

Data Mining with Rattle

A Graphical Interface to R

Open Source Desktop Survival Guide

Togaware Watermark For TeX Catalogue Survival

Togaware Series of Open Source Desktop Survival Guides

This innovative series presents open source and freely available software tools and techniques for a variety of tasks, ranging from working with the GNU/Linux operating system, through common desktop productivity tools, to sophisticated data mining applications. Each volume aims to be self contained, and slim, presenting the information in an easy to follow format without overwhelming the reader with details.

Togaware Watermark For TeX Catalogue Survival

Series Titles

Data Mining with Rattle: A GUI for Two Class Prediction

R for the Data Miner: A Guide for Interacting with R

Text Mining with Rattle: A GUI for Text Mining in R

Data Mining with Rattle

A Graphical Interface to R

Open Source Desktop Survival Guide

Graham Williams
Togaware.com

The procedures and applications presented in this book have been included for their instructional value. They have been tested but are not guaranteed for any particular purpose. Neither the publisher nor the author offer any warranties or representations, nor do they accept any liabilities with respect to the programs and applications.

The book, as you see it presently, is a work in progress, and different sections are progressed depending on feedback. Please send comments, suggestions, updates, and criticisms to Graham.Williams@togaware.com.

I hope you find it useful!

Togaware Watermark For TeX Catalogue Survival

Printed 27th March 2007

Copyright © 2006-2007 by Graham Williams

ISBN 0-9757109-2-3

Togaware Watermark For TeX Catalogue Survival

*Where knowledge is power, data is the fuel and data mining
the engine room for delivering that knowledge.*

Togaware Watermark For TeX Catalogue Survival

Preface

Knowledge leads to wisdom and better understanding. Data mining builds knowledge from information, adding value to the tremendous stores of data that abound today—stores that are ever increasing in size and availability. Emerging from the database community in the late 1980's the discipline of data mining grew quickly to encompass researchers and technologies from Machine Learning, High Performance Computing, Visualisation, and Statistics, recognising the growing opportunity to add value to data. Today, this multi-disciplinary effort continues to deliver new techniques and tools for the analysis of very large collections of data. Searching through databases measuring in the gigabytes and terabytes, data mining delivers discoveries that improve the way an organisation does business. It can enable companies to remain competitive in this modern data rich, knowledge hungry, wisdom scarce world. Data mining delivers knowledge to drive the getting of wisdom.

The range of techniques and algorithms used in data mining may appear daunting and overwhelming. In performing data mining for a data rich client many decisions need to be made regarding the choice of methodology, the choice of data, the choice of tools, and the choice of application.

In this book we deploy the Free and Open Source Software package Rattle, which is built on top of the R system to illustrate the deployment of data mining technology. As Free Software the source code of Rattle and R is available to anyone, and anyone is permitted, and indeed encouraged, to extend the software, and to read the source code to learn from it. Indeed, R is supported by a world wide network of some of the world's leading Statisticians.

Goals

This book presents a unique and easily accessible single stop resource for the data miner. It provides a practical guide to actually doing data mining. It is accessible to the information technology worker, the software engineer, and the data analyst. It also serves well as a textbook for an applications and techniques oriented course on data mining. While much data analysis and modelling relies on a foundation of statistics, the aim here is to not lose the reader in the statistical details. This presents a challenge! At times the presentation will leave the statistically sophisticated wanting a more solid treatment. In these cases the reader is referred to the excellent statistical expositions in [Dalgaard \(2002\)](#), [Venables and Ripley \(2002\)](#), and [Hastie et al. \(2001\)](#).

Organisation

Part [II](#) constitutes a complete guide to using Rattle for data mining.

In Chapter [2](#) we introduce Rattle as a graphical user interface (GUI) developed for making any data mining project a lot simpler. This covers the installation of both R and Rattle, as well as basic interaction with Rattle.

Chapters [3](#) to [12](#) then detail the steps of the data mining process, corresponding to the straightforward interface presented through Rattle. We describe how to get data into Rattle, how to select variables, and how to perform sampling in Chapter [3](#). Chapter [4](#) then reviews various approaches to exploring the data in order for us to gain some insights about the data we are looking at as well as understanding the distribution of the data and to assess the appropriateness of any modelling.

Chapters [8](#) to [11](#) cover modelling, including descriptive and predictive modelling, and text mining. The evaluation of the performance of the models and their deployment is covered in Chapter [12](#). Chapter [13](#) provides an introduction to migrating from Rattle to the underlying R system. It does not attempt to cover all aspects of interacting with R but is sufficient for a competent programmer or software engineer to be able to extend and further fine tune the modelling performed in Rattle. Chap-

ter 14 covers troubleshooting within Rattle.

Part III delves much deeper into the use of R for data mining. In particular, R is introduced as a programming language for data mining. Chapter 15 introduces the basic environment of R. Data and data types are covered in Chapter 16 and R's extensive capabilities in producing stunning graphics is introduced in Chapter 17. We then pull together the capabilities of R to help us understand data in Chapter 18. We then move on to preparing our data for data mining in Chapter 19, building models in Chapter 20, and evaluating our models in Chapter 22.

Part IV reviews the algorithms employed in data mining. The encyclopedic type overview covers many tools and techniques deployed within data mining, ranging from decision tree induction and association rules, to multivariate adaptive regression splines and patient rule induction methods. We also cover standards for sharing data and models.

We continue the Desktop Guide with a snapshot of some current alternative open source and then commercial data mining products in Part V, *Open Source Products*, and Part VI, *Commercial Off The Shelf Products*.

Features

A key feature of this book, that differentiates it from many other very good textbooks on data mining, is the focus on the end-to-end process for data mining. That is, we cover in quite some detail the business understanding, data, modelling, evaluation, and practical deployment. In addition to presenting descriptions of approaches and techniques for data mining using modern tools, we provide a very practical resource with actual examples using Rattle. These will be immediately useful in delivering on data mining projects. We have chosen an easy to use yet very powerful open source software for the examples presented in the book. Anyone can obtain the appropriate software for free from the Internet and quickly set up their computer with a very sophisticated data mining environment, whether they use GNU/Linux, Unix, Mac/OSX, or even MS/Windows.

Audience

The book is accessible to many readers and not necessarily just those with strong backgrounds in computer science or statistics. At times we do introduce some statistical, mathematical, and computer science notations, but intentionally keep it simple. Sometimes this means oversimplifying concepts, but only where it does not lose intent of the concept and only where it retains its fundamental accuracy.

Typographical Conventions

We use the R language to illustrate concepts and modelling in data mining. R is both a programming language and an interpreter. When we illustrate interactive sessions with R we will generally show the R prompt, which by default is “>”. This allows the output from R to more easily be distinguished. However, we generally do not include the continuation prompt (a “+”) which appears when a single command extends over multiple lines, as this makes it more difficult to cut-and-paste from the examples in the electronic version of the book.

In providing example output from commands, at times we will truncate the listing and indicate missing components with [...]. While most examples will illustrate the output exactly as it appears in R there will be times where the format will be modified slightly to fit with publication limitations. This might involve silently removing or adding blank lines.

Acknowledgements

Many thanks to my many students from the Australian National University over the years who have been the reason for me to collect my thoughts and experiences with data mining to bring together into this book. I have benefited from their insights into how they learn best. They have also contributed in a number of ways with suggestions and example applications. I am also in debt to my colleagues over the years, particularly Peter Milne and Zhexue Huang, for their support and contributions over the years in the development of Data Mining in Australia.

Colleagues in various organisations deploying or developing skills in data mining have also provided significant feedback, as well as the motivation, for this book. These include, but are not limited to my Australian Taxation Office colleagues, Stuart Hamilton, Frank Lu, Anthony Nolan, Peter Ricci, Shawn Wicks, and Robert Williams.

This book has grown from a desire to share experiences in using and deploying data mining tools and techniques. A considerable proportion of the material draws on over ten years of teaching data mining to undergraduate and graduate students and running industry outreach courses. The aim is to provide recipe type material that can be instantly deployed, as well as reference material covering the concepts and terminology a data miner is likely to come across.

Many have contributed to the content of the book, providing insights and comments. Illustrative examples of using R have also come from the R mailing lists and I have used many of these to guide the kinds of examples that are included in the book. Many contributors to that list need to be thanked, and include Earl F. Glynn.

Financial support for maintenance of the book is always welcome. Financial support is used to contribute toward the costs of running the web pages and the desktop machine used to make this book available.

I acknowledge the support of many, including: Milton Cabral, Gail McEwen, Wade Thunborg, Longbing Cao, Martin Schultz, Danilo Cilario, Toyota Finance, Michael Stigall, Melanie Hilario, Siva Ganesh, Myra O'Regan, Stephen Zelle, Welling Howell, Adam Weisberg, Takaharu Araki, Caroline Rodriguez, Patrick L Durusau, Menno Bot, Dorene A Gilyard, Henry Walker, Fei Huang, Di Peter Kralicek, Lo Siu Keung, Julian Leslie, Mona Habib, John Chow, Michael Provost, Hamish R Hutchison, Chris Raymond, Keith Lyons, Shawn Swart, Hubert Weikert, and Tom Thomas.

Graham J Williams

Canberra

Togaware Watermark For TeX Catalogue Survival

Chapter 2

Introduction

Rattle (the R Analytical Tool To Learn Easily) is a graphical data mining application built upon the statistical language [R](#). An understanding of R is not required in order to use Rattle. However, a basic introduction is provided in [Chapter 15](#). Rattle is simple to use, quick to deploy, and allows us to rapidly work through the modelling phase of a data mining project. R, on the other hand, provides a very powerful language for performing data mining, and when we need to fine tune our data mining projects we can always migrate from Rattle to R simply by taking Rattle's underlying commands and deploying them within the R console.

