R and Data Mining: Examples and Case Studies[1] [2]

Yanchang Zhao
yanchangzhao@gmail.com
http://www.RDataMining.com

December 1, 2011

[2]The latest version is available at http://www.rdatamining.com. See the website also for an *R Reference Card for Data Mining*.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This book presents examples and case studies on how to use R for data mining applications.

1.1 Data Mining

Data mining is ....

The main techniques for data mining are listed below. More detailed introduction can be found in text books on data mining [Han and Kamber, 2000, Hand et al., 2001, Witten and Frank, 2005].

- Clustering:
- Classification:
- Association Rules:
- Sequential Patterns:
- Time Series Analysis:
- Text Mining:

1.2 R

R [1] [R Development Core Team, 2010a] is a free software environment for statistical computing and graphics. It provides a wide variety of statistical and graphical techniques. R can be extended easily via packages. There are more than 3000 packages available in the CRAN package repository [2], as on May 20, 2011. More details about R are available in *An Introduction to R* [3] [Venables et al., 2010] and *R Language Definition* [4] [R Development Core Team, 2010c].

R is widely used in acadcemia and research, as well as industrial applications.

1.2.1 R Packages and Functions for Data Mining

A collection of R packages and functions available for data mining are listed below. Some of them are not specially for data mining, but they are included here because they are useful in data mining applications.

1. Clustering

[1] http://www.r-project.org/
[2] http://cran.r-project.org/
[3] http://cran.r-project.org/doc/manuals/R-intro.pdf
[4] http://cran.r-project.org/doc/manuals/R-lang.pdf

- Packages:
  - *fpc*
  - *cluster*
  - *pvclust*
  - *mclust*
- Partitioning-based clustering: `kmeans`, `pam`, `pamk`, `clara`
- Hierarchical clustering: `hclust`, `pvclust`, `agnes`, `diana`
- Model-based clustering: `mclust`
- Density-based clustering: `dbscan`
- Plotting cluster solutions: `plotcluster`, `plot.hclust`
- Validating cluster solutions: `cluster.stats`

2. Classification

- Packages:
  - *rpart*
  - *party*
  - *randomForest*
  - *rpartOrdinal*
  - *tree*
  - *marginTree*
  - *maptree*
  - *survival*
- Decision trees: `rpart`, `ctree`
- Random forest: `cforest`, `randomForest`
- Regression, Logistic regression, Poisson regression: `glm`, `predict`, `residuals`
- Survival analysis: `survfit`, `survdiff`, `coxph`

3. Association Rules and Frequent Itemsets

- Packages:
  - *arules*: supports to mine frequent itemsets, maximal frequent itemsets, closed frequent itemsets and association rules
  - *drm*: regression and association models for repeated categorical data
- APRIORI algorithm, a level-wise, breadth-first algorithm which counts transactions: `apriori`, `drm`
- ECLAT algorithm: employs equivalence classes, depth-first search and set intersection instead of counting: `eclat`

4. Sequential Patterns

- Package: *arulesSequences*
- SPADE algorithm: `cSPADE`

5. Time Series

- Package: *timsac*
- Time series construction: `ts`
- Decomposition: `decomp`, `decompose`, `stl`, `tsr`

6. Statistics

   - Package: Base R, *nlme*
   - Analysis of Variance: `aov`, `anova`
   - Density analysis: `density`
   - Statistical test: `t.test`, `prop.test`, `anova`, `aov`
   - Linear mixed-effects model fit: `lme`
   - Principal components and factor analysis: `princomp`

7. Graphics

   - Bar chart: `barplot`
   - Pie chart: `pie`
   - Scattered plot: `dotchart`
   - Histogram: `hist`
   - Density: `densityplot`
   - Candlestick chart, box plot: `boxplot`
   - QQ (quantile-quantile) plot: `qqnorm`, `qqplot`, `qqline`
   - Bi-variate plot: `coplot`
   - Tree: `rpart`
   - Parallel coordinates: `parallel`, `paracoor`, `parcoord`
   - Heatmap, contour: `contour`, `filled.contour`
   - Other plots: `stripplot`, `sunflowerplot`, `interaction.plot`, `matplot`, `fourfoldplot`, `assocplot`, `mosaicplot`
   - Saving graphs: `pdf`, `postscript`, `win.metafile`, `jpeg`, `bmp`, `png`

8. Data Manipulation

   - Missing values: `na.omit`
   - Standardize variables: `scale`
   - Transpose: `t`
   - Sampling: `sample`
   - Stack: `stack`, `unstack`
   - Others: `aggregate`, `merge`, `reshape`

9. Interface to Weka

   - *RWeka*: an R/Weka interface enabling to use all Weka functions in R.

1.3 Datasets

The datasets used in this book.

1.3.1 Iris Dataset

Iris dataset consists of 50 samples from each of three classes of iris flowers [Frank and Asuncion, 2010].One class is linearly separable from the other two, while the latter are not linearly separable from each other. There are five attributes in the dataset:

- sepal length in cm,
- sepal width in cm,
- petal length in cm,
- petal width in cm, and
- class: Iris Setosa, Iris Versicolour, and Iris Virginica.

```
> str(iris)

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

1.3.2 Bodyfat Dataset

`Bodyfat` is a dataset available in package *mboost*. It has 71 rows, which each row contains information of one person. It contains the following 10 numeric columns.

- `age`: age in years.
- `DEXfat`: body fat measured by DXA, response variable.
- `waistcirc`: waist circumference.
- `hipcirc`: hip circumference.
- `elbowbreadth`: breadth of the elbow.
- `kneebreadth`: breadth of the knee.
- `anthro3a`: sum of logarithm of three anthropometric measurements.
- `anthro3b`: sum of logarithm of three anthropometric measurements.
- `anthro3c`: sum of logarithm of three anthropometric measurements.
- `anthro4`: sum of logarithm of three anthropometric measurements.

The value of `DEXfat` is to be predicted by the other variables.

```
> data("bodyfat", package = "mboost")
> str(bodyfat)

'data.frame':        71 obs. of  10 variables:
 $ age         : num  57 65 59 58 60 61 56 60 58 62 ...
 $ DEXfat      : num  41.7 43.3 35.4 22.8 36.4 ...
 $ waistcirc   : num  100 99.5 96 72 89.5 83.5 81 89 80 79 ...
 $ hipcirc     : num  112 116.5 108.5 96.5 100.5 ...
 $ elbowbreadth: num  7.1 6.5 6.2 6.1 7.1 6.5 6.9 6.2 6.4 7 ...
```

```
$ kneebreadth : num  9.4 8.9 8.9 9.2 10 8.8 8.9 8.5 8.8 8.8 ...
$ anthro3a    : num  4.42 4.63 4.12 4.03 4.24 3.55 4.14 4.04 3.91 3.66 ...
$ anthro3b    : num  4.95 5.01 4.74 4.48 4.68 4.06 4.52 4.7 4.32 4.21 ...
$ anthro3c    : num  4.5 4.48 4.6 3.91 4.15 3.64 4.31 4.47 3.47 3.6 ...
$ anthro4     : num  6.13 6.37 5.82 5.66 5.91 5.14 5.69 5.7 5.49 5.25 ...
```

Chapter 2

Data Import/Export

This chapter shows how to import data into R and how to export R data frames. For more details, please refer to *R Data Import/Export* [1] [R Development Core Team, 2010b].

2.1 Save/Load R Data

Data in R can be saved as `.Rdata` files with functions `save`. After that, they can then be loaded into R with `load`.

```
> a <- 1:10
> save(a, file = "data/dumData.Rdata")
> rm(a)
> load("data/dumData.Rdata")
> print(a)

 [1]  1  2  3  4  5  6  7  8  9 10
```

2.2 Import from and Export to .CSV Files

The example below creates a dataframe `a` and save it as a `.CSV` file with `write.csv`. And then, the dataframe is loaded from file to variable `b` with `read.csv`.

```
> var1 <- 1:5
> var2 <- (1:5) / 10
> var3 <- c("R", "and", "Data Mining", "Examples", "Case Studies")
> a <- data.frame(var1, var2, var3)
> names(a) <- c("VariableInt", "VariableReal", "VariableChar")
> write.csv(a, "data/dummmyData.csv", row.names = FALSE)
> #rm(a)
> b <- read.csv("data/dummmyData.csv")
> print(b)

  VariableInt VariableReal VariableChar
1           1          0.1            R
2           2          0.2          and
3           3          0.3  Data Mining
4           4          0.4     Examples
5           5          0.5 Case Studies
```

[1] http://cran.r-project.org/doc/manuals/R-data.pdf

2.3 Import Data from SAS

Package foreign provides function `read.ssd` for importing SAS datasets (`.sas7bdat` files) into R. However, the following points are essential to make importing successful.

- SAS must be available on your computer, and `read.ssd` will call SAS to read SAS datasets and import them into R.
- The file name of a SAS dataset has to be no longer than eight characters. Otherwise, the importing would fail. There is no such a limit when importing from a `.CSV` file.
- During importing, variable names longer than eight characters are truncated to eight characters, which often makes it difficult to know the meanings of variables. One way to get around this issue is to import variable names separately from a `.CSV` file, which keeps full names of variables.

An empty `.CSV` file with variable names can be generated with the following method.

1. Create an empty SAS table `dumVariables` from `dumData` as follows.

```
data work.dumVariables;
   set work.dumData(obs=0);
run;
```

2. Export table `dumVariables` as a .CSV file.

The example below demonstrates importing data from a SAS dataset. Assume that there is a SAS data file `dumData.sas7bdat` and a .CSV file `dumVariables.csv` in folder "`Current working directory/data`".

```
> library(foreign) # for importing SAS data
> sashome <- "C:/Program Files/SAS/SAS 9.1"
> filepath <- "data"
> # filename should be no more than 8 characters, without extension
> fileName <- "dumData"
> # read data from a SAS dataset
> a <- read.ssd(file.path(filepath), fileName, sascmd = file.path(sashome, "sas.exe"))
> print(a)

  VARIABLE VARIABL2     VARIABL3
1        1      0.1            R
2        2      0.2          and
3        3      0.3  Data Mining
4        4      0.4     Examples
5        5      0.5 Case Studies
```

Note that the variable names above are truncated. The full names are imported from a .CSV file with the following code.

```
> variableFileName <- "dumVariables.csv"
> myNames <- read.csv(paste(filepath, variableFileName, sep="/"))
> names(a) <- names(myNames)
> print(a)

  VariableInt VariableReal VariableChar
1           1          0.1            R
2           2          0.2          and
3           3          0.3  Data Mining
4           4          0.4     Examples
5           5          0.5 Case Studies
```

Although one can export a SAS dataset to a `.CSV` file and then import data from it, there are problems when there are special formats in the data, such as a value of "$100,000" for a numeric variable. In this case, it would be better to import from a `.sas7bdat` file. However, variable names may need to be imported into R separately.

Another way to import data from a SAS dataset is to use function `read.xport` to read a file in SAS Transport (XPORT) format.

2.4 Import/Export via ODBC

Package *RODBC* provides connection to ODBC databases.

2.4.1 Read from Databases

```
> library(RODBC)
> Connection <- odbcConnect(dsn="servername",uid="userid",pwd="******")
> Query <- "SELECT * FROM lib.table WHERE ..."
> # or read query from file
> # Query <- readChar("data/myQuery.sql", nchars=99999)
> myData <- sqlQuery(Connection, Query, errors=TRUE)
> odbcCloseAll()
```

There are also `sqlSave` and `sqlUpdate` for writing or updating a table in an ODBC database.

2.4.2 Output to and Input from EXCEL Files

```
> library(RODBC)
> filename <- "data/dummmyData.xls"
> xlsFile <- odbcConnectExcel(filename, readOnly = FALSE)
> sqlSave(xlsFile, a, rownames = FALSE)
> b <- sqlFetch(xlsFile, "a")
> odbcCloseAll()
```

Note that there is a limit of the number of rows to write to an EXCEL file.

Chapter 3

Data Exploration

This chapter shows basic exploration of `iris` data (see Section 1.3.1 for details of iris data).

3.1 Have a Look at Data

Check the dimensionality

```
> dim(iris)
```

```
[1] 150   5
```

Variable names or column names

```
> names(iris)
```

```
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

Structure

```
> str(iris)
```

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

Attributes

```
> attributes(iris)
```

```
$names
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"

$row.names
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
```

```
[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
[145] 145 146 147 148 149 150

$class
[1] "data.frame"
```

Get the first 5 rows

```
> iris[1:5,]

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
```

The first or last parts of `iris` data can be retrieved with `head()` or `tail()`.

```
> head(iris)

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

> tail(iris)

    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

Get Sepal.Length of the first 10 rows

```
> iris[1:10, "Sepal.Length"]

 [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

Same as above

```
> iris$Sepal.Length[1:10]

 [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

3.2 Explore Individual Variables

Distribution of every numeric variable can be checked with function `summary()`, which returns the minimum, maximum, mean, median, and the first (25%) and third quantile (75%). For factors, it shows the frequency of every level.

```
> summary(iris)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

The mean, median and range can be obtained with functions with `mean()`, `median()` and `range()`. Quantiles and percentiles are supported by function `quantile()` as below.

```
> quantile(iris$Sepal.Length)
```

```
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
```

```
> quantile(iris$Sepal.Length, c(.1, .3, .65))
```

```
 10%  30%  65%
4.80 5.27 6.20
```

Frequency

```
> table(iris$Species)
```

```
    setosa versicolor  virginica
        50         50         50
```

Pie chart

```
> pie(table(iris$Species))
```

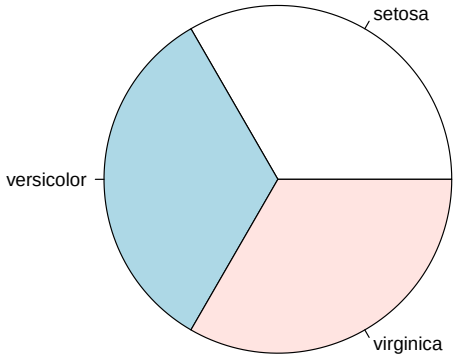

Figure 3.1: Pie Chart

Variance of Sepal.Length

```
> var(iris$Sepal.Length)
```

```
[1] 0.6856935
```

Histogram

```
> hist(iris$Sepal.Length)
```

Figure 3.2: Histogram

Density

```
> plot(density(iris$Sepal.Length))
```

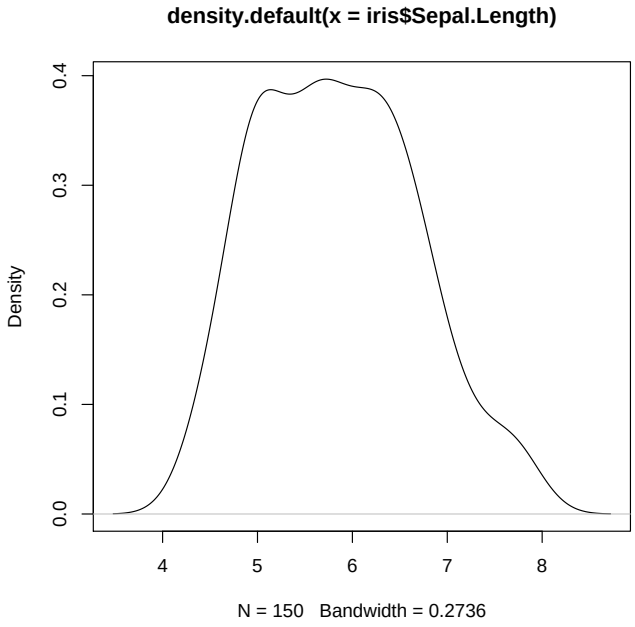

Figure 3.3: Density

3.3 Explore Multiple Variables

Covariance of two variables

```
> cov(iris$Sepal.Length, iris$Petal.Length)
```

```
[1] 1.274315
```

```
> cov(iris[,1:4])
```

```
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

Correlation of two variables

