technique that starts with a seed idea and ends with the technique that can be used. Such development processes are shown intuitively as much as we could. Also, in terms of completeness, we more focus on completeness in the development process instead of topics. There is an impression that that academic research is full of jargons, thus translational efforts are needed. We aim to provide a framework to consolidate our common sense with those jargons, particularly, the scientific considerations beyond those jargons, so readers can develop access to the wonderful resources available to us, made free by academic researchers, to implement data analytics for real-world problems. In dealing with reality, it is essentially performance and practice. In theory we speculate about reality, in reality we apply theories and reflect upon theories. At any rate, the mode of speculation of our mind is, as we believe, one essential faculty for us to learn and make decisions. Thus, this book, through a mode of speculation in introducing both theory and practice, aims to foster this capacity of speculation and help us to come to terms with what is given (as our data), help us to be critical, while not dismissive.