Rattle uses the Gnome graphical user interface and runs under various operating systems, including GNU/Linux, Macintosh OS/X, and MS/Windows. Its intuitive user interface takes us through the basic steps of data mining, as well as illustrating (through a log textview) the actual R code that is used to achieve this. The R code itself can actually be saved to file and used as an automatic script which can be loaded into R (outside of Rattle) to repeat any data mining exercise.

While Rattle by itself may be sufficient for all of a user's needs, it also provides a stepping stone to more sophisticated processing and modelling in R itself. The user is not limited to my views and ideas about how things should be done. For sophisticated and unconstrained data mining, the experienced user can progress to interacting directly with a powerful language.

In this chapter we present the **Rattle** interface, and its basic environment for interaction, including menus and toolbars, and saving and loading projects. Chapter 3 works through the process of loading data into **Rattle** and Chapter 4 presents the various options within **Rattle** for exploring our data. We then go through the process of building models and evaluating the models in Chapters 7 to 12.

We start with the installation of **Rattle**.

2.1 Installing GTK, R, and Rattle

Rattle is distributed as an R package and is available from **CRAN**, the Comprehensive R Archive Network. The latest version is also available as an R package from **Togaware**. The source code is freely available from **Google Code**, where it is also possible to join the **Rattle** users mailing list.

The first step in installing **Rattle** is to install the GTK+ libraries, which provide the Gnome user interface. We need to install the correct package for our operating system. This installation is independent of the installation of R itself and is emphasised as a preliminary step that is often overlooked when installing **Rattle**.

If you are new to R there are just a few steps to get up and running with **Rattle**. If you are running on a Macintosh¹, be sure to run R from inside X11 (off the XCode CD) using the X11 shell to start R. Native Mac GTK+ is not fully supported. (You also need to use gcc 4.0.3, rather than the Mac's 4.0.1.) Be sure to install the glade libraries before installing RGtk2, since RGtk2 will ignore libraries that it can't find at the time of installation (e.g., you may find that `newGladeXML` is undefined).

1. *Install the GTK+ libraries*

For **GNU/Linux** these libraries will already be installed if you are running Gnome, but make sure you also have libglade installed. If you are not running Gnome you may need to install the GTK+ li-

¹Thanks to Daniel Harabor and Antony Unwin for the Mac/OSX information.

2.1 Installing GTK, R, and Rattle

9

braries in your distribution. For example, with the excellent **Debian GNU/Linux** distribution you can simply install the Glade package:

```
Debian: wajig install libglade2-0
```

For **MS/Windows**, install the latest version of the Glade package from the **Glade for Windows** website. Download the self-installing package (e.g., `gtk-win32-devel-2.8.10-rc1.exe`) and open it to install the libraries:

```
MS/Windows: run gtk-win32-devel-2.8.10-rc1.exe
```

An alternative that seems to work quite well (thanks to Andy Liaw for pointing this out) is to run a script that will install everything required for GTK+ on MS/Windows. This R script installs the *rggobi* package (and other things it depends on). You can start up R and then type the command:

```
source("http://www.ggobi.org/download/install.r")
```

This installs the GTK libraries for MS/Windows and the *rggobi* package for R. (You need R installed already of course - see the next step.) For **Mac/OSX**, make sure Apple X11 is installed on your machine as GTK (and anything built with it) is not native to OSX. Using darwinports (from <http://darwinports.opendarwin.org/>) you can install the packages:

```
Mac/OSX: $ sudo port install gtk2          (couple of hours)
Mac/OSX: $ sudo port install libglade2     (about 10 minutes)
```

For **All** installations, after installing libglade or any of the other libraries, be sure to restart the R console, if you have one running. This will ensure R can find the newly installed libraries.

2. *Install R*

For **GNU/Linux** R is packaged for many distributions. For example, on **Debian GNU/Linux** install the packages with:

```
Debian: $ wajig install r-recommended
```

For **MS/Windows** the binary distribution can be obtained from the **R Project** website. Download the self-installing package and open it.

```
MS/Windows: run R-2.4.0-win32.exe
```

For **Mac/OSX** you can also download the package from CRAN and install it.

For **All** installations, to check if you have R installed, start up a Terminal and enter the command `R` (that's just the capital letter R). If the response is that the command is not found, then you probably need to install the R application!

```
$ R

R version 2.4.1 (2006-12-18)
Copyright (C) 2006 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

3. *Install RGtk2*

This package is available on CRAN and from the **RGtk2** web site. From R, use:

```
R: > install.packages("RGtk2")
```

Or to install the most recent release:

```
R: > install.packages("RGtk2", repos="http://www.ggobi.org/r/")
```

On **GNU/Linux** you can generally just install the appropriate package. On Debian this is done with:

```
Debain: $ wajig install r-cran-gtk2
```


On **Mac/OSX** run the command line install, after downloading the source package from <http://www.ggobi.org/rgtk2/RGtk2.2.8.6.tar.gz>:

```
Mac/OSX: $ R CMD INSTALL RGtk2_2.8.6.tar.gz (30 minutes)
```

You may not be able to compile RGtk2 via the R GUI on Mac/OSX as the GTK libraries can not be found when gcc is called. Once installed though, R will detect the package; just don't try to load it within the GUI as GTK is not a native OSX application and it will break. On the Mac/OSX be sure to run R from X11.

For **ALL** installations, to test whether you have RGtk2 installed enter the R command

```
R: > library(RGtk2)
```

4. *Install R Packages*

The following additional R packages are used by Rattle, and without them some functionality will be missing. Rattle will gracefully handle them being missing so there is no need to install them all, or to install them all at once. If you perform an action where Rattle indicates a package is missing, you can then install the package. Type `?install.packages` at the R prompt for further help on installing packages.

```
R: > install.packages(c("ada", "amap", "arules", "bitops",
  "cairoDevice", "cba", "combinat", "doBy", "ellipse",
  "fEcofin", "fCalendar", "fBasics", "fpc",
  "gdata", "gtools", "gplots", "Hmisc", "kernlab",
  "MASS", "mice", "network", "pmml", "randomForest",
  "rggobi", "ROCR", "RODBC", "rpart", "RSvgDevice",
  "XML"))
```

5. *Install Rattle*

From within R you can install Rattle directly from CRAN with:

```
R: > install.packages("rattle")
```

An alternative is to install the most recent release from Togaware:

```
R: > install.packages("rattle",
  repos="http://rattle.togaware.com")
```

If these don't work for some reason you can also download the latest version of the `rattle` package directly from <http://rattle.togaware.com>. Download either the `.tar.gz` file for GNU/Linux and Mac/OSX, or the `.zip` file for MS/Windows, and then install with, for example:

```
R: > install.packages("rattle_2.2.37.zip", repos=NULL)
```

Alternatively, for example on Mac/OSX, you can do the following:

```
Mac: R CMD INSTALL rattle_2.2.37.tar.gz
```

Use the name of the file as it was downloaded. Some people report that the filename as downloaded actually becomes:

```
rattle_2.2.37.tar.gz.tar
```

You can either correct the name or use this name in the command.

6. Start Rattle

From an X Windows terminal if you are using GNU/Linux, or from an XTerminal if you are using Mac/OSX, or from the Rgui.exe if using MS/Windows (we call all these, generically, the **R Console**), you can load the `rattle` package into R's library:

```
R: > library(rattle)
```

This loads the Rattle functionality (which is also available without running the Rattle GUI). To start the Rattle GUI simply run the command:

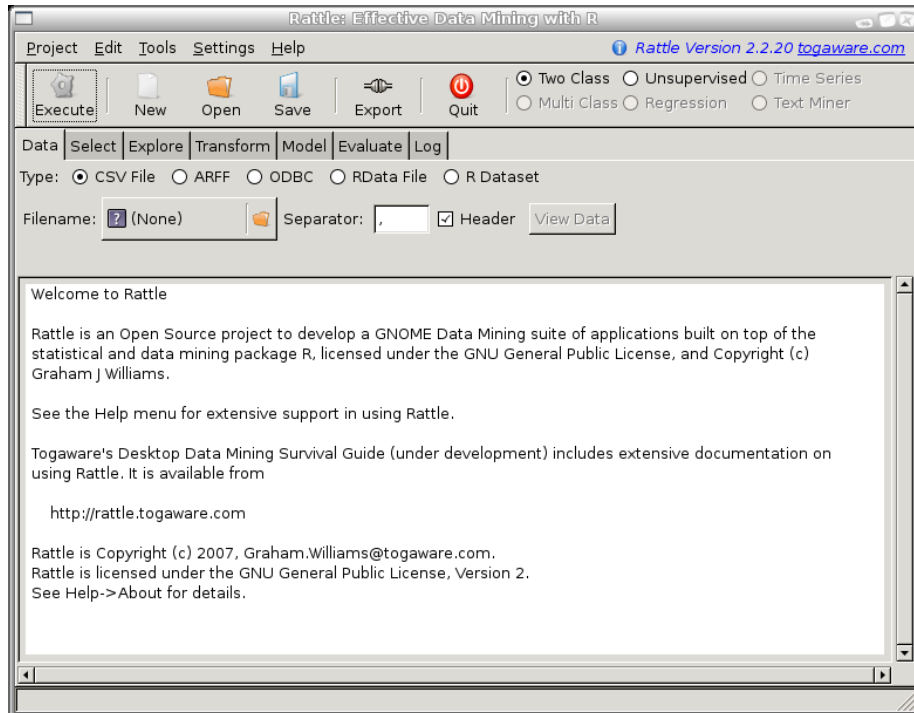
```
R: > rattle()
```

The main Rattle window will be displayed. You will see a welcoming message and a hint about using Rattle.

2.2 The Initial Interface

The user interface for Rattle is designed to flow through the data mining process, progressing through the Tabs from left to right. Here we illustrate the interface for the Two Class paradigm.

The process in Rattle is:



1. Load a **D**ataset;
2. **S**elect variables and entites for exploring and mining;
3. **E**xplore the data;
4. **T**ransform the data into training and test datasets;
5. Build your **M**odels;
6. **E**valuate the models;
7. Review the **L**og of the data mining process.

The collection of Paradigms, displayed as radio buttons to the right of the buttons on the toolbar, allow a multitude of Rattle functionality to be shared while supporting a variety of different types of tasks. For example, selecting the Unsupervised paradigm will expose the Cluster and Associate tabs whilst hiding the Model and Evaluation tabs.

2.3 Interacting with Rattle

The **Rattle** interface is based on a set of tabs through which we progress. For any tab, once we have set up the required information, we need to click the Execute button to perform the actions. Take a moment to explore the interface a little. Notice the Help menu and find that the help layout mimics the tab layout.

We will work through the functionality of **Rattle** with the use of a simple dataset, the *audit* dataset, which is supplied as part of the **Rattle** package (it is also available for download as a CSV file from <http://rattle.togaware.com/audit.csv>). This is an artificial dataset consisting of 2,000 fictional clients who have been audited, perhaps for compliance with regard the amount of a tax refund that is being claimed. For each case an outcome is recorded (whether the taxpayer's claims had to be adjusted or not) and any amount of adjustment that resulted is also recorded.

The dataset is only 2,000 entities in order to ensure model building is relatively quick, for illustrative purposes. Typically, our data contains tens of thousands and more (often millions) of entities. The *audit* dataset contains 13 columns (or variables), with the first being a unique client identifier. Again, real data will often have one hundred or more variables.

We proceed through the typical steps of a data mining project, beginning with a data load and selection, then an exploration of the data, and finally, modelling and evaluation.

The data mining process steps through each tab, left to right, performing the corresponding actions. For any tab, the *modus operandi* is to configure the options available and to then click the Execute button (or F5) to perform the appropriate tasks. It is important to note that the tasks are **not** performed until the Execute button (or F5 or the Execute menu item under Tools) is clicked.

The Status Bar at the base of the window will indicate when the action is completed. Messages from R (e.g., error messages, although many R error messages are captured by **Rattle** and displayed in a popup) will appear in the R console from where **Rattle** was started.

The R Code that is executed underneath will appear in the Log tab. This allows for a review of the R commands that perform the corresponding data mining tasks. The R code snippets can be copied as text from the Log tab and pasted into the R Console from which Rattle is running, to be directly executed. This allows a user to deploy Rattle for basic tasks, yet allow the full power of R to be deployed as needed, perhaps through using more command options than exposed through the Rattle interface. This also allows the user the opportunity to save the whole session to file as a record of the actions taken, and possibly for running directly and automatically through R itself at a later time.

2.4 Menus and Buttons

Before we proceed into the major functionality of Rattle, which is covered in the following chapters, we will review the functions provided by the menus and toolbar buttons. The Open and Save toolbar functions and the corresponding menu items under the Project menu are discussed in Section 2.4.1. Projects allow the current state of your interaction with Rattle to be saved to file for continuing with later on.

2.4.1 Project Menu and Buttons

A project is a packaging of a dataset, variable selections, explorations, clusters and models built from the data. Rattle allows projects to be saved for later resumption of the work or for sharing the data mining project with other users.

A project is typically saved to a file with the `.rattle` extension (although in reality it is just a standard `.RData` file).

At a later time you can load a project into rattle to restore the data, models, and other displayed information relating to the project, and resume your data mining from that point. You can also share these project files with other Rattle users, which is quite useful for data mining teams.

You can rename the files, keeping the `.rattle` extension, without impact-

ing the project file itself—that is, the file name has no formal bearing on the contents, so use it to be descriptive—but best to avoid vacant spaces and unusual characters!

2.4.2 Edit Menu

The Edit menu is currently not implemented.

2.4.3 Tools Menu and Toolbar

Execute

It is important to understand the user interface paradigm used within Rattle. Basically, we will specify within each Tab window what it is we want to happen, and then click the Execute button to have the actions performed. Pressing the F5 function key and selecting the menu item Execute under the Tools menu have the same effect.

Export

The Export button is available to export various objects and entities from Rattle. Details are available together with the specific sections in the following. The nature of the export depends on which tab is active, and within the tab, which option is active. For example, if the Model tab is on display then Export will save the current model as PMML.

Export is not yet implemented for all tabs and options.

2.4.4 Settings

The settings menu is under development. Currently the only choice is to turn tool tips on and off.

2.4.5 Help

Extensive help is available through the Help menu. The structure of the menu mostly follows that of the Tabs of the main interface.

2.5 Paradigms

There are many different uses to which data mining can be put. For identifying fraud or assessing the likelihood of a client to take up a particular product, we might think of the task as deciding between two outcomes. We might think of this as a two class problem. Or the task may be to decide what type of item from amongst a collection of items a client may have a propensity to purchase. We might thus think of this as a multi class problem. Perhaps we wish to predict how much someone might overstate an insurance claim or understate their income for taxation purposes or overstate their income when seeking approval for a credit card. Here we are predicting a continuous outcome, and refer to this as regression. In all of these situations, or *paradigms*, we might think of it as having a teacher who has supplied us examples of the outcomes—whether examples of fraud and non-fraudulent cases, or examples of different types of clients, or examples of clients and their declared income and actual income. In such cases we refer to the task as **supervised modelling**.

Perhaps though we know little about the individual, specific targets, but instead have general information about our population or their purchasing patterns. We might think of what we need to do in this case as building a model without the help of a teacher—or **unsupervised modelling**.

Alternatively, our data may be of a particular type, including time series or perhaps it is textual data. In these cases we have different tasks we wish to perform.

We refer to these different types of tasks in data mining as different paradigms, and Rattle provides different subsets of functionality for the different paradigms.

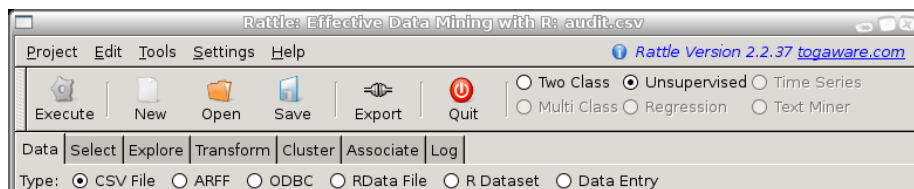


The paradigms are listed at the right end of the toolbar, and are selectable as radio buttons. The paradigms provided by Rattle are:

- **Two Class** classification predictive modelling;
- **Multi Class** classification predictive modelling;
- **Regression** continuous variable predictive modelling;
- **Unsupervised** learning or descriptive modelling;
- **Time Series** for temporal data mining; and
- **Text Mining** for mining of unstructured text data.

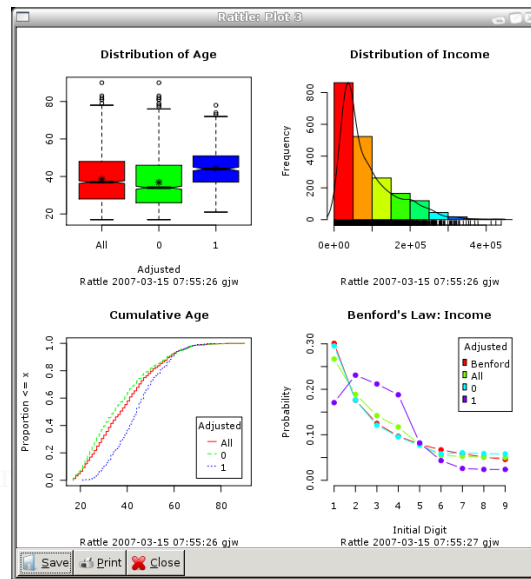
Selecting a paradigm will change the tabs that are available in the main body of Rattle. For example, the default paradigm is the Two Class paradigm, which displays a Model and Evaluate tab, as well as the other tabs. The Model tab exposes a collection of techniques for building two class models. The Evaluate tab provides a collection of tools for evaluating the performance of those models.