```
> cor(iris$Sepal.Length, iris$Petal.Length)
```

```
[1] 0.8717538
```

```
> cor(iris[,1:4])
```

```
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

Distribution in subsets

```
> aggregate(Sepal.Length ~ Species, summary, data=iris)
```

```
     Species Sepal.Length.Min. Sepal.Length.1st Qu. Sepal.Length.Median
1     setosa             4.300                4.800                5.000
2 versicolor             4.900                5.600                5.900
3  virginica             4.900                6.225                6.500
  Sepal.Length.Mean Sepal.Length.3rd Qu. Sepal.Length.Max.
1             5.006                5.200             5.800
2             5.936                6.300             7.000
3             6.588                6.900             7.900
```

Box plot, also known as box-and-whisker plot, shows the median, first and third quartile of a distribution (i.e., the 50%, 25% and 75% points in cumulative distribution), and outliers. The bar in the middle is the median. The box shows the interquartile range (IQR), which is the range between the 75% and 25% observation.

```
> boxplot(Sepal.Length~Species, data=iris)
```

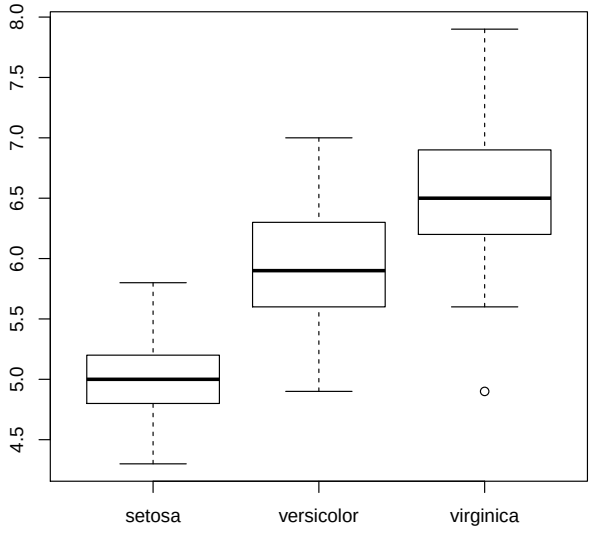

Figure 3.4: Boxplot

Scatter plot

```
> plot(iris$Sepal.Length, iris$Sepal.Width)
```

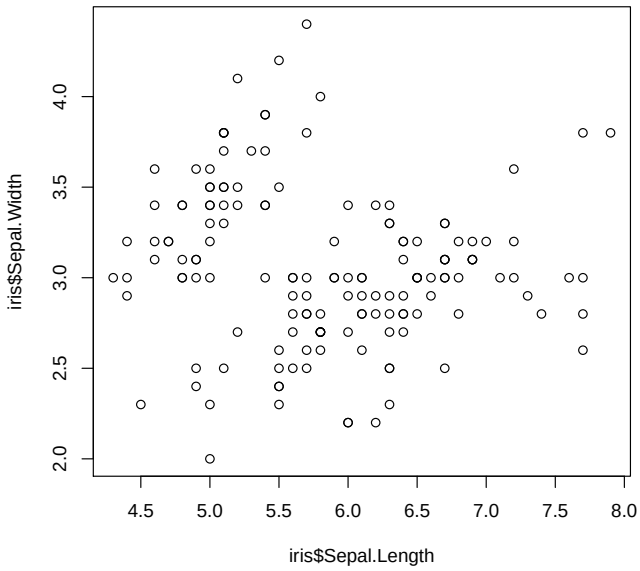

Figure 3.5: Scatter Plot

Pairs plot

```
> pairs(iris)
```

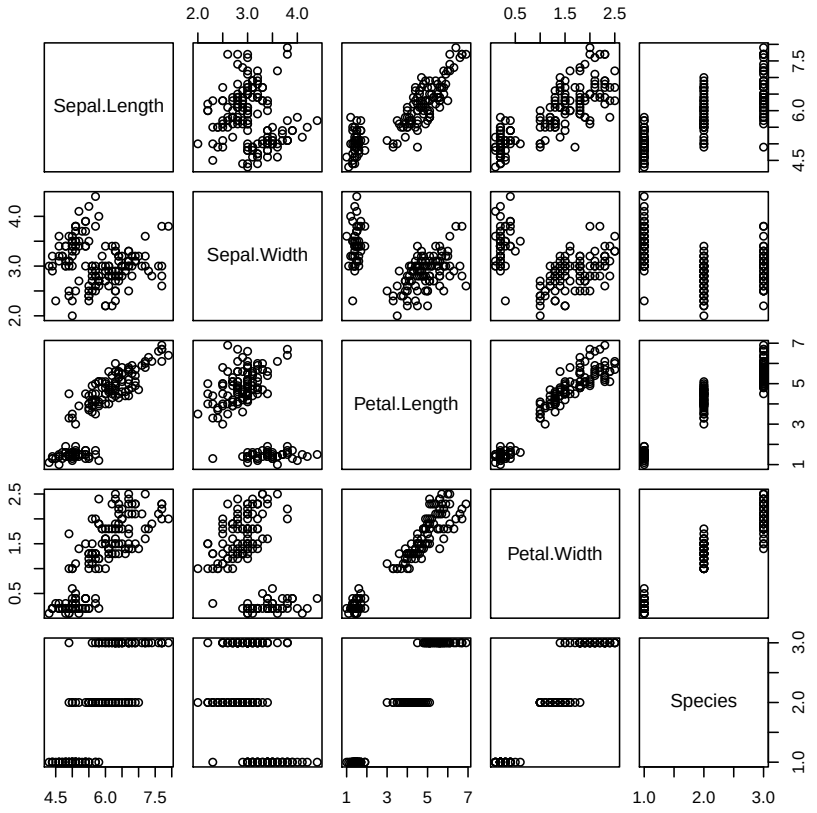

Figure 3.6: Pairs Plot

3.4 More Exploration

3D Scatter plot

```
> library(scatterplot3d)
> scatterplot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```

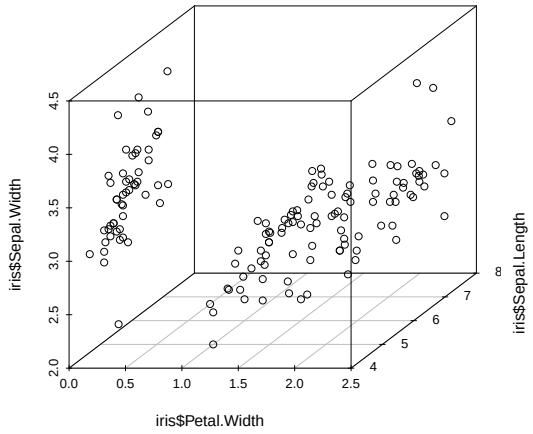

Figure 3.7: 3D Scatter plot

Level Plot

```
> library(lattice)
> print(levelplot(Petal.Width~Sepal.Length*Sepal.Width, iris, cuts=9, col.regions=grey.colors(10)
```

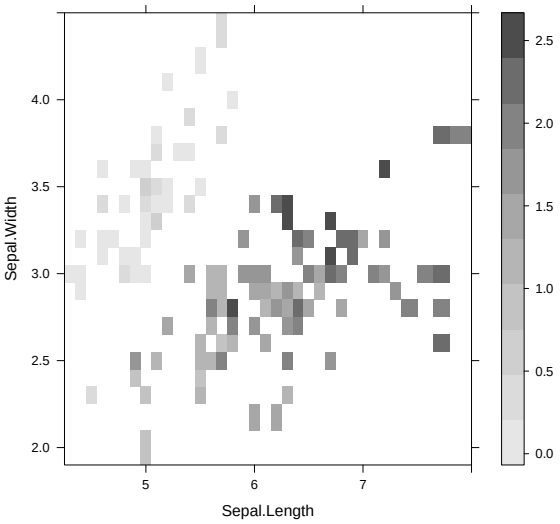

Figure 3.8: Level Plot

Contour

```
> filled.contour(volcano, color = terrain.colors, asp = 1, plot.axes=contour(volcano, add=T) )
```

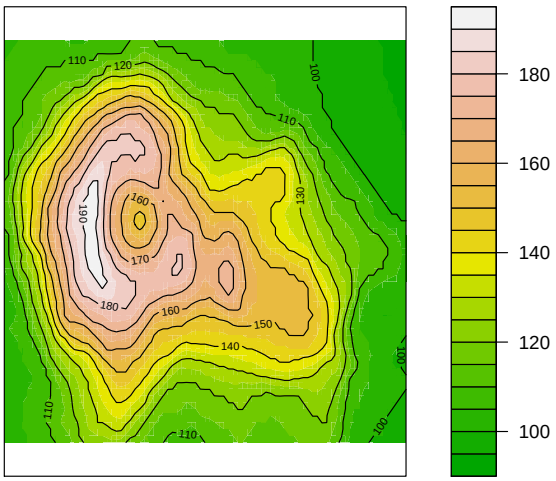

Figure 3.9: Contour

3D Surface

```
> persp(volcano, theta = 25, phi = 30, expand = 0.5, col = "lightblue")
```

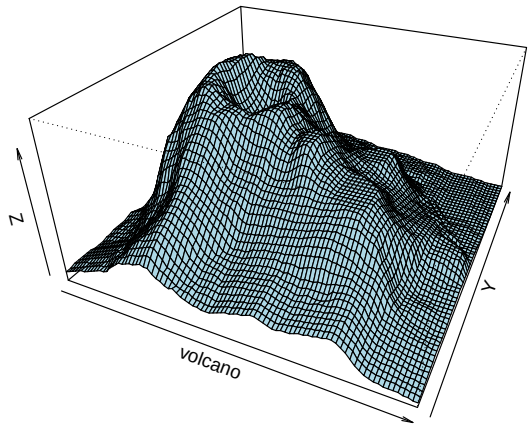

Figure 3.10: 3D Surface

Interactive 3D Scatter Plot

```
> library(rgl)
> plot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```

3.5 Save Charts as Files

Save as a .PDF file

```
> pdf("myPlot.pdf")
> x <- 1:50
> plot(x, log(x))
> graphics.off()
```

Save as a postscript file

```
> postscript("myPlot.ps")
> x <- -20:20
> plot(x, x^2)
> graphics.off()
```

Chapter 4

Decision Trees

There are a couple of R packages on decision trees, regression trees and random forest, such as *rpart*, *rpartOrdinal*, *randomForest*, *party*, *tree*, *marginTree* and *maptree*.

This chapter shows how to build prediction models with packages *party*, *rpart* and *randomForest*.

4.1 Building Decision Trees with Package *party*

This section shows how to build a decision tree for `iris` data (see Section 1.3.1 for details of the data) with `ctree` in package *party* [Hothorn et al., 2010]. `Sepal.Length`, `Sepal.Width`, `Petal.Length` and `Petal.Width` are used to predict the `Species` of flowers. In the package, function `ctree` builds a decision tree, and `predict` makes prediction for unlabeled data.

The `iris` data is split below into two subsets: training (70%) and testing (30%).

```
> str(iris)
```

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

```
> set.seed(1234)
> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
> trainData <- iris[ind==1,]
> testData <- iris[ind==2,]
```

Load package *party*, build a decision tree, and check the prediction.

```
> library(party)
> myFormula <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
> iris_ctree <- ctree(myFormula, data=trainData)
> # check the prediction
> table(predict(iris_ctree), trainData$Species)
```

```
            setosa versicolor virginica
  setosa         40          0         0
  versicolor      0         37         3
  virginica       0          1        31
```

Have a look at the built tree.

```
> print(iris_ctree)
```

```
        Conditional inference tree with 4 terminal nodes

Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  112

1) Petal.Length <= 1.9; criterion = 1, statistic = 104.643
  2)*  weights = 40
1) Petal.Length > 1.9
  3) Petal.Width <= 1.7; criterion = 1, statistic = 48.939
    4) Petal.Length <= 4.4; criterion = 0.974, statistic = 7.397
      5)*  weights = 21
    4) Petal.Length > 4.4
      6)*  weights = 19
  3) Petal.Width > 1.7
    7)*  weights = 32
```

```
> plot(iris_ctree)
```

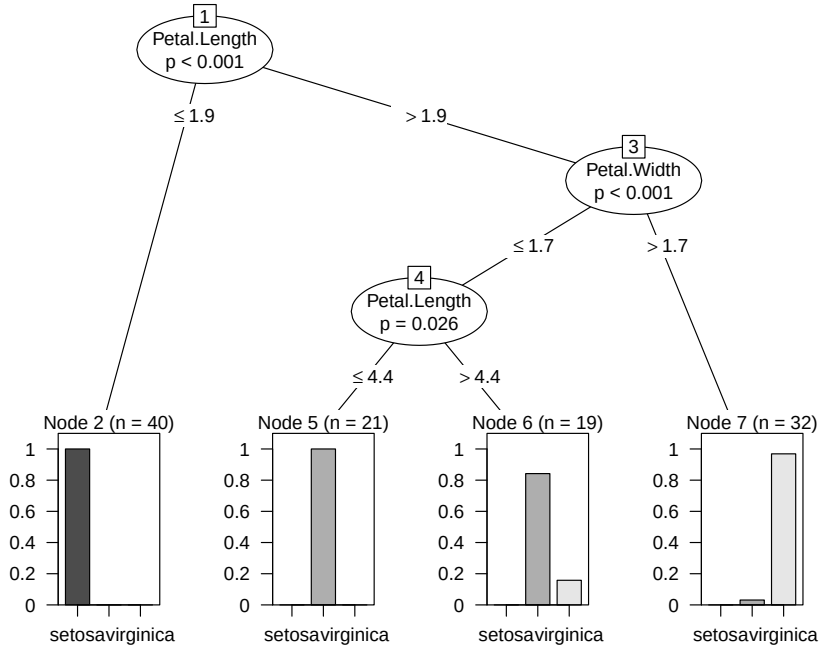

Figure 4.1: Decision Tree

```
> plot(iris_ctree, type="simple")
```

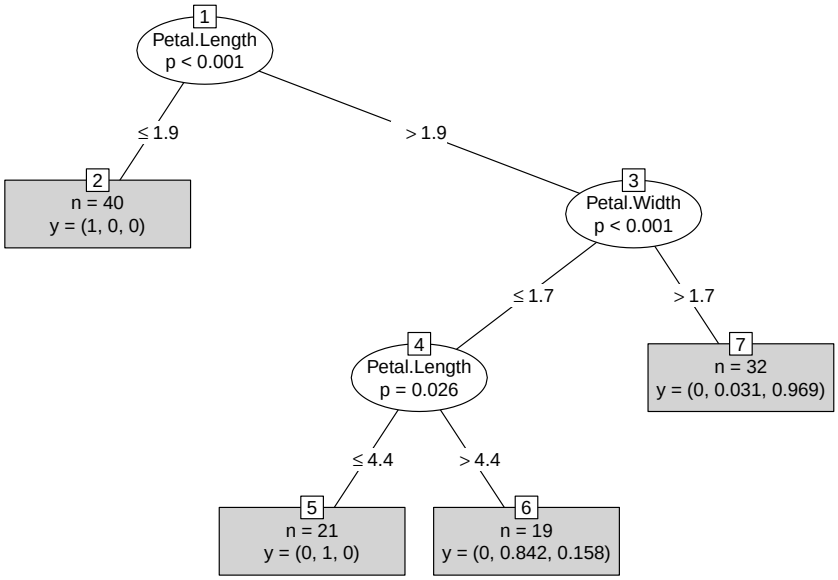

Figure 4.2: Decision Tree (Simple Style)

Test the built tree with test data.

```
> # predict on test data
> testPred <- predict(iris_ctree, newdata = testData)
> table(testPred, testData$Species)

testPred     setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         12         2
  virginica       0          0        14
```

The current version of `ctree` (i.e. version 0.9-9995) does not handle missing values well. An instance with a missing value may sometimes go to the left sub-tree and sometimes to the right.

Another issue is that, when a variable exists in training data and is fed into `ctree` but does not appear in the built decision tree, the test data must also have that variable to make prediction. Otherwise, a call to `predict` would fail. Moreover, if the value levels of a categorical variable in test data are different from that in train data, it would also fail to make prediction on the test data. One way to get around the above issue is, after building a decision tree, to call `ctree` build a new decision tree with data containing only those variables existing in the first tree, and to explicitly set the levels of categorical variables in test data to the levels of the corresponding variables in train data.

4.2 Building Decision Trees with Package *rpart*

Package *rpart* [Therneau et al., 2010] is used to build a decision tree on `bodyfat` data (see Section 1.3.2 for details of the data). Function `rpart` is used to build a decision tree, and the tree with the minimum prediction error is select. After that, it is applied to makes prediction for unlabeled data with function `predict`.

```
> data("bodyfat", package = "mboost")
> dim(bodyfat)

[1] 71 10

> attributes(bodyfat)

$names
 [1] "age"          "DEXfat"       "waistcirc"    "hipcirc"      "elbowbreadth"
 [6] "kneebreadth"  "anthro3a"     "anthro3b"     "anthro3c"     "anthro4"

$row.names
 [1] "47"  "48"  "49"  "50"  "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"
[13] "59"  "60"  "61"  "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"
[25] "71"  "72"  "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"  "81"  "82"
[37] "83"  "84"  "85"  "86"  "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"
[49] "95"  "96"  "97"  "98"  "99"  "100" "101" "102" "103" "104" "105" "106"
[61] "107" "108" "109" "110" "111" "112" "113" "114" "115" "116" "117"

$class
[1] "data.frame"

> bodyfat[1:5,]

   age DEXfat waistcirc hipcirc elbowbreadth kneebreadth anthro3a anthro3b
47  57  41.68     100.0   112.0          7.1         9.4     4.42     4.95
48  65  43.29      99.5   116.5          6.5         8.9     4.63     5.01
49  59  35.41      96.0   108.5          6.2         8.9     4.12     4.74
50  58  22.79      72.0    96.5          6.1         9.2     4.03     4.48
51  60  36.42      89.5   100.5          7.1        10.0     4.24     4.68
   anthro3c anthro4
47     4.50    6.13
48     4.48    6.37
49     4.60    5.82
50     3.91    5.66
51     4.15    5.91

> library(rpart)
> myFormula <- DEXfat ~ age + waistcirc + hipcirc + elbowbreadth + kneebreadth
> bodyfat_rpart <- rpart(myFormula, data = bodyfat, control = rpart.control(minsplit = 10))
> attributes(bodyfat_rpart)

$names
 [1] "frame"     "where"     "call"      "terms"     "cptable"   "splits"
 [7] "method"    "parms"     "control"   "functions" "y"         "ordered"

$class
[1] "rpart"

> print(bodyfat_rpart$cptable)

          CP nsplit  rel error    xerror       xstd
1 0.66289544      0 1.00000000 1.0219736 0.16843699
2 0.09376252      1 0.33710456 0.4035820 0.09175474
3 0.07703606      2 0.24334204 0.4180559 0.08719801
4 0.04507506      3 0.16630598 0.3302274 0.07686651
```

```
5 0.01844561      4 0.12123092 0.2520385 0.05796306
6 0.01818982      5 0.10278532 0.2564169 0.05809595
7 0.01000000      6 0.08459549 0.2153285 0.05701589
```

```
> print(bodyfat_rpart)
```

```
n= 71

node), split, n, deviance, yval
      * denotes terminal node

 1) root 71 8535.98400 30.78282
   2) waistcirc< 88.4 40 1315.35800 22.92375
     4) hipcirc< 96.25 17  285.91370 18.20765
       8) age< 59.5 11   97.00440 15.96000 *
       9) age>=59.5 6   31.45788 22.32833 *
     5) hipcirc>=96.25 23  371.86530 26.40957
      10) waistcirc< 80.75 13  117.60710 24.13077 *
      11) waistcirc>=80.75 10   98.99016 29.37200 *
   3) waistcirc>=88.4 31 1562.16200 40.92355
     6) kneebreadth< 11.15 28  615.52590 39.26036
      12) hipcirc< 109.9 13  136.29600 35.27846 *
      13) hipcirc>=109.9 15   94.46997 42.71133 *
     7) kneebreadth>=11.15 3  146.28030 56.44667 *
```

```
> plot(bodyfat_rpart)
> text(bodyfat_rpart, use.n=TRUE)
```

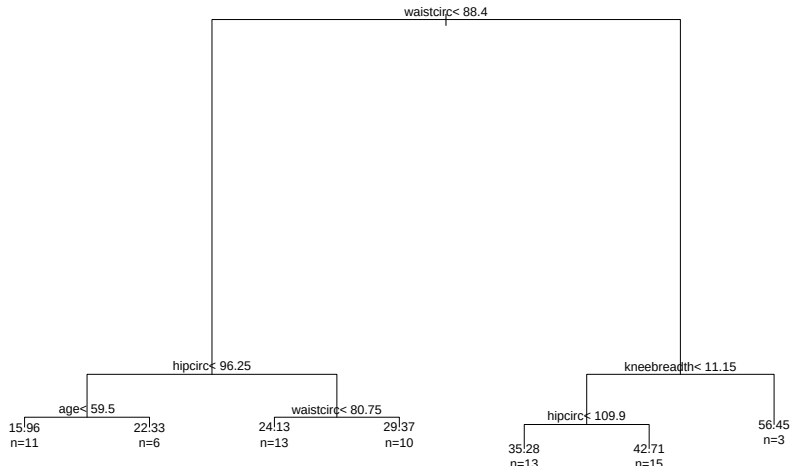

Figure 4.3: Decision Tree with *rpart*

```
> opt <- which.min(bodyfat_rpart$cptable[,"xerror"])
> cp <- bodyfat_rpart$cptable[opt, "CP"]
> bodyfat_prune <- prune(bodyfat_rpart, cp = cp)
> print(bodyfat_prune)
```

```
n= 71

node), split, n, deviance, yval
      * denotes terminal node

 1) root 71 8535.98400 30.78282
   2) waistcirc< 88.4 40 1315.35800 22.92375
     4) hipcirc< 96.25 17  285.91370 18.20765
       8) age< 59.5 11   97.00440 15.96000 *
       9) age>=59.5 6   31.45788 22.32833 *
     5) hipcirc>=96.25 23  371.86530 26.40957
      10) waistcirc< 80.75 13  117.60710 24.13077 *
      11) waistcirc>=80.75 10   98.99016 29.37200 *
   3) waistcirc>=88.4 31 1562.16200 40.92355
     6) kneebreadth< 11.15 28  615.52590 39.26036
      12) hipcirc< 109.9 13  136.29600 35.27846 *
      13) hipcirc>=109.9 15   94.46997 42.71133 *
     7) kneebreadth>=11.15 3  146.28030 56.44667 *
```

```
> DEXfat_pred <- predict(bodyfat_prune, newdata = bodyfat)
```

The predicted values are compared with real labels.

```
> xlim <- range(bodyfat$DEXfat)
> plot(DEXfat_pred ~ DEXfat, data = bodyfat, xlab = "Observed", ylab = "Predicted", ylim = xlim,
> abline(a = 0, b = 1)
```

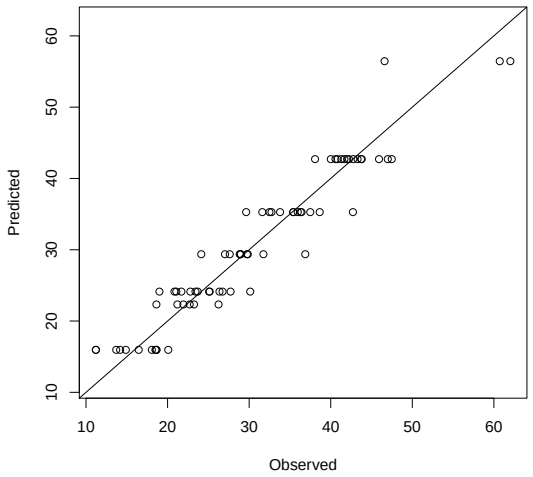

Figure 4.4: Prediction Result

4.3 Random Forest

Package *randomForest* is used to build a predictive model for `iris` data (see Section 1.3.1 for details of the data). An alternative way is to use function `cforest` from package *randomForest*.

The `iris` data is split below into two subsets: training (70%) and testing (30%).

```
> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
> trainData <- iris[ind==1,]
> testData <- iris[ind==2,]
```

Load *randomForest* and then train a random forest.

```
> library(randomForest)
> rf <- randomForest(Species ~ ., data=trainData, ntree=100, proximity=TRUE)
> table(predict(rf), trainData$Species)
```

```
            setosa versicolor virginica
  setosa         38          0         0
  versicolor      0         33         2
  virginica       0          2        28
```

```
> print(rf)
```

```
Call:
 randomForest(formula = Species ~ ., data = trainData, ntree = 100,      proximity = TRUE)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 2

        OOB estimate of  error rate: 3.88%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         38          0         0  0.00000000
versicolor      0         33         2  0.05714286
virginica       0          2        28  0.06666667
```

```
> attributes(rf)
```

```
$names
 [1] "call"            "type"            "predicted"       "err.rate"
 [5] "confusion"       "votes"           "oob.times"       "classes"
 [9] "importance"      "importanceSD"    "localImportance" "proximity"
[13] "ntree"           "mtry"            "forest"          "y"
[17] "test"            "inbag"           "terms"

$class
[1] "randomForest.formula" "randomForest"
```

Error rates with various number of trees

```
> plot(rf)
```

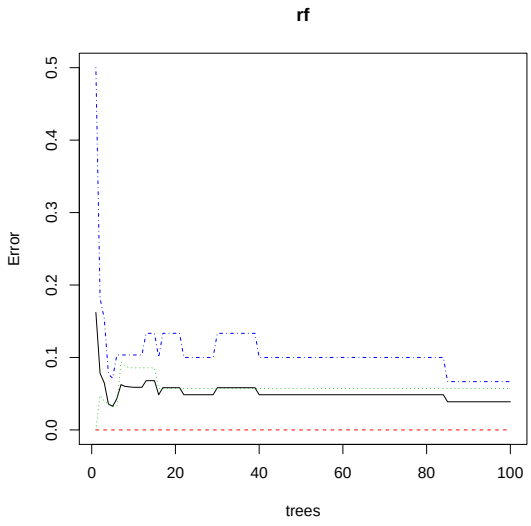

Figure 4.5: Error Rate of Random Forest

Variable importance.

```
> importance(rf)

             MeanDecreaseGini
Sepal.Length         6.653214
Sepal.Width          1.319307
Petal.Length        29.236710
Petal.Width         30.427564

> varImpPlot(rf)
```

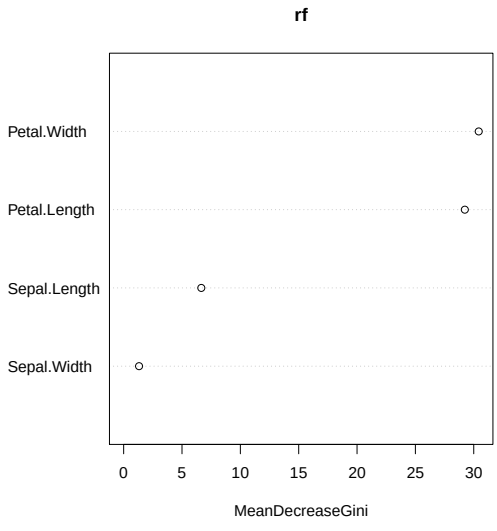

Figure 4.6: Variable Importance

Test the built random forest on test data

```
> irisPred <- predict(rf, newdata=testData)
> table(irisPred, testData$Species)

irisPred     setosa versicolor virginica
  setosa         12          0         0
  versicolor      0         15         3
  virginica       0          0        17

> plot(margin(rf,testData$Species))
```

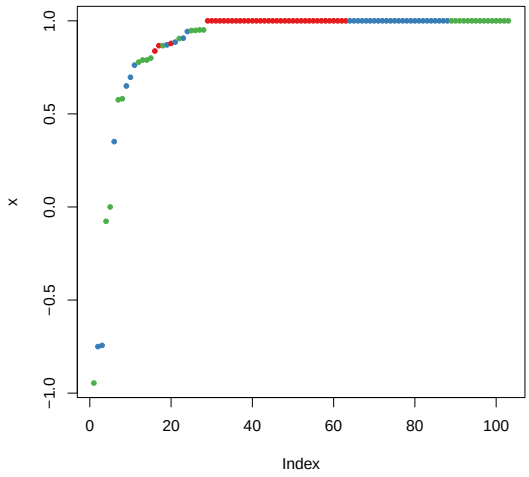

Figure 4.7: Margin of Predictions

The margin of a data point is as the proportion of votes for the correct class minus maximum proportion of votes for the other classes. Generally speaking, positive margin means correct classification.

Chapter 5

Regression

Regression is to build a function of *independent variables* (also known as *predictors*) to predict a *dependent variable* (also called *response*). For example, banks assess the risk of home-loan customers based on their age, income, expenses, occupation, number of dependents, total credit limit, etc.

A collection of some helpful R functions for regression analysis is available as a reference card on *R Functions for Regression Analysis* [1].

This chapter shows how to do linear regression with function `lm`, generalized linear regression with `glm`, and non-linear regression with `nls`.

5.1 Linear Regression

Linear regression is to predict response with a linear function of predictors as follows:

$$y = c_0 + c_1 x_1 + c_2 x_2 + \cdots + c_k x_k,$$

where $x_1, x_2, \cdots, x_k$ are predictors and y is the response to predict.

Linear regression is demonstrated below with function `lm` on the Australian CPI (Consumer Price Index) data, which are CPIs in four quarters in every year from 2008 to 2010 [2].

[1] `http://cran.r-project.org/doc/contrib/Ricci-refcard-regression.pdf`
[2] From Australian Bureau of Statistics <`http://www.abs.gov.au`>

```
> ##cpi <- read.csv("CPI.csv")
> year <- rep(2008:2010, each=4)
> quarter <- rep(1:4, 3)
> cpi <- c(162.2, 164.6, 166.5, 166.0, 166.2, 167.0, 168.6, 169.5,
+          171.0, 172.1, 173.3, 174.0)
> plot(cpi, xaxt="n", ylab="CPI", xlab="")
> # draw x-axis
> axis(1, labels=paste(year,quarter,sep="Q"), at=1:12, las=3)
```

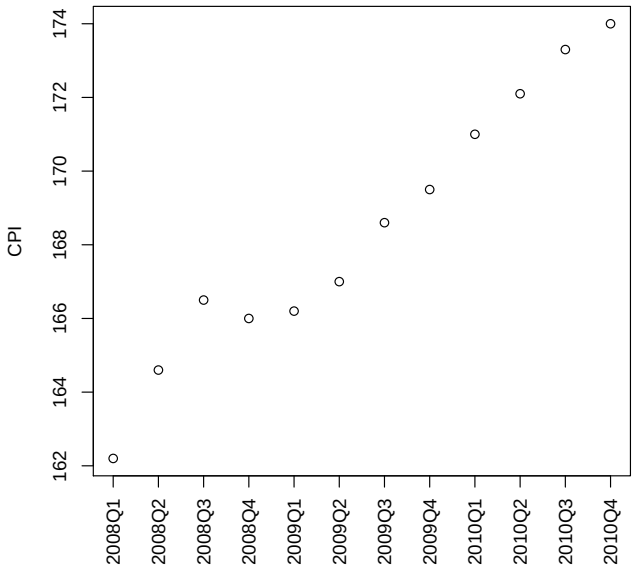

Figure 5.1: Australian CPIs in Year 2008 to 2010

We check the correlation between `CPI` and the other variables, `year` and `quarter`.

```
> cor(year,cpi)

[1] 0.9096316

> cor(quarter,cpi)

[1] 0.3738028
```

Then a linear regression model is built on the above data, using `year` and `quarter` as predictors and `CPI` as response.

```
> fit <- lm(cpi ~ year + quarter)
> fit

Call:
lm(formula = cpi ~ year + quarter)

Coefficients:
(Intercept)         year      quarter
  -7644.487        3.887        1.167
```

With the above linear model, CPI is calculated as

$$\text{cpi} = c_0 + c_1 * \text{year} + c_2 * \text{quarter},$$

where c_0, c_1 and c_2 are coefficients from model `fit`. Therefore, the CPIs in 2011 can be get as follows. A simpler way for this is using function `predict`, which will be demonstrated at the end of this subsection.

```
> cpi2011 <- fit$coefficients[[1]] + fit$coefficients[[2]]*2011 + fit$coefficients[[3]]*(1:4)
```