Selecting the Unsupervised paradigm removes the Model and the Evaluate tabs, replacing them with a Cluster and an Associate tab, for cluster analysis and association analysis, respectively.



2.6 Interacting with Plots

Rattle uses what is called the Cairo device for displaying any plots. If the Cairo device is not available within your installation then Rattle resorts to the default window device (*x11* for Linux and *window* for MS/Windows). The Cairo device has a number of advantages, one being that the device can be encapsulated within other windows, as is done with Rattle to provide various operating system independent functionality.



The Save button of the plot window (on the Cairo device) allows you to save the graphics to a file in one of the supported formats: **pdf**, **png** (good for vector images and text), and **jpg** (good for colourful images). A popup will request the filename to save to. The default is to save as PDF format, saving to a file with the filename extension of **.pdf**. You can choose to save in either **.png** or **.jpg** format simply by specifying these filename extensions.

Togaware Watermark For TeX Catalogue Survival

Chapter 3

Data

Data is the starting point for all data mining—without it there is nothing to mine. Today there is certainly no shortage of data—but turning that data into information and knowledge is no simple matter. In this chapter we explore issues relating to data, in particular, loading and selecting the data for modelling.

3.1 Nomenclature

Data miners have a plethora of terminology for many of the same things, due primarily from the history of data mining with its roots in many disciplines. Throughout this book we will use a single, consistent nomenclature, that is generally accepted.

We refer to collections of data as **datasets**. This might be a *matrix* or a database *table*. A dataset consists of rows which we might refer to as **entities**, and those entities are described in terms of **variables** which form the columns. Synonyms for *entity* include *record* and *object*, while synonyms for *variable* include *attribute* and *feature*.

Variables can serve one of two roles: as *input variables* or *output variables* (Hastie et al., 2001). **Input variables** are measured or preset data items while **output variables** are those that are “influenced” by the input variables. Often in data mining we build models to predict the output

variables in terms of the input variables. Input variables are also known as *predictors*, *independent variables*, *observed variables* and *descriptive variables*. Output variables are also known as *response* and *dependent variables*.

A **categorical variable** takes on a value from a fixed set of values (e.g., *low*, *medium*, and *high*) while a **numeric variable** has values that are integers or real numbers. Synonyms for *categorical variable* include *nominal variable*, *qualitative variable* and *factor*, while synonyms for *numeric variable*, include *quantitative variable* and *continuous*.

Thus, we will talk of **datasets** consisting of **entities** described using **variables**, which might consist of a mixture of **input variables** and **output variables**, either of which may be **categorical** or **numeric**.

3.2 Loading Data

The Data tab is the starting point for Rattle, and is where we can load a specific dataset into Rattle.

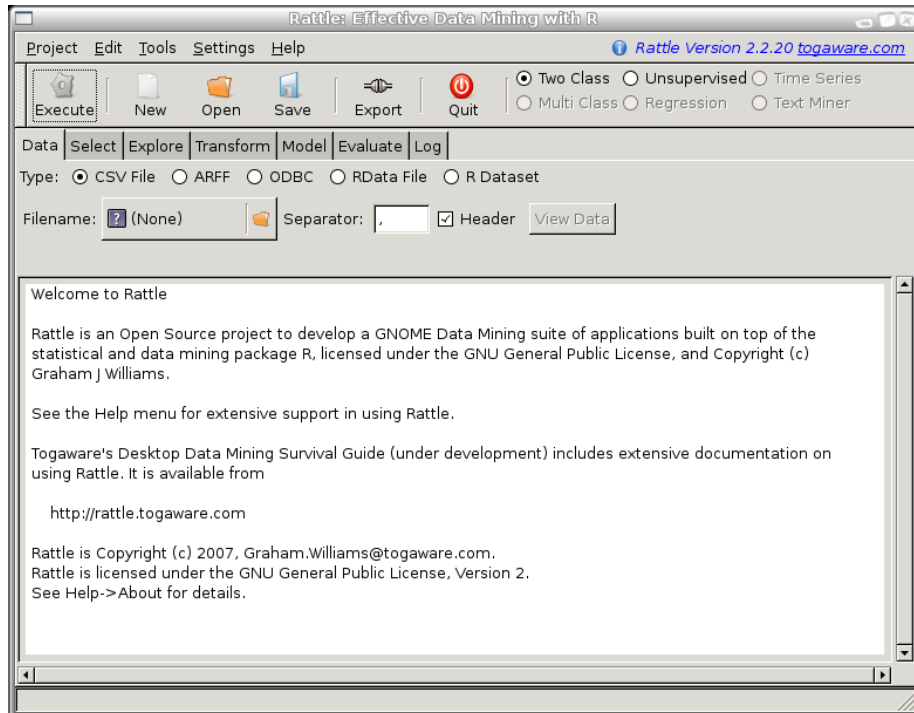
Rattle is able to load data from various sources. Support is directly included in Rattle for comma separated data files (`.csv` files as might be exported by a spreadsheet), tab separated files (`.txt`, which are also commonly exported from spreadsheets), the common data mining dataset format used by Weka (`.arff` files), and from an ODBC connection (thus allowing connection to an enormous collection of data sources including MS/Excel, MS/Access, SQL Server, Oracle, IBM DB2, Teradata, MySQL, and Postgress).

Underneath, R is very flexible in where it obtains its data from, and data from almost any source can be loaded. Consequently, Rattle is able to access this same variety of sources. It does, however, require the loading of the data into the R console and then within Rattle loading it as an R Dataset. All kinds of data can be loaded directly into R—including loading data directly from CSV and TXT files, MS/Excel spreadsheets, MS/Access databases, SAS, SPSS, Minitab, Oracle, MySQL, and SQLite.

Once a dataset has been identified the name of the dataset will be dis-

3.2 Loading Data

23



played in the title of the Rattle window.

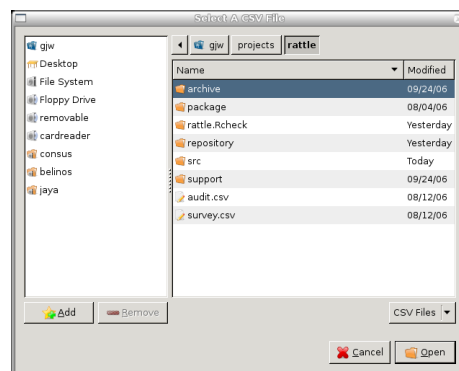


3.2.1 CSV Option

The CSV option of the Data tab is an easy way to load data into Rattle. CSV stands for “comma separated value” and is a standard file format often used to exchange data between various applications. CSV files can be exported from various spreadsheets and databases, including MS/Excel, Gnumeric, SAS/Enterprise Miner, Teradata’s Warehouse, and many, many, other applications. This is a pretty good option for importing your data into Rattle, although it does lose meta data information (that is, information about the data types of the dataset). Without this meta data R sometimes guesses at the wrong data type for a particular column, but it isn’t usually fatal!

A CSV file is actually a normal text file that you could load into any text editor to review its contents. A CSV file usually begins with a header row, listing the names of the variables, each separated by a comma. If any name (or indeed, any value in the file) contains an embedded comma, then that name (or value) will be surrounded by quote marks. The remainder of the file after the header is expected to consist of rows of data that record information about the entities, with fields generally separated by commas recording the values of the variables for this entity.

To make a CSV file known to Rattle we click the Filename button. A file chooser dialog will pop up. We can use this to browse our file system to find the file we wish to load into Rattle. By default, only files that have a `.csv` extension will be listed (together with folders). The pop up includes a pull down menu near the bottom right, above the Open button, to allow you to select which files are listed.



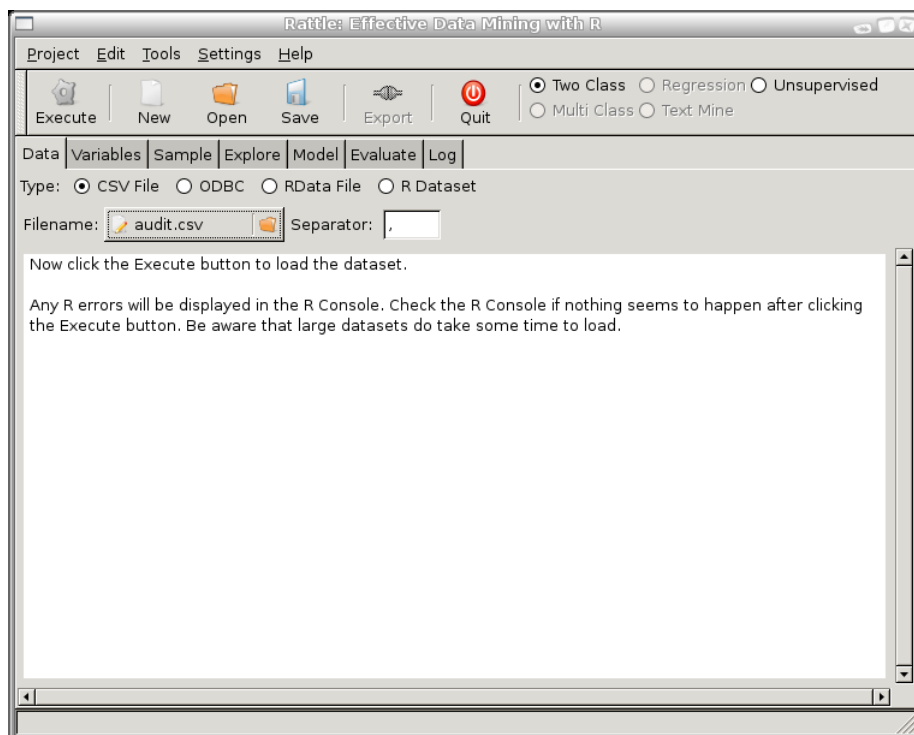
You can list only files that end with a `.csv` or a `.txt` or else to list all files. The `.txt` files are similar to CSV files but tend to use tab to separate columns in the data, rather than commas. The window on the left of the popup allows us to browse to the different file systems available to us, while the series of boxes at the top let us navigate through a series of

3.2 Loading Data

25

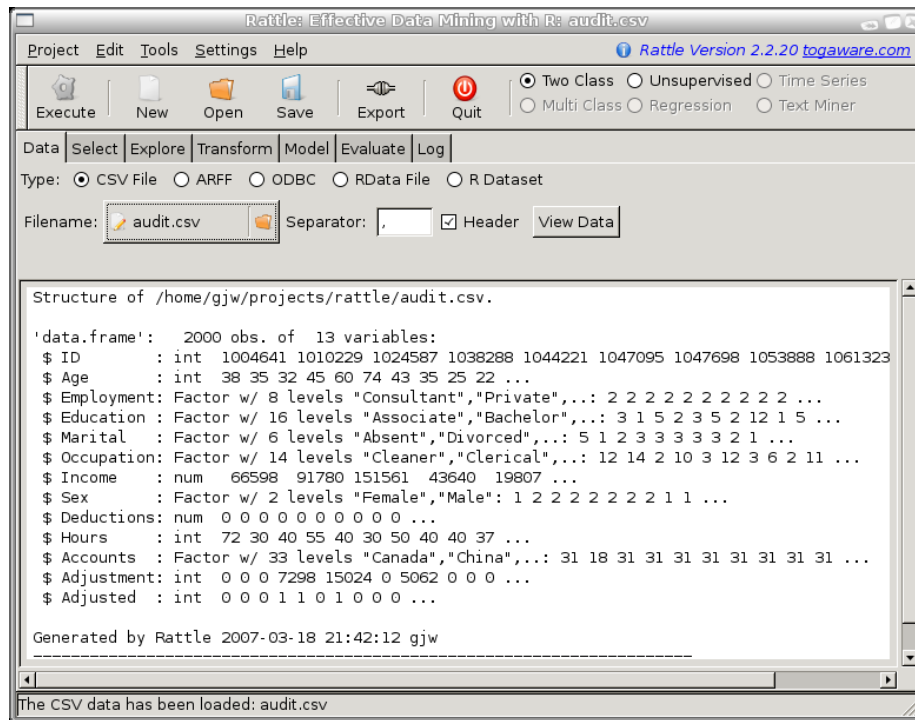
folders on a single file system. Once we have navigated to the folder on the file system on which we have saved the `audit.csv` file, we can select this file in the main panel of the file chooser dialog. Then click the Open button to tell Rattle that this is the file we are interested in.

Notice that the textview of the Data tab has changed to give a reminder as to what we need to do next.



We note that we have not yet told Rattle to actually load the data—we have just identified where the data is. So we now click the Execute button (or press the F5 key) to load the dataset from the `audit.csv` file. Since Rattle is a simple graphical interface sitting on top of R itself, the message in the textview also reminds us that some errors encountered by R on loading the data (and in fact during any operation performed by Rattle) may be displayed in the R Console.

The contents of the textview of the Data tab has now changed again.



The panel contains a brief summary of the dataset. We have loaded 2,000 entities (called observations in R), each described by 13 variables. The data type, and the first few values, for each entity are also displayed. We can start getting an idea of the shape of the data, noting that Adjusted, for example looks like it might be a categorical variable, with values 0 and 1, but R identifies it as an integer! That's fine.

You can choose the field delimiter through the Separator entry. A comma is the default. To load a .txt file which uses a tab as the field separator enter `\\t` as the separator. You can also leave the separator empty and any white space will be used as the separator.

Any data with missing values, or having the value "NA" or else ".", is treated as a missing value, which is represented in R as the string NA. Support for the "." convention allows the importation of CSV data generated by SAS.

3.2.2 ARFF Option

The Attribute-Relation File Format (ARFF) is an ASCII text file format that is essentially a CSV file with a header that describes the meta-data. ARFF was developed for use in the Weka machine learning software and there are quite a few datasets in this format now.

An example of the format for our *audit* dataset is:

```
@relation audit
@attribute ID numeric
@attribute Age numeric
@attribute Employment {Consultant, PSFederal, PSLocal, ...}
@attribute Education {Associate, Bachelor, College, Doctorate, ...}
@attribute Marital {Absent, Civil, Divorced, Married, ...}
@attribute Occupation {Cleaner, Clerical, Executive, Farming, ...}
@attribute Income numeric
@attribute Sex {Female, Male}
@attribute Deductions numeric
@attribute Hours numeric
@attribute Accounts {Canada, China, Columbia, Cuba, Ecuador, ...}
@attribute Adjustment numeric
@attribute Adjusted {0, 1}
@data
1004641,38,Private,College,Separated,Service,71511.95,Female,0,...
1010229,35,Private,Associate,Unmarried,Transport,73603.28,Male,...
1024587,32,Private,HSgrad,Divorced,Clerical,82365.86,Male,0,40,...
1038288,45,Private,?,Civil,Repair,27332.32,Male,0,55,...
1044221,60,Private,College,Civil,Executive,21048.33,Male,0,40,...
...
```

A dataset is firstly described, beginning with the name of the dataset (or the *relation* in ARFF terminology). Each of the variables (or *attribute* in ARFF terminology) used to describe the entities is then identified, together with their data type, each definition on a single line (we have truncated the lines in the above example). Numeric variables are identified as *numeric*, *real*, or *integer*. For categorical variables we simply see a list the of possible values.

Two other data type recognised by ARFF are *string* and *date*. A *string* data type simple indicates that the variable can have any string as its value. A *date* data type also optionally specifies the format in which the date is presented, with the default being in ISO-8601 format which is equivalent to the specification of:

```
@attribute lodged date "yyyy-MM-dd'T'HH:mm:ss"
```

The actual entities are then listed, each on a single line, with fields separated by commas, much like a CSV file.

Comments can be included in the file, introduced at the beginning of a line with a %, whereby the remainder of the line is ignored.

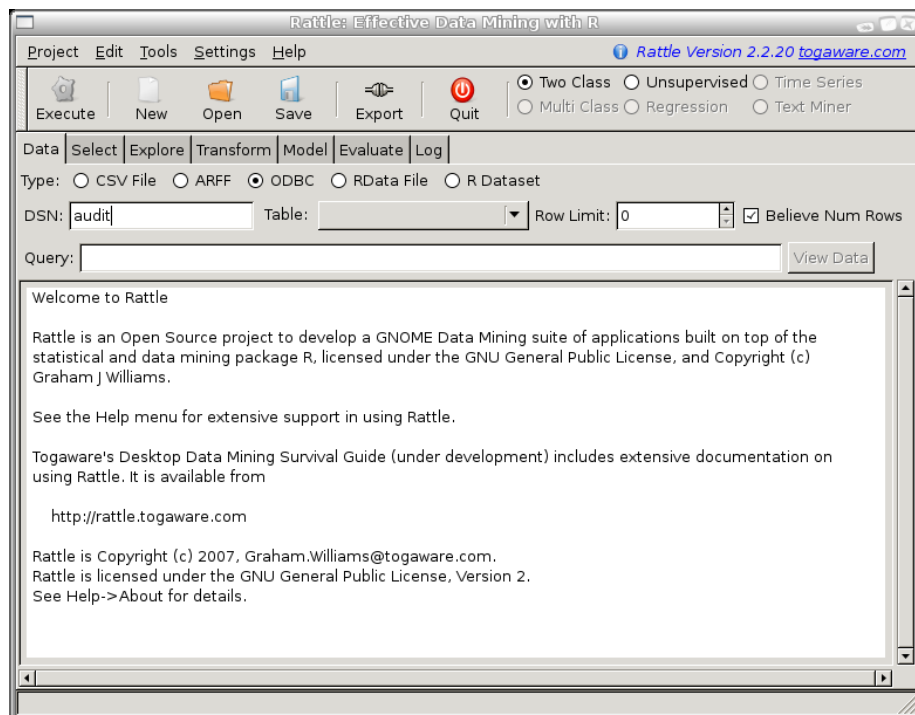
A significant advantage of the ARFF data file over the CSV data file is the meta data information. This is particularly useful in Rattle where for categorical data the possible values are determined from the data (which may not included every possible value) rather than from a full list of possible values.

Also, the ability to include comments ensure we can record extra information about the data set, including how it was derived, where it came from, and how it might be cited.

Missing values in an ARFF dataset are identified using the question mark ?. These are identified by *read.arff* underneath and we see them as the usual NA in Rattle.

3.2.3 ODBC Option

Rattle supports obtaining a dataset from any database accessible through ODBC (Open Database Connectivity).

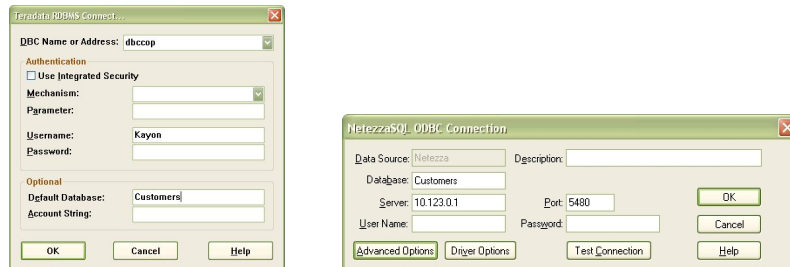


The key to using ODBC is to know (or to set up) the data source name (the DSN). Within Rattle we specify a known DSN by typing the name into the text entry. Once that is done, we press the Enter key and Rattle will attempt to connect. This may require a username and password to be supplied. For a Teradata Warehouse connection you will be presented with a dialog box like that in the figure below.

For a Netezza ODBC connection we will get a window like the one in the following (on the left):

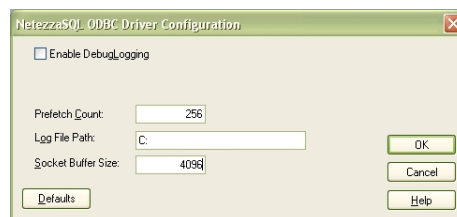
If the connection is successful we will find a list of available tables in the Table combobox.

We can choose a Table, and also include a limit on the number of rows



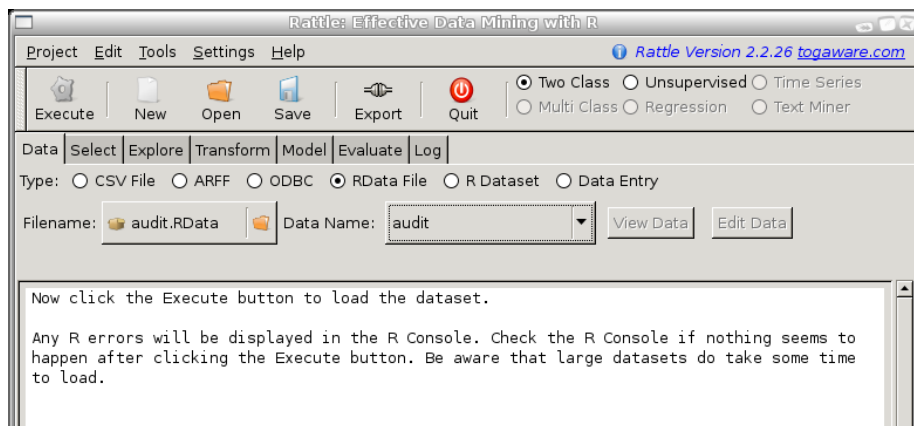
that we wish to load into Rattle. This allows us to get a smaller sample of the data for testing purposes before loading up a large dataset. If the Row Limit is set to 0 then all of the rows from the table are retrieved. Unfortunately there is now SQL standard for limiting the number of rows returned from a query. For the Teradata and Netezza warehouses the SQL keyword is `LIMIT` and this is what is used by Rattle.

The Believe Num Rows option is an oddity required for some ODBC drivers and appears to be associated with the pre-fetch behaviour of these drivers. The default is to activate the check button (i.e., Believe Num Rows is True). However, if you find that you are not retrieving all rows from the source table, the the ODBC driver may be using a pre-fetch mechanism that does not “correctly” report the number of rows (it is probably only reporting the number of rows limited to the size of the pre-fetch). In these cases deactivate the Believe Num Rows check button. See Section 16.9 for more details. Another solution is to either disable the pre-fetch option of the driver, or to increase its count. For example, in connecting through the Netezza ODBC driver the configuration window is available, where you can change the default Prefetch Count value.



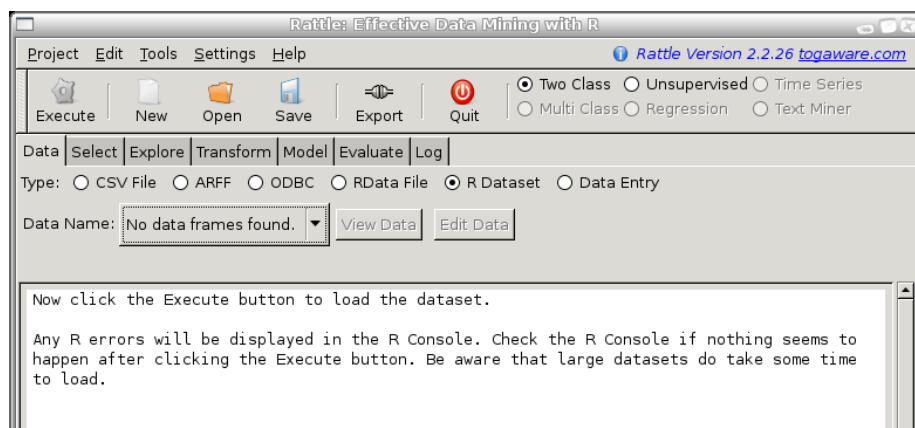
3.2.4 RData File Option

Using the *RData File* option data can be loaded directly from a native R data file (usually with the *.RData* or *.RData* extension). Such files may contain multiple datasets (compressed) and you will be given an option to choose just one of the available datasets from the combo box.



3.2.5 R Dataset Option

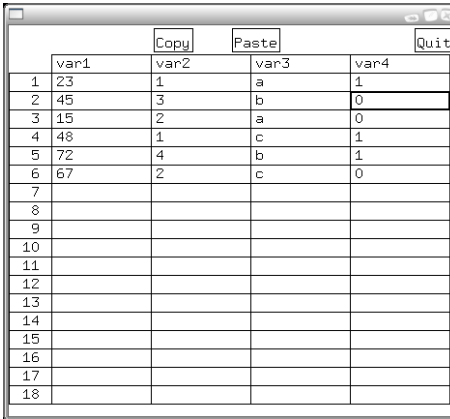
Rattle can use a dataset that is already loaded into R (although it will take a copy of it, with memory implications). Only data frames are currently supported, and Rattle will list for you the names of all of the available data frames.



The data frames need to be constructed in the same R session that is running Rattle (i.e., the same R Console in which you load the Rattle package). This provides much more flexibility in loading data into Rattle, than is provided directly through the actual Rattle interface. For example, you may want to load data from an SQLite database directly, and have this available in Rattle.

3.2.6 Data Entry

The Data Entry option provides a simple mechanism to manually enter data for use in Rattle. Of course, you would not want to do this for anything but a small dataset, but at least the option is there for some very simple exploration of Rattle without the overhead of loading some other dataset into Rattle.



	var1	var2	var3	var4
1	23	1	a	1
2	45	3	b	0
3	15	2	a	0
4	48	1	c	1
5	72	4	b	1
6	67	2	c	0
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				

Togaware Watermark For TeX Catalogue Survival

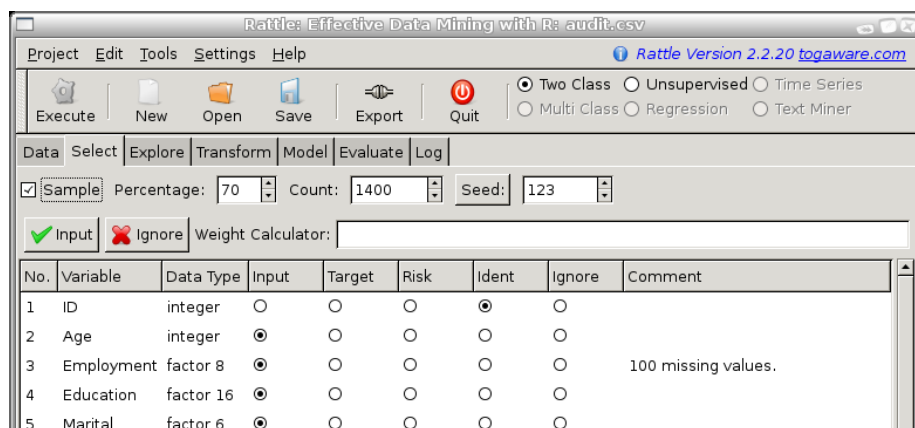
3.3 Select Data