More details of the model:

```
> attributes(fit)

$names
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"

$class
[1] "lm"

> fit$coefficients

 (Intercept)         year      quarter
-7644.487500     3.887500     1.166667
```

The differences between observed values and fitted values are

```
> #differences between observed values and fitted values
> residuals(fit)

          1           2           3           4           5           6
-0.57916667  0.65416667  1.38750000 -0.27916667 -0.46666667 -0.83333333
          7           8           9          10          11          12
-0.40000000 -0.66666667  0.44583333  0.37916667  0.41250000 -0.05416667

> summary(fit)

Call:
lm(formula = cpi ~ year + quarter)

Residuals:
    Min      1Q  Median      3Q     Max
-0.8333 -0.4948 -0.1667  0.4208  1.3875

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -7644.4875   518.6543 -14.739 1.31e-07 ***
year            3.8875     0.2582  15.058 1.09e-07 ***
quarter         1.1667     0.1885   6.188 0.000161 ***
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1

Residual standard error: 0.7302 on 9 degrees of freedom
Multiple R-squared: 0.9672,        Adjusted R-squared: 0.9599
F-statistic: 132.5 on 2 and 9 DF,  p-value: 2.108e-07
```

```
> plot(fit)
```

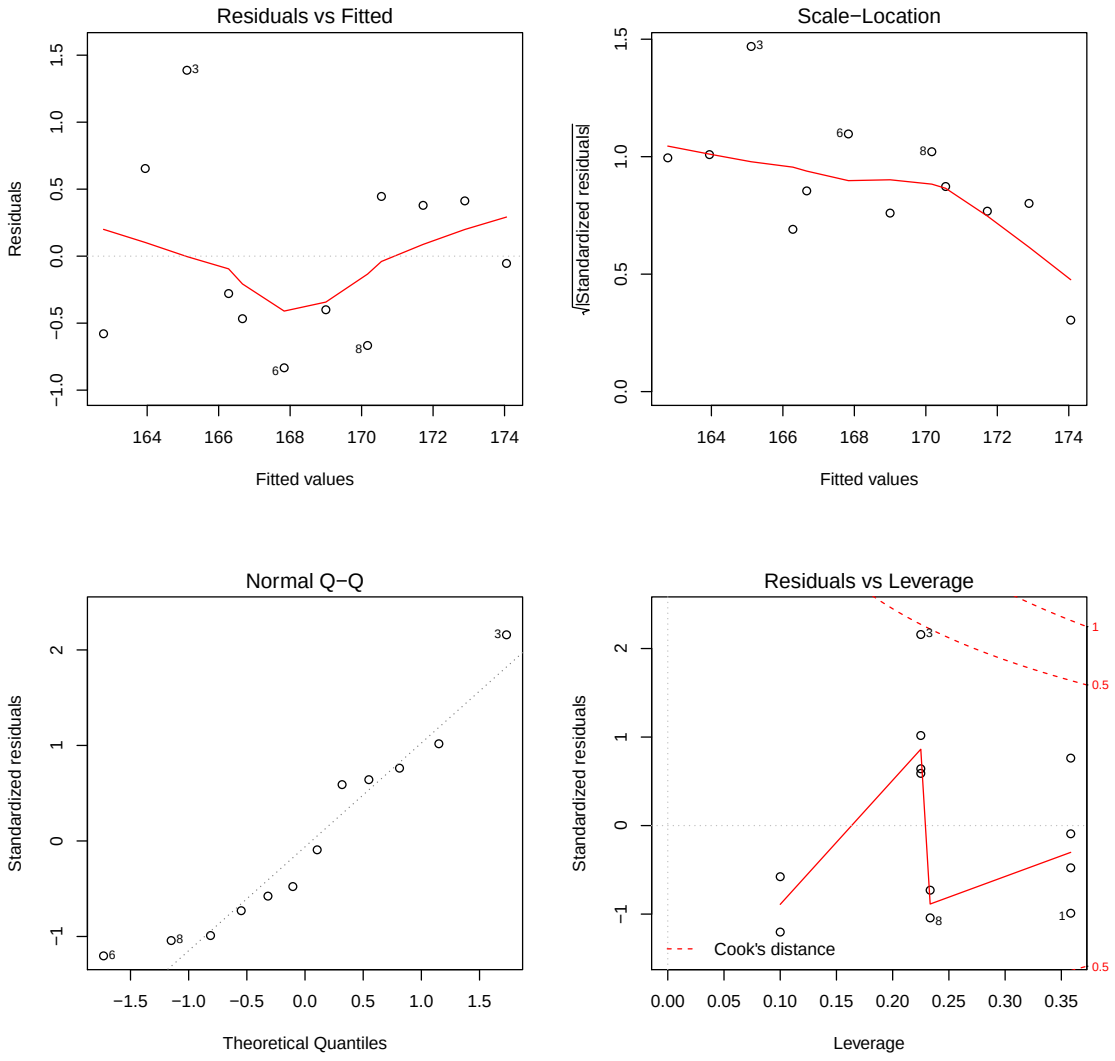

Figure 5.2: Prediction with Linear Regression Model - 1

With the model, the CPIs in year 2011 can be predicted as follows, and the predicted values are shown as red triangles in Figure 5.3.

```
> data2011 <- data.frame(year=2011, quarter=1:4)
> cpi2011 <- predict(fit, newdata=data2011)
> style <- c(rep(1,12), rep(2,4))
> plot(c(cpi, cpi2011), xaxt="n", ylab="CPI", xlab="", pch=style, col=style)
> axis(1, at=1:16, las=3,
+      labels=c(paste(year,quarter,sep="Q"), "2011Q1", "2011Q2", "2011Q3", "2011Q4"))
```

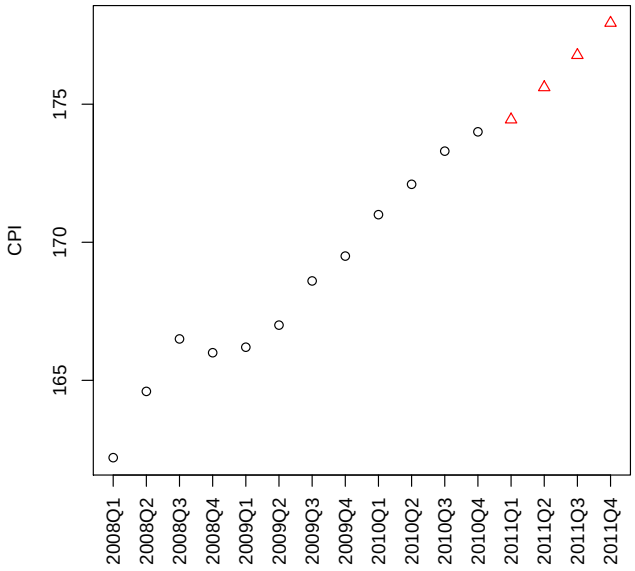

Figure 5.3: Prediction of CPIs in 2011 with Linear Regression Model

5.2 Logistic Regression

Logistic regression is used to predict the probability of occurrence of an event by fitting data to a logistic curve. A logistic regression model is built as the following equation:

$$logit(y) = c_0 + c_1x_1 + c_2x_2 + \cdots + c_kx_k,$$

where $x_1, x_2, \cdots, x_k$ are predictors, y is a response to predict, and $logit(y) = ln(\frac{y}{1-y})$. The above equation can also be written as

$$y = \frac{1}{1 + e^{-(c_0+c_1x_1+c_2x_2+\cdots+c_kx_k)}}.$$

Logistic regression can be built with function `glm` by setting `family` to `binomial(link="logit")`. Detailed introductions on logistic regression can be found at the following links.

- R Data Analysis Examples - Logit Regression
  `http://www.ats.ucla.edu/stat/r/dae/logit.htm`

- Logistic Regression (with R)
  `http://nlp.stanford.edu/~manning/courses/ling289/logistic.pdf`

5.3 Generalized Linear Regression

The generalized linear model (GLM) generalizes linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value. It unifies various other statistical models, including linear regression, logistic regression and Poisson regression. Function `glm` is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

A generalized linear model is built below with `glm` on `bodyfat` data (see Section 1.3.2 for details of the data).

```
> data("bodyfat", package = "mboost")
> myFormula <- DEXfat ~ age + waistcirc + hipcirc + elbowbreadth + kneebreadth
> bodyfat.glm <- glm(myFormula, family = gaussian("log"), data = bodyfat)
> summary(bodyfat.glm)
```

```
Call:
glm(formula = myFormula, family = gaussian("log"), data = bodyfat)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-11.5688   -3.0065    0.1266    2.8310   10.0966

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.734293   0.308949   2.377  0.02042 *
age          0.002129   0.001446   1.473  0.14560
waistcirc    0.010489   0.002479   4.231 7.44e-05 ***
hipcirc      0.009702   0.003231   3.003  0.00379 **
elbowbreadth 0.002355   0.045686   0.052  0.95905
kneebreadth  0.063188   0.028193   2.241  0.02843 *
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1

(Dispersion parameter for gaussian family taken to be 20.31433)

    Null deviance: 8536.0  on 70  degrees of freedom
Residual deviance: 1320.4  on 65  degrees of freedom
AIC: 423.02

Number of Fisher Scoring iterations: 5
```

```
> pred <- predict(bodyfat.glm, type = "response")
```

In the code above, `type` indicates the type of prediction required. The default is on the scale of the linear predictors, and the alternative `"response"` is on the scale of the response variable.

```
> plot(bodyfat$DEXfat, pred, xlab="Observed Values", ylab="Predicted Values")
```

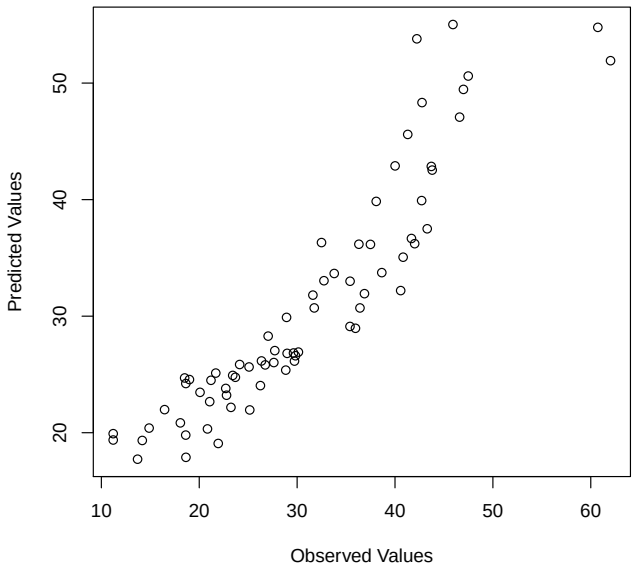

Figure 5.4: Prediction with Generalized Linear Regression Model

In the above code, if `family = gaussian("identity")` is used, the resulted model would be similar to linear regression. One can also make it a logistic regression by setting `family` to `binomial("logit")`.

5.4 Non-linear Regression

While linear regression is to find the line that comes closest to data, non-linear regression is to fit a curve through data. Function `nls` provides nonlinear regression. More details on non-linear regression can be found at

- A Complete Guide to Nonlinear Regression
  `http://www.curvefit.com/`.

Chapter 6

Clustering

6.1 K-means Clustering

This chapter demonstrates k-means clustering of `iris` data (see Section 1.3.1 for details of the data)..

```
> newiris <- iris
> newiris$Species <- NULL
```

Apply `kmeans` to `newiris`, and store the clustering result in `kc`. The cluster number is set to 3.

```
> (kc <- kmeans(newiris, 3))

K-means clustering with 3 clusters of sizes 38, 50, 62

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     6.850000    3.073684     5.742105    2.071053
2     5.006000    3.428000     1.462000    0.246000
3     5.901613    2.748387     4.393548    1.433871

Clustering vector:
  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [75] 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1
[112] 1 1 3 3 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 3 1 1 1 1 3 1 1 1 3 1 1 1 3 1
[149] 1 3

Within cluster sum of squares by cluster:
[1] 23.87947 15.15100 39.82097
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"
```

Compare the `Species` label with the clustering result

```
> table(iris$Species, kc$cluster)
```

```
            1  2  3
setosa      0 50  0
versicolor  2  0 48
virginica  36  0 14
```

The above result shows that cluster "setosa" can be easily separated from the other clusters, and that clusters "'versicolor" and "virginica" are to a small degree overlapped with each other.

Plot the clusters and their centers. Note that there are four dimensions in the data and that only the first two dimensions are used to draw the plot below. Some black points close to the green center (asterisk) are actually closer to the black center in the four dimensional space. Note that the results of k-means clustering may vary from run to run, due to random selection of initial cluster centers.

```
> plot(newiris[c("Sepal.Length", "Sepal.Width")], col = kc$cluster)
> points(kc$centers[,c("Sepal.Length", "Sepal.Width")], col = 1:3, pch = 8, cex=2)
```

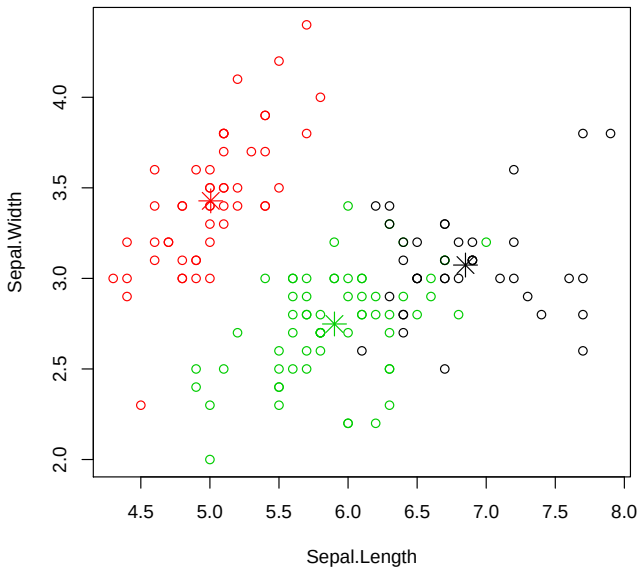

Figure 6.1: Results of K-means Clustering

6.2 Hierarchical Clustering

This page demonstrates a hierarchical clustering with hclust on `iris` data (see Section 1.3.1 for details of the data).

Draw a sample of 40 records from iris data, and remove variable `Species`

```
> idx <- sample(1:dim(iris)[1], 40)
> irisSample <- iris[idx,]
> irisSample$Species <- NULL
```

Hierarchical clustering

```
> hc <- hclust(dist(irisSample), method="ave")
```

```
> plot(hc, hang = -1, labels=iris$Species[idx])
```

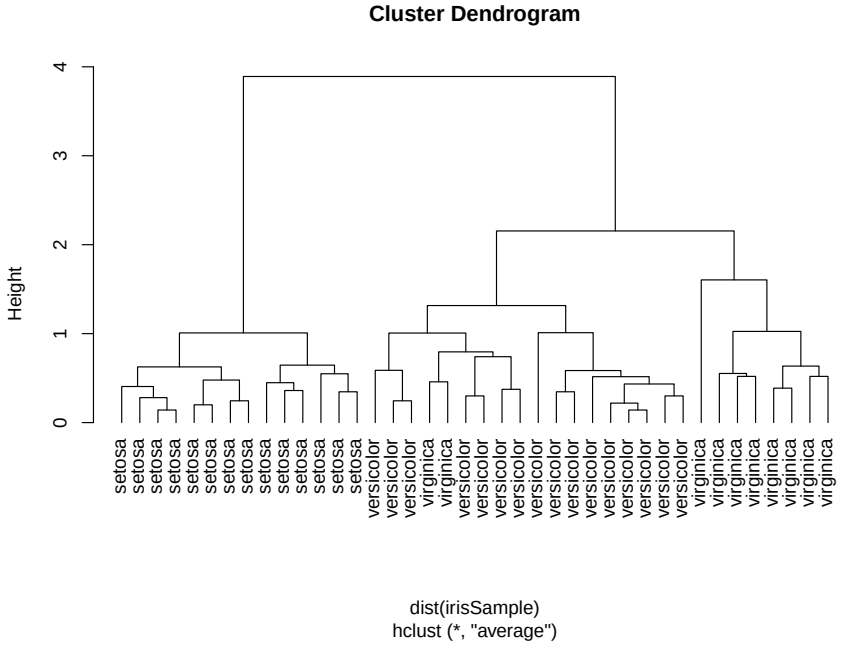

Figure 6.2: Cluster Dendrogram

Similar to the above clustering of k-means, Figure 6.2 also shows that cluster "setosa" can be easily separated from the other two clusters, and that clusters "versicolor" and "virginica" are to a small degree overlapped with each other.

6.3 Density-based Clustering

DBSCAN from package *fpc* provides a density-based clustering for numeric data [Ester et al., 1996]. The idea of density-based clustering is to group objects into one cluster if they are connected to one another by densely populated area. There are two key parameters in DBSCAN:

- `eps`: Reachability Distance, which defines the size of neighborhood;
- `MinPts`: Reachability minimum no. of points.

If the number of points in the neighborhood of point α is no less than `MinPts`, then α is a *dense point*. All the points in its neighborhood are *density-reachable* from α and are put into the same cluster as α.

The strengths of density-based clustering are that it can discover clusters with various shapes and sizes and is insensitive to noise. As a comparison, k-means tends to find clusters with sphere shape and with similar sizes.

```
> library(fpc)
> newiris <- iris[-5] # remove class tags
> ds <- dbscan(newiris, eps=0.42, MinPts=5)
> # compare clusters with original class labels
> table(ds$cluster, iris$Species)

    setosa versicolor virginica
  0      2         10        17
```

```
  1      48           0           0
  2       0          37           0
  3       0           3          33
```

In the above table, "1" to "3" in the first column are three discovered clusters, while "0" stands for noises, i.e., objects that are not assigned to any clusters. The noises are shown as black circles in figures below.

```
> plot(ds, newiris)
```

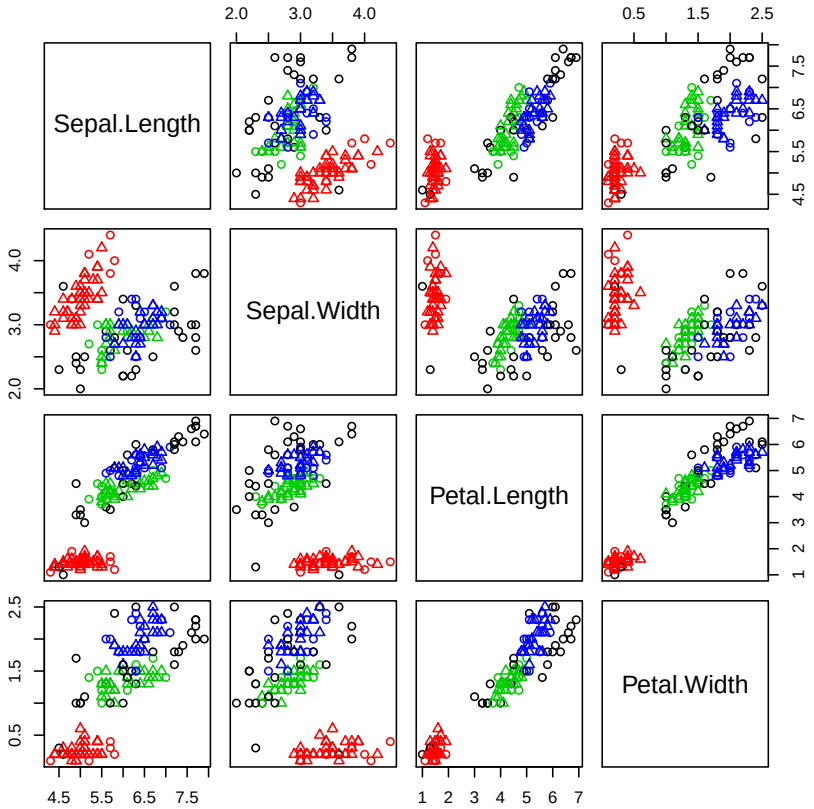

Figure 6.3: Density-based Clustering - I

```
> plot(ds, newiris[c(1,4)])
```

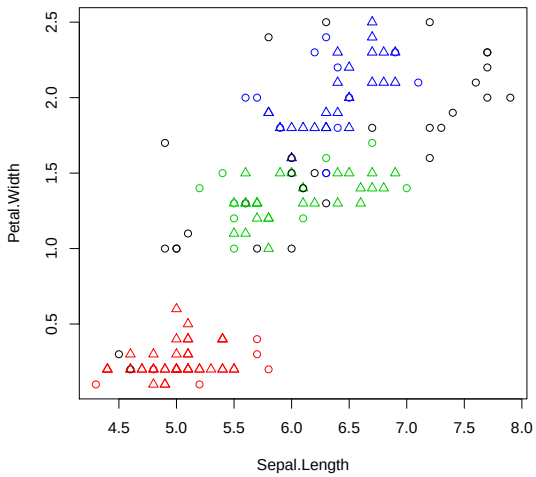

Figure 6.4: Density-based Clustering - II

Another way to show the clusters. Note that the data are projected to distinguish classes.

```
> plotcluster(newiris, ds$cluster)
```

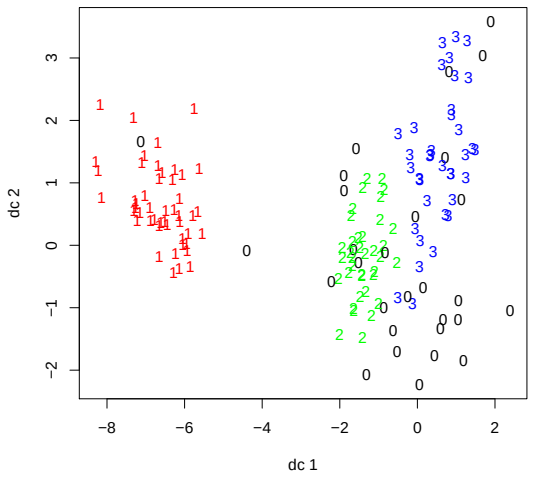

Figure 6.5: Density-based Clustering - III

Label new data

The clustering model can then be used to label new data, based on the similarity between a new object and the clusters. The following example draws a sample of 10 objects from iris and adds small noise to them to make a new dataset for labeling.

```
> # create a new dataset for labeling
> set.seed(435)
> idx <- sample(1:nrow(iris), 10)
> newData <- iris[idx,-5]
> newData <- newData + matrix(runif(10*4, min=0, max=0.2), nrow=10, ncol=4)
> # label new data
> myPred <- predict(ds, newiris, newData)
> # check the labels assigned to new data
> plot(newiris[c(1,4)], col=1+ds$cluster)
> points(newData[c(1,4)], pch="*", col=1+myPred, cex=3)
> # check cluster labels
> table(myPred, iris$Species[idx])

myPred setosa versicolor virginica
     0      0          0         1
     1      3          0         0
     2      0          3         0
     3      0          1         2
```

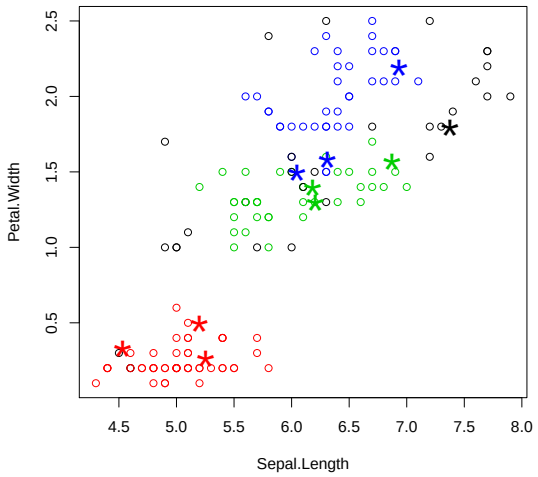

Figure 6.6: Prediction with Clustering Model

As we can see from the above result, in the 10 new unlabeled data, 8(=3+3+2) are assigned with correct class labels. The new data are shown as asterisk("*") in the above figure and the colors stand for cluster labels.

6.4 Fuzzy Clustering

This section is not available yet in this version.

6.5 Subspace Clustering

This section is not available yet in this version.

Chapter 7

Outlier Detection

This chapter is not available yet in this version.

Chapter 8

Time Series Analysis and Mining

This chapter presents examples on time series decomposition, forecast, clustering and classification.

8.1 Time Series Data in R

Class `ts` represents data which has been sampled at equispaced points in time. A frequency of 7 indicates that a time series is composed of weekly data, and 12 and 4 are used for respectively a monthly series and a quarterly series. An example below shows the construction of a time series with 30 values (1 to 30). `Frequency=12` and `start=c(2011,3)` suggest that it is a monthly series starting from March 2011.

```
> a <- ts(1:30, frequency=12, start=c(2011,3))
> print(a)

     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2011           1   2   3   4   5   6   7   8   9  10
2012  11  12  13  14  15  16  17  18  19  20  21  22
2013  23  24  25  26  27  28  29  30

> str(a)

 Time-Series [1:30] from 2011 to 2014: 1 2 3 4 5 6 7 8 9 10 ...

> attributes(a)

$tsp
[1] 2011.167 2013.583   12.000

$class
[1] "ts"
```

8.2 Time Series Decomposition

Time Series Decomposition is to decompose a time series into trend, seasonal, cyclical and irregular components. The trend component stands for long term trend, the seasonal component is seasonal variation, the cyclical component is repeated but non-periodic fluctuations, and the residuals are irregular component.

using moving averages

A time series of `AirPassengers` is used below as an example to demonstrate time series decomposition. It is composed of monthly totals of Box Jenkins international airline passengers from 1949 to 1960. It has 144(=12*12) values.

```
> plot(AirPassengers)
```

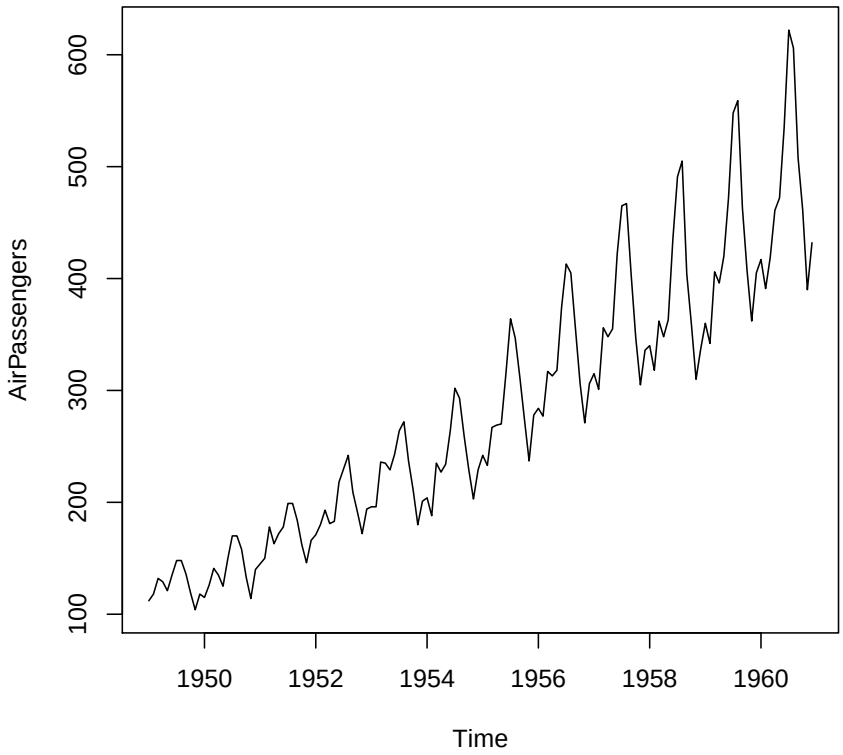

Figure 8.1: A Time Series of `AirPassengers`

Function `decompose` is called to `AirPassengers` to break it into various components.

```
> # decompose time series
> apts <- ts(AirPassengers, frequency = 12)
> f <- decompose(apts)
> # seasonal figures
> f$figure

 [1] -24.748737 -36.188131  -2.241162  -8.036616  -4.506313  35.402778
 [7]  63.830808  62.823232  16.520202 -20.642677 -53.593434 -28.619949

> plot(f$figure, type="b", xaxt="n", xlab="")
> # get names of 12 months in English words
> monthNames <- months(ISOdate(2011,1:12,1))
> # label x-axis with month names
> # las is set to 2 for vertical label orientation
> axis(1, at=1:12, labels=monthNames, las=2)
```

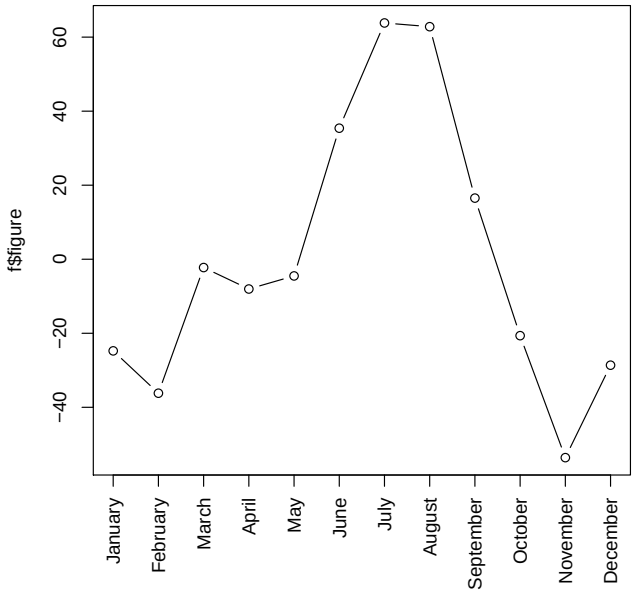

Figure 8.2: Seasonal Component

```
> plot(f)
```

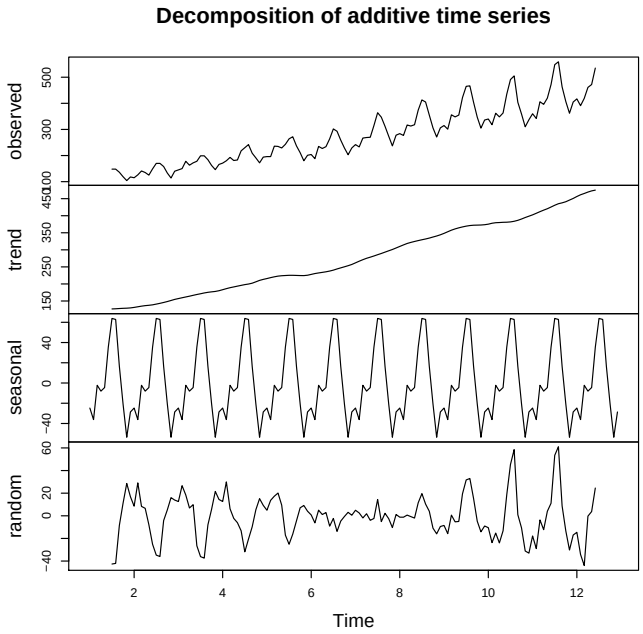

Figure 8.3: Time Series Decomposition

The first chart is the original time series. The second is trend in the data, the third shows seasonal factors, and the last chart is the remaining components after removing trend and seasonal factors. Some other functions for time series decomposition are `stl` in package *stats*, `decomp` in package *timsac*, and `tsr` in package *ast*.

8.3 Time Series Forecasting

Time series forecasting is to forecast future events based on known past data. One example is to predict the opening price of a stock based on its past performance. Some popular models are autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA).