The Select tab is used to select a subset of rows (entities) to include in the data mining, and to identify the role played by each of the variables in the dataset.

Remember, for any changes that we make to the Variables tab to actually take effect we need to click the Execute button (or press F5 or choose the Execute menu item from the Tools menu.)

3.3.1 Sampling Data

The Sample option allows us to partition our dataset into a training dataset and a testing dataset, and to select different random samples if we wish to explore the sensitivity of our models to different data samples.



Here we specify how we might partition the dataset for exploratory and modelling purposes. The default for Rattle is to build two subsets of the dataset: one is a training dataset from which to build models, while the other is used for testing the performance of the model. The default for Rattle is to use a 70% training and a 30% testing split, but you are welcome to turn sampling off, or choose other samplings. A very small sampling may be required to perform some explorations of the smaller dataset, or to build models using the more computationally expensive algorithms (like support vector machines).

R uses random numbers to generate samples, which may present a problem with regard repeatable modelling. This presents itself through the fact that each time the *sample* function is called we will get a different random sample. However, R provides the *set.seed* function to set a seed for the next random numbers it generates. Thus, by setting the seed to the same number each time you can be assured of obtaining the same sample.

Rattle allows you to specify the random number generator seed. By default, this is the number 123, but you can change this to get a different random sample on the next Execute. By keep the random number generator seed constant you can guarantee to get the same model, and by changing it you can explore the sensitivity of the modelling to different samples of the dataset.

Often in modelling we build our model on a training dataset and then test its performance on a test dataset.

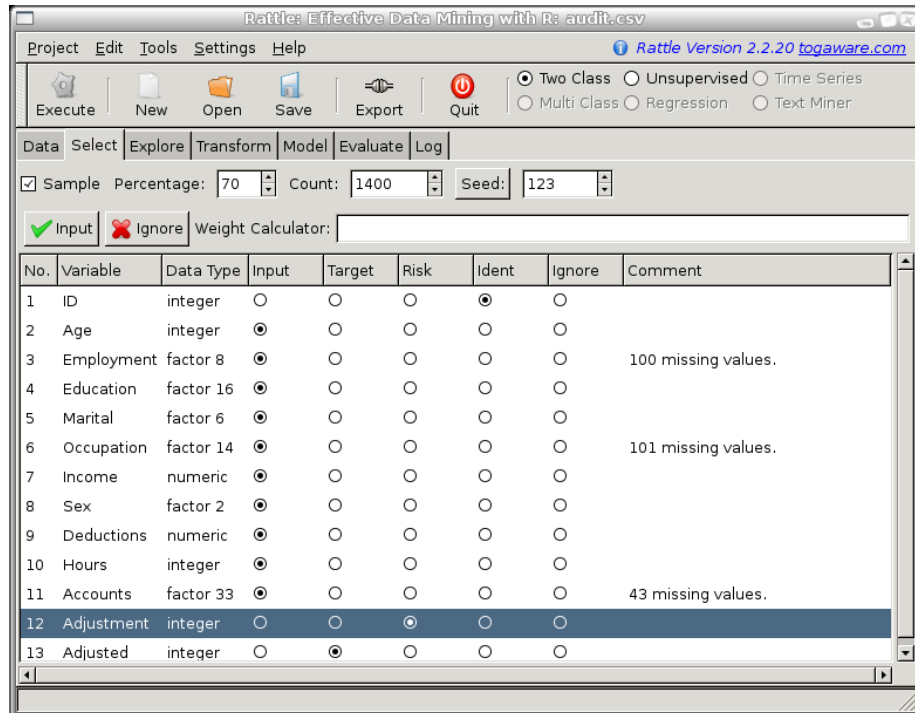
Togaware Watermark For TeX Catalogue Survival

3.3.2 Variable Roles

Variables can be inputs to modelling, the target variable for modelling, the risk variable, an identifier, or an ignored variable. The default role for most variables is that of an Input variable. Generally, these are the variables that will be used to predict the value of a Target variable.

Rattle uses simple heuristics to guess at a Target role for one of the variables. Here we see that Adjusted has been selected as the target variable. In this instance it is correct. The heuristic involves examining the number of distinct values that a variable has, and if it has less than 5, then it is considered as a candidate. The candidate list is ordered starting with the last variable (often the last variable is the target), and then proceeding from the first onwards to find the first variable that meets the conditions of looking like a target.

Any numeric variables that have a unique value for each record is automatically identified as an Ident. Any number of variables can be tagged as being an Ident. All Ident variables are ignored when modelling, but are used after scoring a dataset, being written to the resulting score file so that the cases that are scored can be identified.



Sometimes not all variables in your dataset should be used or may not be appropriate for a particular modelling task. For example, the random forest model builder does not handle categorical variables with more than 32 levels, so you may choose to Ignore Accounts. You can change the role of any variable to suit your needs, although you can only have one Target and one Risk.

For an example of the use of the Risk variable, see Section 7.1.

3.3.3 Automatic Role Identification

Special variable names can be used with data imported into Rattle (and in fact for any data used by Rattle) to identify their role. Any variable with a name beginning with `IGNORE_` will have the default role of Ignore. Similarly `RISK_` and `TARGET_`. Any variable beginning with `IMP_` is assumed to be an imputed variable, and if there exists a variable with the same name, but without the `IMP_` prefix, that variable will be marked as

Ignore.

3.3.4 Weights Calculator

Togaware Watermark For TeX Catalogue Survival

Togaware Watermark For TeX Catalogue Survival

Chapter 4

Explore

A key task in any data mining project is **exploratory data analysis** (often abbreviated as EDA), which generally involves getting a basic understanding of a dataset. Statistics, the fundamental tool here, is essentially about uncertainty—to understand it and thereby to make allowance for it. It also provides a framework for understanding the discoveries made in data mining. Discoveries need to be statistically sound and statistically significant—uncertainty associated with modelling needs to be understood.