Here is an example to fit an ARIMA model to a univariate time series and then use it for forecasting.

```
> fit <- arima(AirPassengers, order=c(1,0,0), list(order=c(2,1,0), period=12))
> fore <- predict(fit, n.ahead=24)
> # error bounds at 95% confidence level
> U <- fore$pred + 2*fore$se
> L <- fore$pred - 2*fore$se
> ts.plot(AirPassengers, fore$pred, U, L, col=c(1,2,4,4), lty = c(1,1,2,2))
> legend("topleft", c("Actual", "Forecast", "Error Bounds (95% Confidence)"),
+        col=c(1,2,4), lty=c(1,1,2))
```

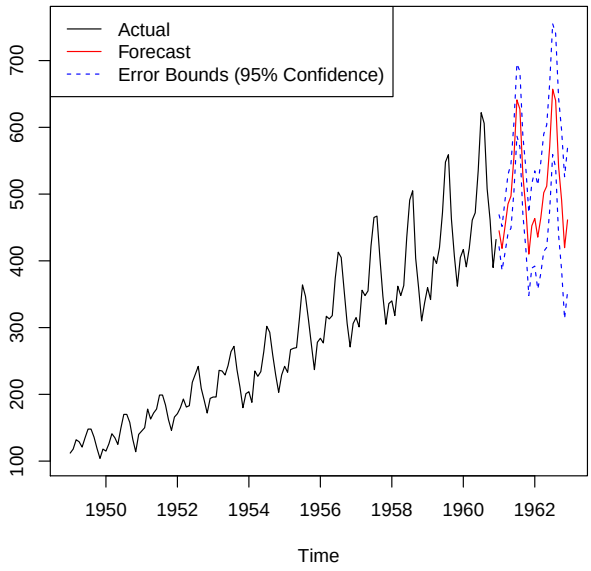

Figure 8.4: Time Series Forecast

The red solid line shows the forecasted values, and the blue dotted lines are error bounds at a confidence level of 95%.

8.4 Time Series Clustering

Time series clustering is to partition time series data into groups based on similarity or distance, so that time series in the same cluster are similar.

Measure of distance/dissimilarity: Euclidean distance, Manhattan distance, Maximum norm, Hamming distance, the angle between two vectors (inner product), Dynamic Time Warping (DTW) distance, etc.

8.4.1 Dynamic Time Warping

Dynamic Time Warping (DTW) [Keogh and Pazzani, 2001] finds optimal alignment between two time series. Package *dtw* [Giorgino, 2009] is used in the example below. In this package, function `dtw(x, y, ...)` computes dynamic time warp and finds optimal alignment between two time series `x` and `y`, and `dtwDist(mx, my=mx, ...)` or `dist(mx, my=mx, method="DTW", ...)` calculates the distances between time series `mx` and `my`.

```
> library(dtw)

Loaded dtw v1.14-3. See ?dtw for help, citation("dtw") for usage conditions.

> idx <- seq(0, 2*pi, len=100)
> a <- sin(idx) + runif(100)/10
> b <- cos(idx)
> align <- dtw(a, b, step=asymmetricP1, keep=T)
> dtwPlotTwoWay(align)
```

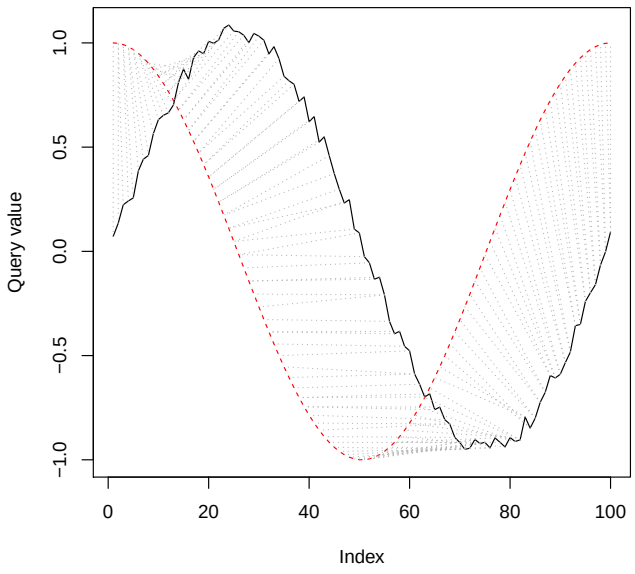

Figure 8.5: Alignment with Dynamic Time Warping

8.4.2 Synthetic Control Chart Time Series Data

Synthetic Control Chart Time Series [1] The dataset contains 600 examples of control charts synthetically generated by the process in Alcock and Manolopoulos (1999). Each control chart is a time series with 60 values. There are six classes:

- 1-100 Normal
- 101-200 Cyclic
- 201-300 Increasing trend
- 301-400 Decreasing trend
- 401-500 Upward shift
- 501-600 Downward shift

Firstly, the data is read into R with `read.table`. Parameter `sep` is set to `""` (no space between double quotation marks), which is used when the separator is white space, i.e., one or more spaces, tabs, newlines or carriage returns.

[1] `http://kdd.ics.uci.edu/databases/synthetic_control/synthetic_control.html`

```
> sc <- read.table("data/synthetic_control.data", header=F, sep="")
> # show one sample from each class
> idx <- c(1,101,201,301,401,501)
> sample1 <- t(sc[idx,])
> plot.ts(sample1, main="")
```

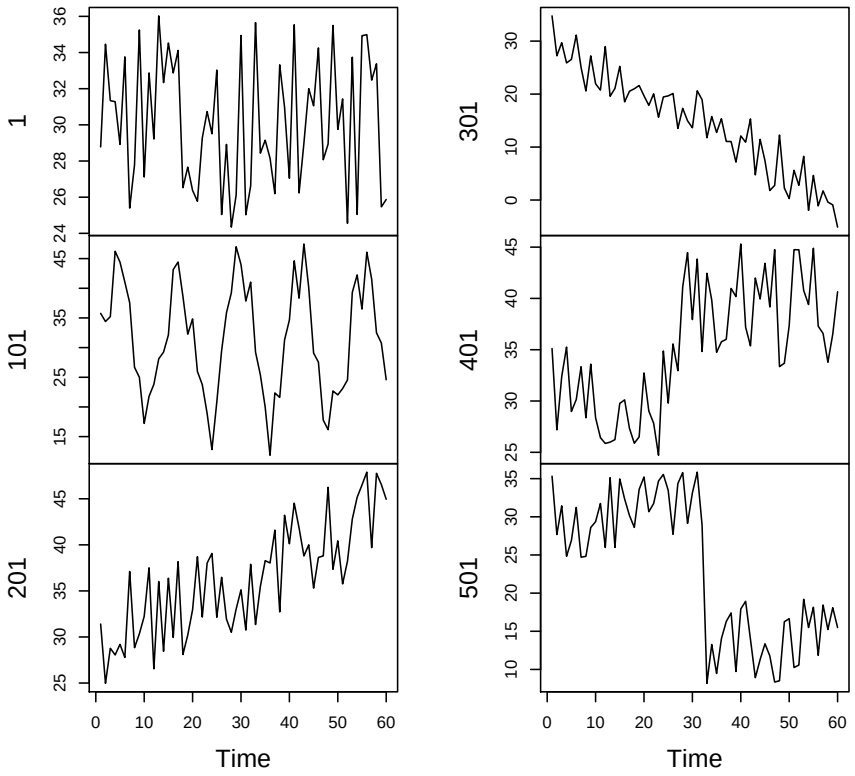

Figure 8.6: Six Classes in Synthetic Control Chart Time Series

8.4.3 Hierarchical Clustering with Euclidean Distance

n cases were randomly sampled from each class.

```
> n <- 10
> s <- sample(1:100, n)
> idx <- c(s, 100+s, 200+s, 300+s, 400+s, 500+s)
> sample2 <- sc[idx,]
> observedLabels <- c(rep(1,n), rep(2,n), rep(3,n), rep(4,n), rep(5,n), rep(6,n))
> # hierarchical clustering with Euclidean distance
> hc <- hclust(dist(sample2), method="ave")
> plot(hc, labels=observedLabels, main="")
```

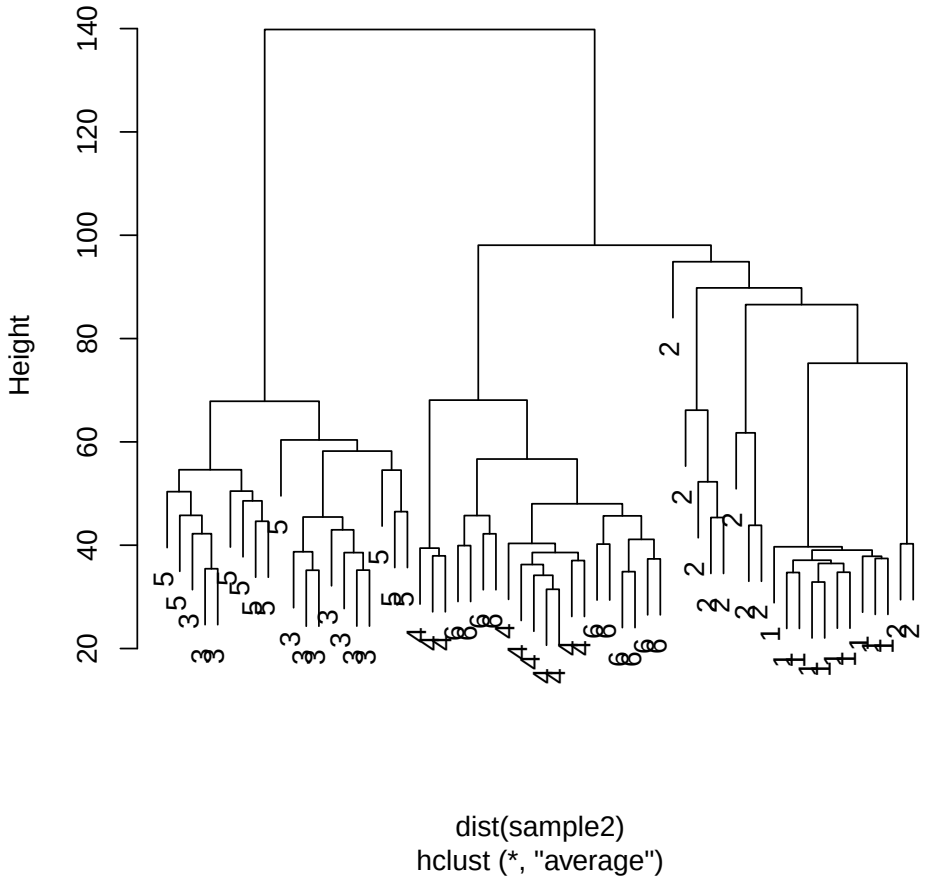

Figure 8.7: Hierarchical Clustering with Euclidean Distance

cut tree to get 8 clusters

```
> memb <- cutree(hc, k=8)
> table(observedLabels, memb)
```

```
               memb
observedLabels  1  2  3  4  5  6  7  8
             1 10  0  0  0  0  0  0  0
             2  0  4  2  3  1  0  0  0
             3  0  0  0  0  0 10  0  0
             4  0  0  0  0  0  0  3  7
```

```
5  0  0  0  0  0 10  0  0
6  0  0  0  0  0  0  0 10
```

8.4.4 Hierarchical Clustering with DTW Distance

```
> distMatrix <- dist(sample2, method="DTW")
> hc <- hclust(distMatrix, method="average")
> plot(hc, labels=observedLabels, main="")
> # cut tree to get 8 clusters
> memb <- cutree(hc, k=8)
> table(observedLabels, memb)

              memb
observedLabels  1  2  3  4  5  6  7  8
             1 10  0  0  0  0  0  0  0
             2  0  6  2  2  0  0  0  0
             3  0  0  0  0 10  0  0  0
             4  0  0  0  0  0  3  7  0
             5  0  0  0  0  0  0  0 10
             6  0  0  0  0  0  0 10  0
```

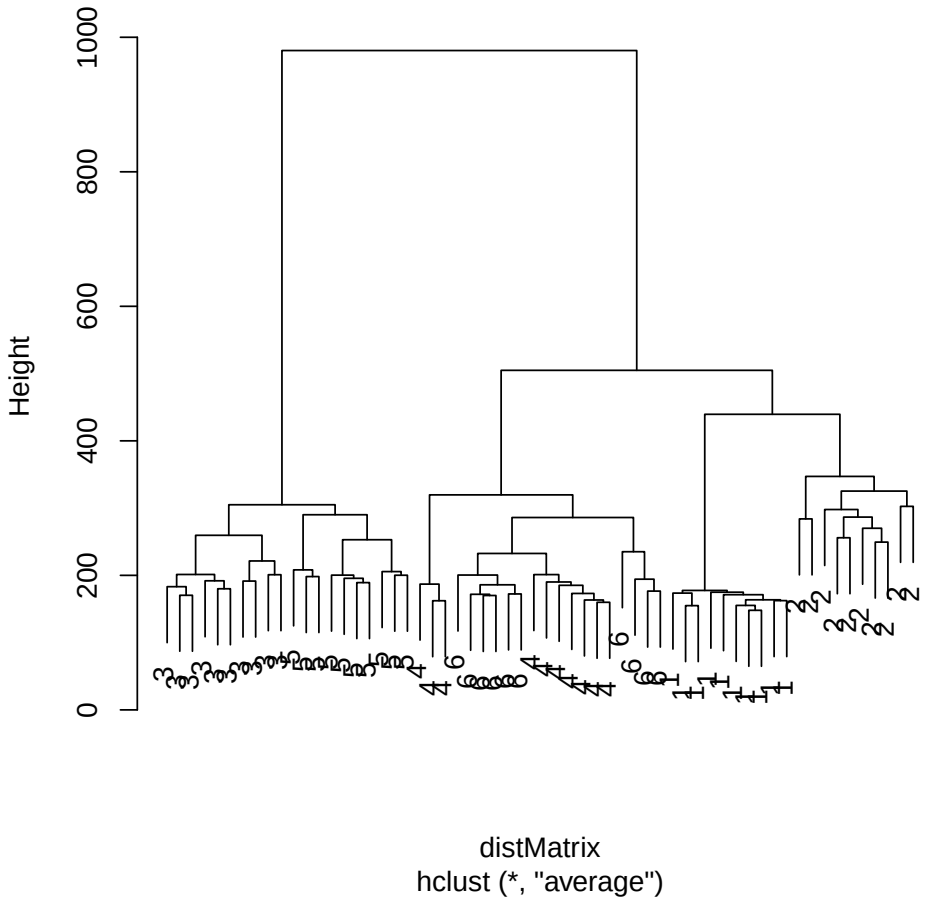

Figure 8.8: Hierarchical Clustering with DTW Distance

8.5 Time Series Classification

Time series classification is to build a classification model based on labeled time series and then use the model to predict the label of unlabeled time series.

Some techniques for feature extraction are Singular Value Decomposition (SVD), Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT), Piecewise Aggregate Approximation (PAA), Perpetually Important Points (PIP), Piecewise Linear Representation, and Symbolic Representation.

8.5.1 Classification with Original Data

We use `ctree` from package `party` [Hothorn et al., 2010] to demonstrate classification of time series with original data. The class labels are changed into categorical values before feeding the data into `ctree`, so that we won't get class labels as a real number like 1.35.

```
> classId <- c(rep("1",100), rep("2",100), rep("3",100),
+              rep("4",100), rep("5",100), rep("6",100))
> newSc <- data.frame(cbind(classId, sc))
> library(party)
> ct <- ctree(classId ~ ., data=newSc,
+             controls = ctree_control(minsplit=30, minbucket=10, maxdepth=5))
> pClassId <- predict(ct)
> table(classId, pClassId)

       pClassId
classId   1   2   3   4   5   6
      1  97   0   0   0   0   3
      2   1  93   2   0   0   4
      3   0   0  96   0   4   0
      4   0   0   0 100   0   0
      5   4   0  10   0  86   0
      6   0   0   0  87   0  13

> # accuracy
> (sum(classId==pClassId)) / nrow(sc)

[1] 0.8083333

> plot(ct, ip_args=list(pval=FALSE), ep_args=list(digits=0))
```

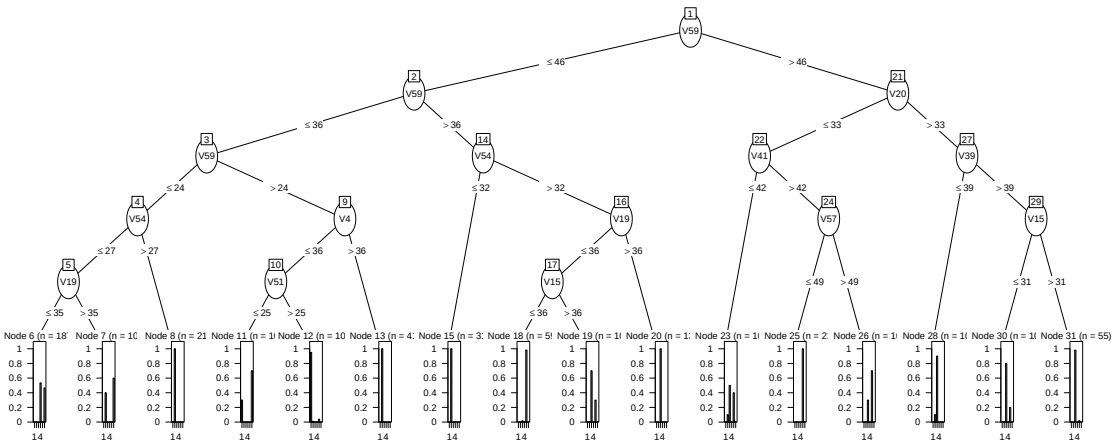

Figure 8.9: Decision Tree

8.5.2 Classification with Extracted Features

DWT (Discrete Wavelet Transform) [Burrus et al., 1998]

Wavelet transform provides a multi-resolution representation using wavelets. An example of Haar Wavelet Transform, the simplest DWT, is available at `http://dmr.ath.cx/gfx/haar/`. DFT (Discrete Fourier Transform) is another popular feature extraction technique [Agrawal et al., 1993].

An example on extracting DWT (with Haar filter) coefficients is shown below. Package *wavelets* [Aldrich, 2010] is used for discrete wavelet transform. In the package, function `dwt(X, filter, n.levels, ...)` computes the discrete wavelet transform coefficients, where `X` is a univariate or multi-variate time series, `filter` indicates which wavelet filter to use, and `n.levels` specifies the level of decomposition. It returns an object of class `dwt`, whose slot `W` contains wavelet coefficients and `V` contains scaling coefficients. The original time series can be reconstructed via an inverse

discrete wavelet transform with function `idwt()` in the same package.

```
> library(wavelets)
> wtData <- NULL
> for (i in 1:nrow(sc)) {
+   a <- t(sc[i,])
+   wt <- dwt(a, filter="haar", boundary="periodic")
+   wtData <- rbind(wtData,
+   unlist(c(wt@W, wt@V[[wt@level]])))
+ }
> wtData <- as.data.frame(wtData)
> wtSc <- data.frame(cbind(classId, wtData))
```

Decision Tree with DWT

```
> ct <- ctree(classId ~ ., data=wtSc,
+             controls = ctree_control(minsplit=30, minbucket=10, maxdepth=5))
> pClassId <- predict(ct)
> table(classId, pClassId)
```

```
       pClassId
classId  1  2  3  4  5  6
      1 97  3  0  0  0  0
      2  1 99  0  0  0  0
      3  0  0 81  0 19  0
      4  0  0  0 63  0 37
      5  0  0 16  0 84  0
      6  0  0  0  1  0 99
```

```
> (sum(classId==pClassId)) / nrow(wtSc)
```

```
[1] 0.8716667
```

```
> plot(ct, ip_args=list(pval=FALSE), ep_args=list(digits=0))
```

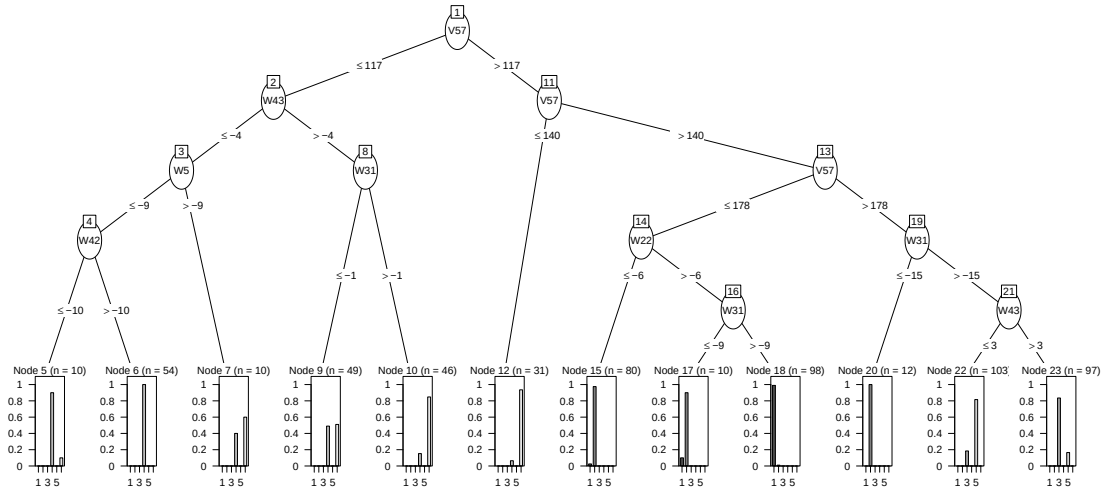

Figure 8.10: Decision Tree with DWT

8.5.3 k-NN Classification

k-NN Classification find the k nearest neighbors of a new instance; label it by majority voting; needs an ecient indexing structure for large datasets

```
> k <- 20
> newTS <- sc[501,] + runif(100)*15
> distances <- dist(newTS, sc, method="DTW")
> s <- sort(as.vector(distances), index.return=TRUE)
> # class IDs of k nearest neighbors
> table(classId[s$ix[1:k]])

 4  6
 3 17
```

For the 20 nearest neighbors of the new time series, three of them are of class 4, and 17 are of class 6. With majority voting, that is, taking the more frequent label as winner, the label of the new time series is set to class 6.

8.6 Discussion

There are many R functions and packages available for time series decomposition and forecasting. However, there are no R functions or packages specially time series classification and clustering. There are a lot of research publications on techniques specially for classifying/clustering time series data, but no R implementations (as far as I know).

To do time series classification, one is suggested to extract and build features first, and then apply existing classification techniques, such as SVM, k-NN, neural networks, regression and decision trees, to the feature set.

For time series clustering, one needs to work out his/her own distance or similarity metrics, and then use existing clustering techniques, such as k-means or hierarchical clustering, to find clustering structure.

8.7 Further Readings

An introduction of R functions and packages for time series is available as *CRAN Task View: Time Series Analysis* at `http://cran.r-project.org/web/views/TimeSeries.html`.

R code examples for time series can be found in slides *Time Series Analysis and Mining with R* at `http://www.rdatamining.com/docs`.

Some further readings on time series representation, similarity, clustering and classification are [Agrawal et al., 1993, Burrus et al., 1998, Chan and Fu, 1999, Chan et al., 2003, Keogh and Pazzani, 1998, Keogh et al., 2000, Keogh and Pazzani, 2000, Mörchen, 2003, Rafiei and Mendelzon, 1998, Vlachos et al., 2003, Wu et al., 2000, Zhao and Zhang, 2006].

Chapter 9

Association Rules

This chapter is not available yet in this version.

Chapter 10

Sequential Patterns

This chapter is not available yet in this version.

Chapter 11

Text Mining

This chapter shows how to do text mining in R. Twitter text of @RDataMining are used as data to analyze. There are three packages used in the examples: *twitteR*, *tm* and *wordcloud*. Package *twitteR* provides access to Twitter data, *tm* provides functions for text mining, and *wordcloud* visualizes the result with a word cloud [1] .

11.1 Retrieving Text from Twitter

Twitter text is used in this chapter to demonstrate text mining. Tweets are be extracted from Twitter with the code below using `userTimeline()` in package *twitteR*.

Package *twitteR* can be found at `http://cran.r-project.org/bin/windows/contrib/r-old-release/`. Note that it was built under R version 2.13, and may not work under the latest version of R. Package *twitteR* depends on package *RCurl*, which is available at `http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.14/`. Another way to retrieve text from Twitter is using package *XML*, and an example on that is given at `http://heuristically.wordpress.com/2011/04/08/text-data-mining-twitter-r/`.

```
> library(twitteR)
> # retrieve the first 100 tweets (or all tweets if fewer than 100)
> # from the user timeline of @rdatammining
> rdmTweets <- userTimeline("rdatamining", n=100)
> n <- length(rdmTweets)
> rdmTweets[1:3]
```

The above code can print the first three tweets, but each tweet is printed in one single line, which may exceed the boundary of paper. Therefore, the following code is used in this book to print the first three tweets by wrapping the text to fit the width of paper.

```
> for (i in 1:3) {
+   cat(paste("[[", i, "]]\n", sep=""))
+   writeLines(strwrap(rdmTweets[[i]]$getText(), 80))
+   cat("\n")
+ }

[[1]]
Text Mining Tutorial http://t.co/jPHHLEGm

[[2]]
R cookbook with examples http://t.co/aVtIaSEg
```

[1] `http://en.wikipedia.org/wiki/Word_cloud`

```
[[3]]
Access large amounts of Twitter data for data mining and other tasks within R
via the twitteR package. http://t.co/ApbAbnxs
```

11.2 Transforming Text

The tweets are first converted to a data frame and then to a corpus.

```
> # convert tweets to a data frame
> df <- do.call("rbind", lapply(rdmTweets, as.data.frame))
> dim(df)

[1] 79 10

> library(tm)
> # build a corpus, which is a collection of text documents
> # VectorSource specifies that the source is character vectors.
> myCorpus <- Corpus(VectorSource(df$text))
```

After that, the corpus needs a couple of transformations, including changing letters to lower case, removing punctuations/numbers and removing stop words. The general English stop-word list is tailored by adding "available" and "via" and removing "r".

```
> # convert to lower case
> myCorpus <- tm_map(myCorpus, tolower)
> # remove punctuation
> myCorpus <- tm_map(myCorpus, removePunctuation)
> # remove numbers
> myCorpus <- tm_map(myCorpus, removeNumbers)
> # remove stopwords; add two extra stop words;
> # keep "r" by removing it from stopwords
> myStopwords <- c(stopwords('english'), "available", "via")
> idx <- which(myStopwords == "r")
> myStopwords <- myStopwords[-idx]
> myCorpus <- tm_map(myCorpus, removeWords, myStopwords)
```

11.3 Stemming Words

In many cases, words need to be stemmed to retrieve their radicals. For instance, "example" and "examples" are both stemmed to "exampl". However, after that, one may want to complete the stems to their original forms, so that the words would look "normal".

```
> # keep a copy of corpus to serve as a dictionary for stemCompletion
> dictCorpus <- myCorpus
> # stem words in a text document with the snowball stemmers,
> # which requires packages Snowball, RWeka, rJava, RWekajars
> myCorpus <- tm_map(myCorpus, stemDocument)
> # inspect the first three documents (tweets)
> inspect(myCorpus[1:3])

A corpus with 3 text documents

The metadata consists of 2 tag-value pairs and a data frame
```

```
Available tags are:
  create_date creator
Available variables in the data frame are:
  MetaID

[[1]]
text mine tutori
httptcojphhlegm

[[2]]
r cookbook  exampl httptcoavtiaseg

[[3]]
access  amount  twitter data  data mine   task  r   twitter packag httptcoapbabnx
```

```
> # stem completion
> myCorpus <- tm_map(myCorpus, stemCompletion, dictionary=dictCorpus)
```

Print the first three documents in the built corpus.

```
> inspect(myCorpus[1:3])
```

```
[[1]]
text miners tutorial httptcojphhlegm

[[2]]
r cookbook examples httptcoavtiaseg

[[3]]
access amounts twitter data data miners task r twitter package httptcoapbabnxs
```

Something unexpected in the above stemming and stem completion is that, word "mining" is first stemmed to "mine", and then is completed to "miners", instead of "mining", although there are many instances of "mining" in the tweets, compared to only one instance of "miners".

11.4 Building a Document-Term Matrix

```
> # build a term-document matrix
> # set minWordLength = 1 to keep "r" in the term-document matrix
> myDtm <- TermDocumentMatrix(myCorpus, control = list(minWordLength = 1))
> # inspect the document-term matrix
> inspect(myDtm[266:270,31:40])
```

```
A term-document matrix (5 terms, 10 documents)

Non-/sparse entries: 9/41
Sparsity           : 82%
Maximal term length: 12
Weighting          : term frequency (tf)

              Docs
Terms          31 32 33 34 35 36 37 38 39 40
  r             0  0  1  1  1  0  1  2  1  0
  ramachandran  0  0  0  0  0  0  1  0  0  0
  ranked        0  0  0  1  0  0  0  0  0  0
```

```
  rapidminer    0  0  0  0  0  0  0  0  0  0
  rdatamining   0  0  1  0  0  0  0  0  0  0

> #rownames(myDtm) shows a list of terms.
```

Based on the above matrix, many data mining tasks can be done, for example, clustering, classification and association analysis.

11.5 Frequent Terms and Associations

```
> # inspect most popular words
> findFreqTerms(myDtm, lowfreq=10)

[1] "analysis" "data"     "examples" "miners"   "package"  "r"        "slides"
[8] "tutorial" "users"

> # which words are associated with "r"?
> findAssocs(myDtm, 'r', 0.30)

       r    users examples  package canberra     cran     list
    1.00     0.44     0.34     0.31     0.30     0.30     0.30

> # which words are associated with "mining"?
> # Here "miners" is used instead of "mining",
> # because the latter is stemmed and then completed to "miners". :-)
> findAssocs(myDtm, 'miners', 0.30)

        miners           data classification   httptcogbnpv         mahout
          1.00           0.56           0.47           0.47           0.47
recommendation           sets       supports       frequent        itemset
          0.47           0.47           0.47           0.40           0.39

> # findAssocs(myDtm, 'mining', 0.30)
```

11.6 Word Cloud

After building a document-term matrix, we can show the importance of words with a word cloud (also kown as a tag cloud) . In the code below, word "miners" are changed back to "mining".

```
> # word cloud
> library(wordcloud)
> m <- as.matrix(myDtm)
> # calculate the frequency of words
> v <- sort(rowSums(m), decreasing=TRUE)
> myNames <- names(v)
> k <- which(names(v)=="miners")
> myNames[k] <- "mining"
> d <- data.frame(word=myNames, freq=v)
> # d <- data.frame(word=names(v), freq=v)
> wordcloud(d$word, d$freq, min.freq=3)
```

Figure 11.1: Word Cloud of @RDataMining Tweets

The above word cloud clearly shows that "r", "data" and "mining" are the three most important words, which validates that the @RDataMining tweets present information on R and data mining. The other important words are "analysis", "examples", "slides", "tutorial" and "package", which shows that it focuses on documents and examples on analysis and R packages.

11.7 Clustering Words

We then try to find clusters of words with hierarchical clustering.

```
> # remove sparse terms
> myDtm2 <- removeSparseTerms(myDtm, sparse=0.95)
> m <- as.matrix(myDtm2)
> # cluster terms
> d <- dist(scale(m))
> fit <- hclust(d, method="ward")
> plot(fit)
> # cut tree into 5 clusters
> groups <- cutree(fit, k=5)
> rect.hclust(fit, k=5)
```

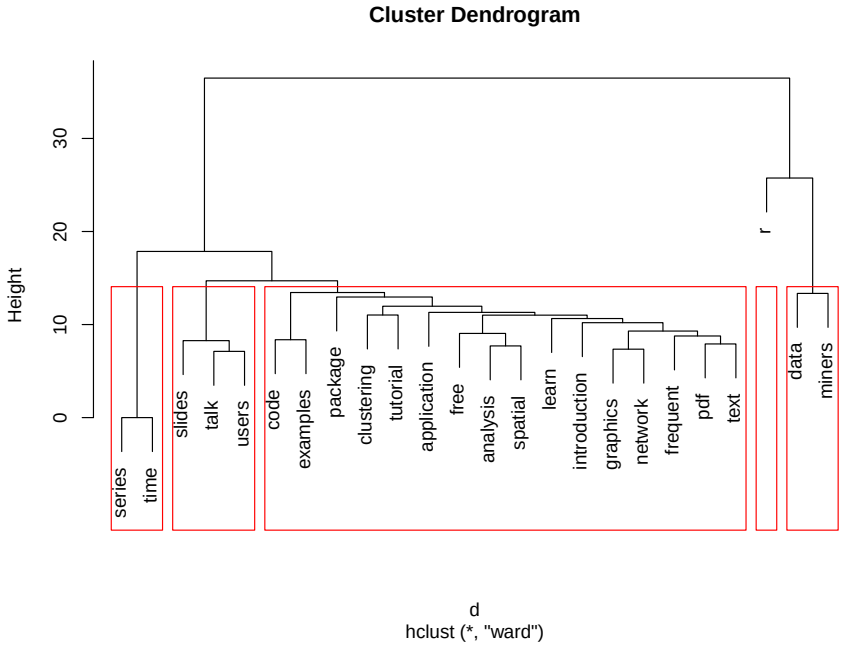

Figure 11.2: Clustering of Words in @RDataMining Tweets

Words "series" and "time" are clustered into one group, because there are a couple of tweets on time series analysis. The second cluster from left comprises "slides", "talk" and "users", which could be because of tweets on slides and talks at R users groups. The two rightmost clusters consists of "r", "data" and "mining", which are the keywords of @RDataMining tweets.

11.8 Clustering Tweets

Tweets are clustered below with k-means and k-medoids algorithms. We first try k-means clustering, which takes the values in the matrix as numeric ones.

```
> m2 <- t(m) # transpose the matrix to cluster documents (tweets)
> #k-means clustering of tweets
> kmeansResult <- kmeans(m2, 8)
> # output the three most important words in every cluster
> for (i in 1:8) {
+   cat(paste("cluster", i, ":  "))
+   sorted <- sort(kmeansResult$centers[i,], decreasing=T)
+   #cat(names(sorted)[sorted>0.4])
+   cat(names(sorted)[1:3])
```

```
+   cat("\n")
+   #print(rdmTweets[which(kmeansResult$cluster==i)])
+ }

cluster 1 :  tutorial data miners
cluster 2 :  data miners r
cluster 3 :  data r analysis
cluster 4 :  r analysis miners
cluster 5 :  r package examples
cluster 6 :  r slides code
cluster 7 :  network analysis free
cluster 8 :  clustering introduction miners

> # cluster centers
> kmeansResult$centers

    analysis application clustering       code     data  examples       free
1 0.00000000  0.00000000 0.25000000 0.00000000 0.750000 0.0000000 0.00000000
2 0.00000000  0.18750000 0.06250000 0.06250000 1.437500 0.1250000 0.06250000
3 0.50000000  0.00000000 0.00000000 0.00000000 1.166667 0.1666667 0.16666667
4 1.00000000  0.00000000 0.00000000 0.33333333 0.000000 0.3333333 0.00000000
5 0.12500000  0.00000000 0.12500000 0.12500000 0.000000 0.3750000 0.00000000
6 0.05555556  0.05555556 0.05555556 0.22222222 0.000000 0.2222222 0.05555556
7 0.66666667  0.00000000 0.00000000 0.00000000 0.000000 0.0000000 0.33333333
8 0.04761905  0.04761905 0.14285714 0.04761905 0.000000 0.0000000 0.09523810
   frequent  graphics introduction      learn    miners   network   package
1 0.0000000 0.0000000    0.0000000 0.00000000 0.7500000 0.0000000 0.0000000
2 0.1250000 0.0000000    0.0625000 0.06250000 1.3125000 0.0000000 0.1875000
3 0.0000000 0.1666667    0.1666667 0.00000000 0.0000000 0.0000000 0.1666667
4 0.0000000 0.0000000    0.0000000 0.00000000 1.0000000 0.0000000 0.0000000
5 0.0000000 0.0000000    0.1250000 0.25000000 0.0000000 0.0000000 0.5000000
6 0.0000000 0.1666667    0.0000000 0.05555556 0.1111111 0.1111111 0.1111111
7 0.0000000 0.3333333    0.0000000 0.00000000 0.3333333 1.0000000 0.0000000
8 0.0952381 0.0000000    0.1428571 0.04761905 0.1428571 0.0000000 0.0952381
         pdf         r     series    slides   spatial      talk      text
1 0.25000000 0.0000000 0.00000000 0.2500000 0.2500000 0.0000000 0.2500000
2 0.00000000 0.6250000 0.00000000 0.0625000 0.0000000 0.0000000 0.0625000
3 0.16666667 0.8333333 0.16666667 0.0000000 0.3333333 0.0000000 0.0000000
4 0.00000000 1.6666667 1.00000000 0.3333333 0.0000000 1.0000000 0.0000000
5 0.00000000 2.0000000 0.00000000 0.0000000 0.1250000 0.0000000 0.0000000
6 0.05555556 1.0000000 0.05555556 0.2777778 0.0000000 0.1111111 0.1111111
7 0.00000000 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000 0.3333333
8 0.09523810 0.0000000 0.14285714 0.1428571 0.0000000 0.0952381 0.0000000
        time   tutorial     users
1 0.00000000 1.25000000 0.0000000
2 0.00000000 0.00000000 0.1250000
3 0.16666667 0.00000000 0.1666667
4 1.00000000 0.00000000 0.6666667
5 0.00000000 0.37500000 0.2500000
6 0.05555556 0.05555556 0.1666667
7 0.00000000 0.33333333 0.0000000
8 0.14285714 0.00000000 0.0000000
```

We then try k-medoids clustering, which takes the values in the matrix as categoric ones.

```
> # m2[1:10,1:5]
> m3 <- m2
> m3[m3>0] <- 1
> # m3[1:10,1:5]
> library(fpc)
> clusterNum <- 8
> NA

[1] NA

> result <- pam(m3, clusterNum)
> # print cluster medoids
> for (i in 1:clusterNum) {
+   cat(paste("cluster", i, ":  "))
+   cat(colnames(result$medoids)[which(result$medoids[i,]==1)])
+   cat("\n")
+ }

cluster 1 :  frequent miners
cluster 2 :  r
cluster 3 :  data miners r
cluster 4 :
cluster 5 :  data miners
cluster 6 :  package r
cluster 7 :  r slides talk users
cluster 8 :  analysis miners r series talk time users
```

Some other analysis can be done are graph mining and social network analysis. For example, a graph of words can be derived from a document-term matrix, and then we can use techniques for graph mining to find links between words and groups of words. A graph of tweets (documents) can also be generated and analyzed in a similar way. It can also be presented and analyzed as a bipartite graph with two disjoint sets of vertices, that is, words and tweets.

11.9 Packages and Further Readings

Package *tm* []: A framework for text mining applications within R.

Package *tm.plugin.mail* []: Text Mining E-Mail Plug-In. A plug-in for the tm text mining framework providing mail handling functionality.

Package *textcat* [] provides n-Gram Based Text Categorization.

Introduction to the tm Package – Text Mining in R `http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf`

Text Mining Infrastructure in R `http://www.jstatsoft.org/v25/i05`

Chapter 12

Social Network Analysis

This chapter is not available yet in this version.

Chapter 13

Case Study I: Analysis and Forecasting of House Price Indices

This chapter and the other case studies are not available in this version. They are reserved exclusively for a book version to be published by Elsevier Inc.

Chapter 14

Case Study II: Customer Response Prediction

This chapter and the other case studies are not available in this version. They are reserved exclusively for a book version to be published by Elsevier Inc.

Chapter 15

Online Resources

There are many free online resources on R and data mining, and some of them are listed below.

15.1 R Reference Cards

- *R Reference Card*, by Tom Short
  http://rpad.googlecode.com/svn-history/r76/Rpad_homepage/R-refcard.pdf
- *R Reference Card for Data Mining*
  http://www.rdatamining.com/docs
- *R Reference Card*, by Jonathan Baron
  http://cran.r-project.org/doc/contrib/refcard.pdf
- *R Functions for Regression Analysis*, by Vito Ricci
  http://cran.r-project.org/doc/contrib/Ricci-refcard-regression.pdf
- *R Functions for Time Series Analysis*, by Vito Ricci
  http://cran.r-project.org/doc/contrib/Ricci-refcard-ts.pdf

15.2 R

- *Quick-R*
  http://www.statmethods.net/
- *R Tutorial*
  http://www.cyclismo.org/tutorial/R/index.html
- The R Manuals, including *an Introduction to R*, *R Language Definition*, *R Data Import/Export*, and other R manuals
  http://cran.r-project.org/manuals.html
- *R for Beginners*
  http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf
- *Econometrics in R*
  http://cran.r-project.org/doc/contrib/Farnsworth-EconometricsInR.pdf
- *Using R for Data Analysis and Graphics - Introduction, Examples and Commentary*
  http://www.cran.r-project.org/doc/contrib/usingR.pdf
- Lots of R Contributed Documents, including non-English ones
  http://cran.r-project.org/other-docs.html

- *The R Journal*
  http://journal.r-project.org/current.html

15.3 Data Mining

- *Introduction to Data Mining*, by Pang-Ning Tan, Michael Steinbach and Vipin Kumar
  Lecture slides (in both PPT and PDF formats) and three sample chapters on classification, association and clustering available at the link below.
  http://www-users.cs.umn.edu/%7Ekumar/dmbook
- *Mining of Massive Datasets*, by Anand Rajaraman and Jeff Ullman
  The whole book and lecture slides are free and downloadable in PDF format.
  http://infolab.stanford.edu/%7Eullman/mmds.html
- Lecture notes of data mining course, by Cosma Shalizi at CMU
  R code examples are provided in some lecture notes, and also in solutions to home works.
  http://www.stat.cmu.edu/%7Ecshalizi/350/
- Tutorial on Spatial and Spatio-Temporal Data Mining
  http://www.inf.ufsc.br/%7Evania/tutorial_icdm.html
- Tutorial on Discovering Multiple Clustering Solutions
  http://dme.rwth-aachen.de/en/DMCS
- *A Complete Guide to Nonlinear Regression*
  http://www.curvefit.com/
- A paper on *Open-Source Tools for Data Mining*, published in 2008
  http://eprints.fri.uni-lj.si/893/1/2008-OpenSourceDataMining.pdf

15.4 Data Mining with R

- *Data Mining with R - Learning by Case Studies*
  http://www.liaad.up.pt/~ltorgo/DataMiningWithR/
- *Data Mining Algorithms In R*
  http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R
- *Data Mining Desktop Survival Guide*
  http://www.togaware.com/datamining/survivor/

15.5 Decision Trees

- *An Introduction to Recursive Partitioning Using the RPART Routines*
  http://www.mayo.edu/hsr/techrpt/61.pdf

15.6 Time Series Analysis

- *An R Time Series Tutorial*
  http://www.stat.pitt.edu/stoffer/tsa2/R_time_series_quick_fix.htm
- *Time Series Analysis with R*
  http://www.statoek.wiso.uni-goettingen.de/veranstaltungen/zeitreihen/sommer03/ts_r_intro.pdf

- *Using R (with applications in Time Series Analysis)*
  http://people.bath.ac.uk/masgs/time%20series/TimeSeriesR2004.pdf
- *CRAN Task View: Time Series Analysis*
  http://cran.r-project.org/web/views/TimeSeries.html
- *Time Series Analysis for Business Forecasting*
  http://home.ubalt.edu/ntsbarsh/stat-data/Forecast.htm

15.7 Spatial Data Analysis

- *Applied Spatio-temporal Data Analysis with FOSS: R+OSGeo*
  http://www.geostat-course.org/GeoSciences_AU_2011

15.8 Text Mining

- *Text Mining Infrastructure in R*
  http://www.jstatsoft.org/v25/i05
- *Introduction to the tm Package Text Mining in R*
  http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf
- *Text Mining Handbook* (with R code examples)
  http://www.casact.org/pubs/forum/10spforum/Francis_Flynn.pdf
- *Distributed Text Mining in R*
  http://epub.wu.ac.at/3034/

15.9 Regression

- *A Complete Guide to Nonlinear Regression*
  http://www.curvefit.com/

Bibliography

[Agrawal et al., 1993] Agrawal, R., Faloutsos, C., and Swami, A. N. (1993). Efficient similarity search in sequence databases. In Lomet, D., editor, *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, Chicago, Illinois. Springer Verlag. DFT.

[Aldrich, 2010] Aldrich, E. (2010). wavelets: A package of funtions for computing wavelet filters, wavelet transforms and multiresolution analyses. http://cran.r-project.org/web/packages/wavelets/index.html.

[Burrus et al., 1998] Burrus, C. S., Gopinath, R. A., and Guo, H. (1998). *Introduction to Wavelets and Wavelet Transforms: A Primer*. Prentice-Hall, Inc. DWT.

[Chan et al., 2003] Chan, F. K., Fu, A. W., and Yu, C. (2003). Harr wavelets for efficient similarity search of time-series: with and without time warping. *IEEE Trans. on Knowledge and Data Engineering*, 15(3):686–705.

[Chan and Fu, 1999] Chan, K.-p. and Fu, A. W.-c. (1999). Efficient time series matching by wavelets. In *Internation Conference on Data Engineering (ICDE '99)*, Sydney.

[Cleveland et al., 1990] Cleveland, R. B., Cleveland, W. S., McRae, J. E., and Terpenning, I. (1990). Stl: a seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–73.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231.

[Frank and Asuncion, 2010] Frank, A. and Asuncion, A. (2010). UCI machine learning repository. university of california, irvine, school of information and computer sciences. http://archive.ics.uci.edu/ml.

[Giorgino, 2009] Giorgino, T. (2009). Computing and visualizing dynamic timewarping alignments in R: The dtw package. *Journal of Statistical Software*, 31(7):1–24.

[Han and Kamber, 2000] Han, J. and Kamber, M. (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Hand et al., 2001] Hand, D. J., Mannila, H., and Smyth, P. (2001). *Principles of Data Mining (Adaptive Computation and Machine Learning)*. The MIT Press.

[Hothorn et al., 2010] Hothorn, T., Hornik, K., Strobl, C., and Zeileis, A. (2010). Party: A laboratory for recursive partytioning. http://cran.r-project.org/web/packages/party/.

[Keogh et al., 2000] Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. (2000). Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286.

[Keogh and Pazzani, 1998] Keogh, E. J. and Pazzani, M. J. (1998). An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD 1998*, pages 239–243.

[Keogh and Pazzani, 2000] Keogh, E. J. and Pazzani, M. J. (2000). A simple dimensionality reduction technique for fast similarity search in large time series databases. In *PAKDD*, pages 122–133.

[Keogh and Pazzani, 2001] Keogh, E. J. and Pazzani, M. J. (2001). Derivative dynamic time warping. In *the 1st SIAM Int. Conf. on Data Mining (SDM-2001)*, Chicago, IL, USA.

[Mörchen, 2003] Mörchen, F. (2003). Time series feature extraction for data mining using DWT and DFT. Technical report, Departement of Mathematics and Computer Science Philipps-University Marburg. DWT & DFT.

[R Development Core Team, 2010a] R Development Core Team (2010a). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

[R Development Core Team, 2010b] R Development Core Team (2010b). *R Data Import/Export*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-10-0.

[R Development Core Team, 2010c] R Development Core Team (2010c). *R Language Definition*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-13-5.

[Rafiei and Mendelzon, 1998] Rafiei, D. and Mendelzon, A. O. (1998). Efficient retrieval of similar time sequences using DFT. In Tanaka, K. and Ghandeharizadeh, S., editors, *FODO*, pages 249–257.

[Therneau et al., 2010] Therneau, T. M., Atkinson, B., and Ripley, B. (2010). *rpart: Recursive Partitioning*. R package version 3.1-46.

[Venables et al., 2010] Venables, W. N., Smith, D. M., and R Development Core Team (2010). *An Introduction to R*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-12-7.

[Vlachos et al., 2003] Vlachos, M., Lin, J., Keogh, E., and Gunopulos, D. (2003). A wavelet-based anytime algorithm for k-means clustering of time series. In *Workshop on Clustering High Dimensionality Data and Its Applications, at the 3rd SIAM International Conference on Data Mining*, San Francisco, CA, USA.

[Witten and Frank, 2005] Witten, I. and Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, CA., USA, second edition.

[Wu et al., 2000] Wu, Y.-l., Agrawal, D., and Abbadi, A. E. (2000). A comparison of DFT and DWT based similarity search in time-series databases. In *Proceedings of the 9th ACM CIKM Int'l Conference on Informationand Knowledge Management*, pages 488–495, McLean, VA.

[Zhao and Zhang, 2006] Zhao, Y. and Zhang, S. (2006). Generalized dimension-reduction framework for recent-biased time series analysis. *IEEE Transactions on Knowledge and Data Engineering*, 18(2):231–244.

Index