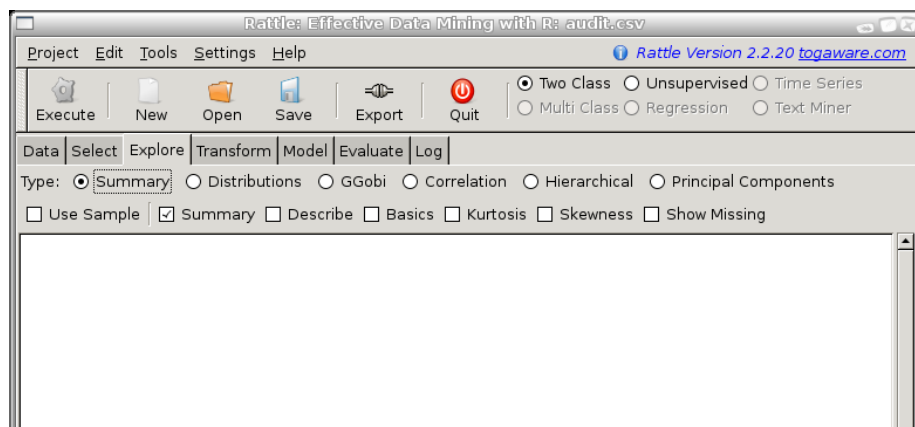
We explore the shape or distribution of our data before we begin mining. Through this exploration we begin to understand the “lay of the land,” just as a miner works to understand the terrain before blindly digging for gold. Through this exploration we may identify problems with the data, including missing values, noise and erroneous data, and skewed distributions. This will then drive our choice of tools for preparing and transforming our data and for mining it.

Rattle provides tools ranging from textual summaries to visually appealing graphical summaries, tools for identifying correlations between variables, and a link to the very sophisticated **GGobi** tool for visualising data. The Explore tab provides an opportunity to understand our data in various ways.

4.1 Summary Option

While a picture might tell a thousand stories, textual summaries still play an important roll in our understanding of data. We saw a basic summary of our data after first loading the data into Rattle (page 26). The data types and the first few values for each of the variables are automatically listed. This is the most basic of summaries, and even so, begins to tell a story about the data. It is the beginnings of understanding the data.

Rattle's Summary option of the Explore tab provides a number of more detailed textual summaries of our data.

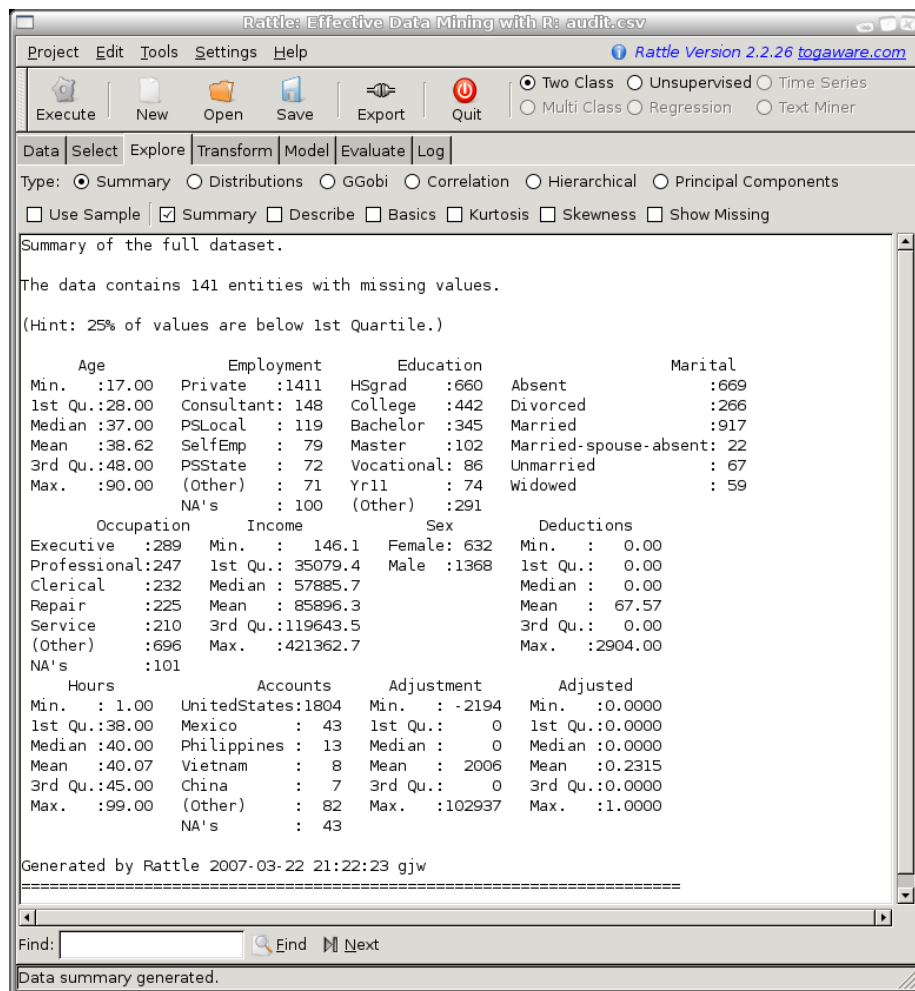


With the Use Sample check button we can choose to summarise the whole dataset, or just the training dataset. We might choose to only summarise the sample when the dataset itself is very large and the summaries take a long time to perform. We would usually not choose the sample option.

The rest of the check buttons of the Summary option allows us to fine tune what it is we wish to explore textually. We can choose to display one or many of the summary options. The first three—Summary, Describe, and Basic—are three alternatives that provide overall statistics for each variable (although the Basics option only summarises numeric variables). The final two, Kurtosis and Skewness provide specific measures of the characteristics of the data. These are separated out so that we can compare the kurtosis or skewness directly across a number of variables. These two measures both apply only to numeric data.

4.1.1 Summary

The Summary check button provides numerous measures for each variable, including, in the first instance, the minimum, maximum, median, mean, and the first and third quartiles. Generally, if the mean and median are significantly different then we would think that there are some entities with very large values in the data pulling the mean in one direction. It does not seem to be the case for Age but is for Income.



Rattle: Effective Data Mining with R: audit.csv
Rattle Version 2.2.26 togaware.com

Project Edit Tools Settings Help

Execute New Open Save Export Quit

Two Class Unsupervised Time Series
Multi Class Regression Text Miner

Data Select Explore Transform Model Evaluate Log

Type: ☒ Summary ☐ Distributions ☐ GGobi ☐ Correlation ☐ Hierarchical ☐ Principal Components

☐ Use Sample ☒ Summary ☐ Describe ☐ Basics ☐ Kurtosis ☐ Skewness ☐ Show Missing

Summary of the full dataset.

The data contains 141 entities with missing values.

(Hint: 25% of values are below 1st Quartile.)

Age		Employment		Education		Marital	
Min.	:17.00	Private	:1411	HSgrad	:660	Absent	:669
1st Qu.	:28.00	Consultant	:148	College	:442	Divorced	:266
Median	:37.00	PSLocal	:119	Bachelor	:345	Married	:917
Mean	:38.62	SelfEmp	:79	Master	:102	Married-spouse-absent	:22
3rd Qu.	:48.00	PSState	:72	Vocational	:86	Unmarried	:67
Max.	:90.00	(Other)	:71	Yr11	:74	Widowed	:59
		NA's	:100	(Other)	:291		

Occupation		Income		Sex		Deductions	
Executive	:289	Min.	:146.1	Female	:632	Min.	:0.00
Professional	:247	1st Qu.	:35079.4	Male	:1368	1st Qu.	:0.00
Clerical	:232	Median	:57885.7			Median	:0.00
Repair	:225	Mean	:85896.3			Mean	:67.57
Service	:210	3rd Qu.	:119643.5			3rd Qu.	:0.00
(Other)	:696	Max.	:421362.7			Max.	:2904.00
NA's	:101						

Hours		Accounts		Adjustment		Adjusted	
Min.	:1.00	UnitedStates	:1804	Min.	: -2194	Min.	:0.0000
1st Qu.	:38.00	Mexico	:43	1st Qu.	:0	1st Qu.	:0.0000
Median	:40.00	Philippines	:13	Median	:0	Median	:0.0000
Mean	:40.07	Vietnam	:8	Mean	:2006	Mean	:0.2315
3rd Qu.	:45.00	China	:7	3rd Qu.	:0	3rd Qu.	:0.0000
Max.	:99.00	(Other)	:82	Max.	:102937	Max.	:1.0000
		NA's	:43				

Generated by Rattle 2007-03-22 21:22:23 gjw

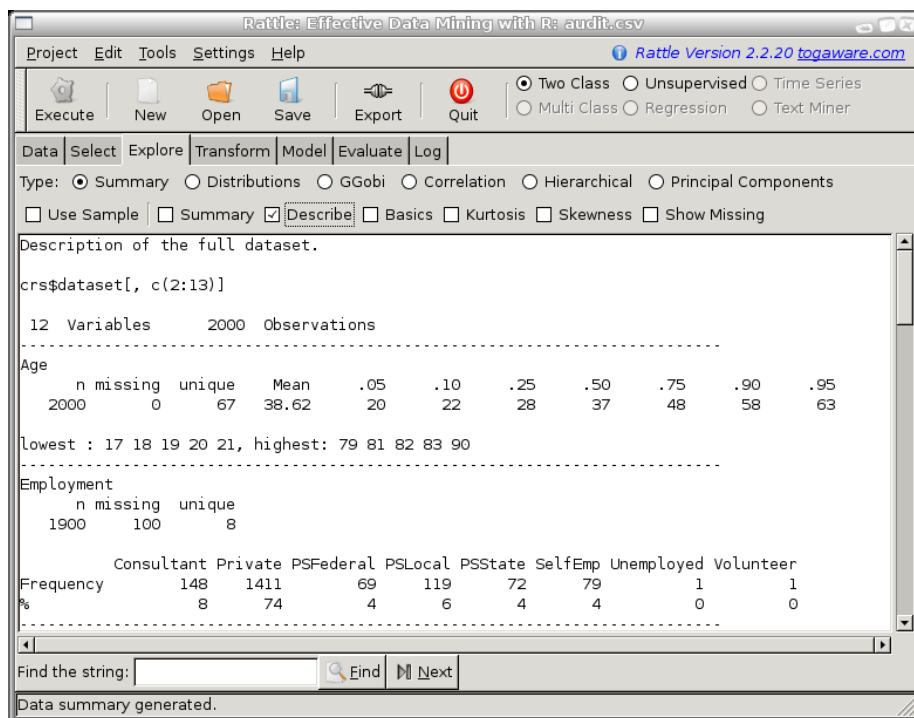
Find: Find Next

Data summary generated.

For categorical variables the

4.1.2 Describe

The Describe check button

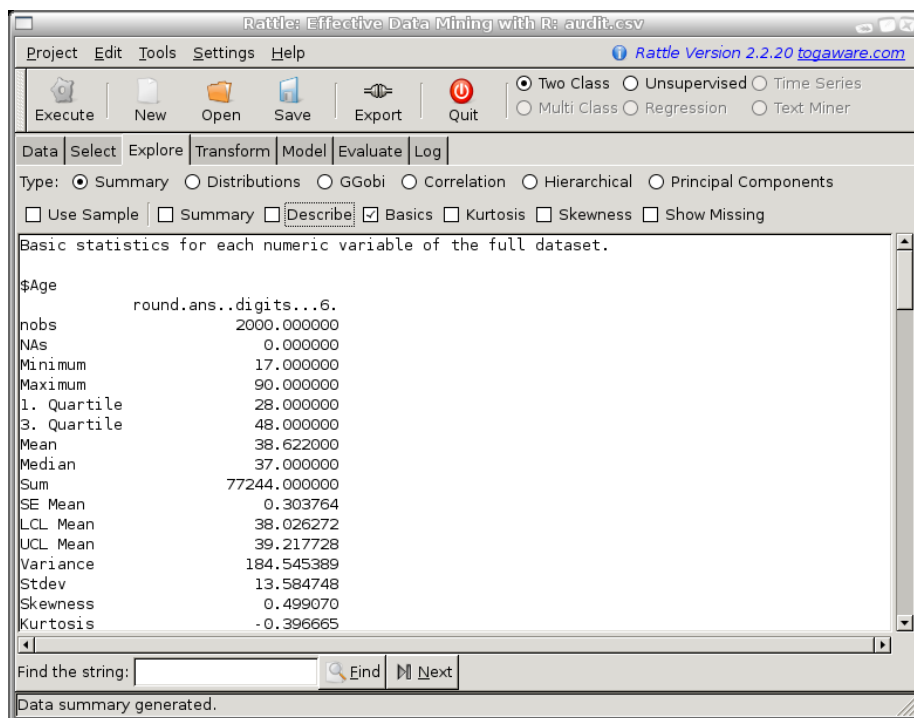


4.1 Summary Option

43

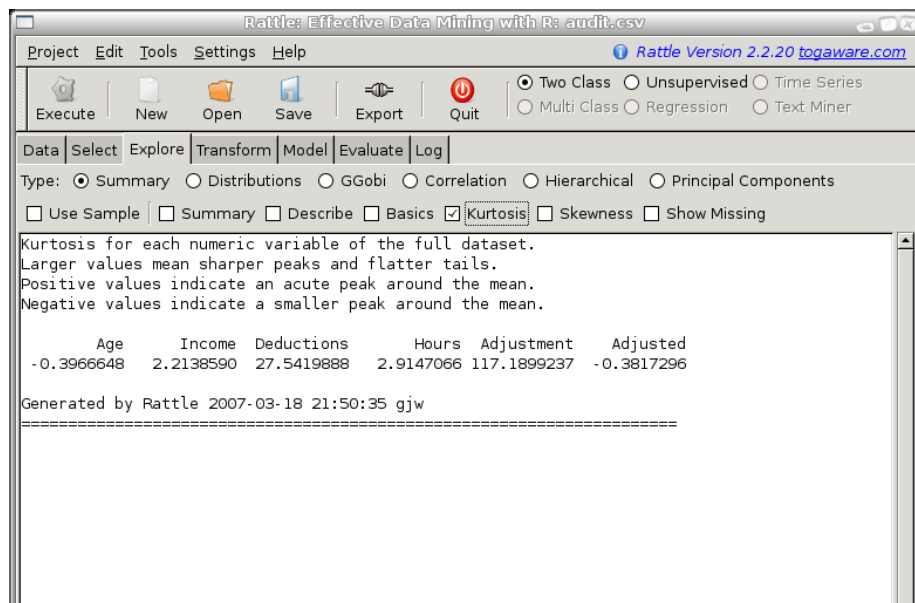
4.1.3 Basics

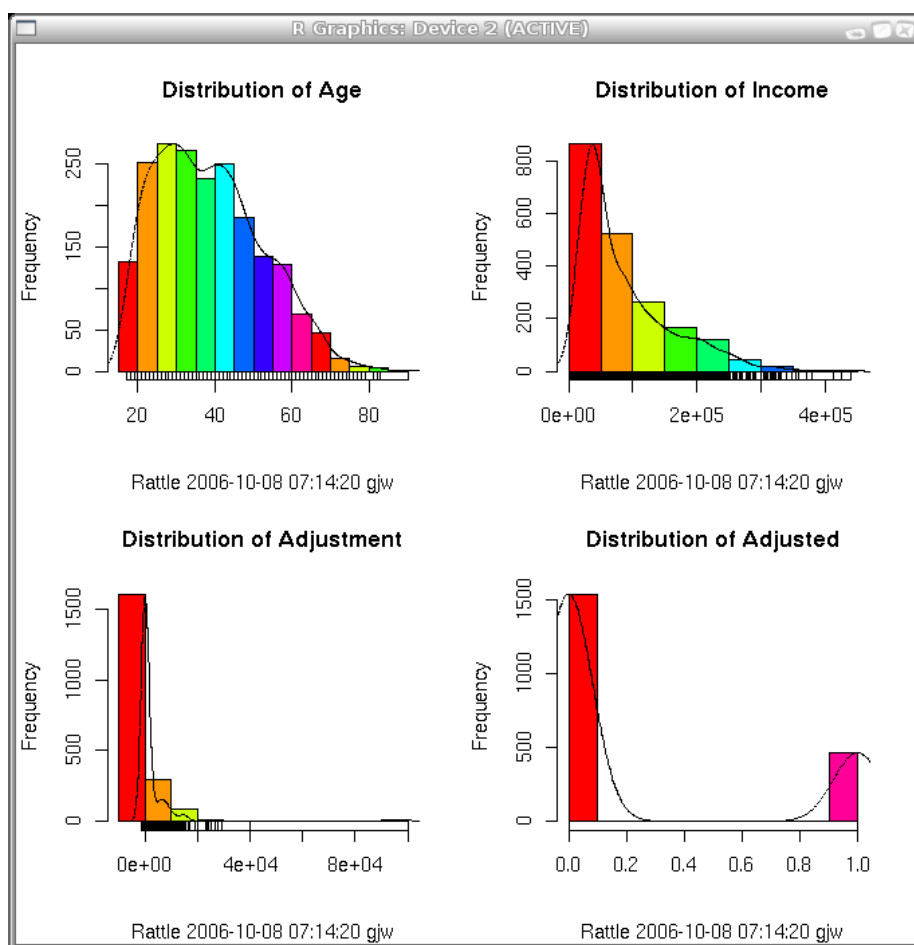
The Basics check button



4.1.4 Kurtosis

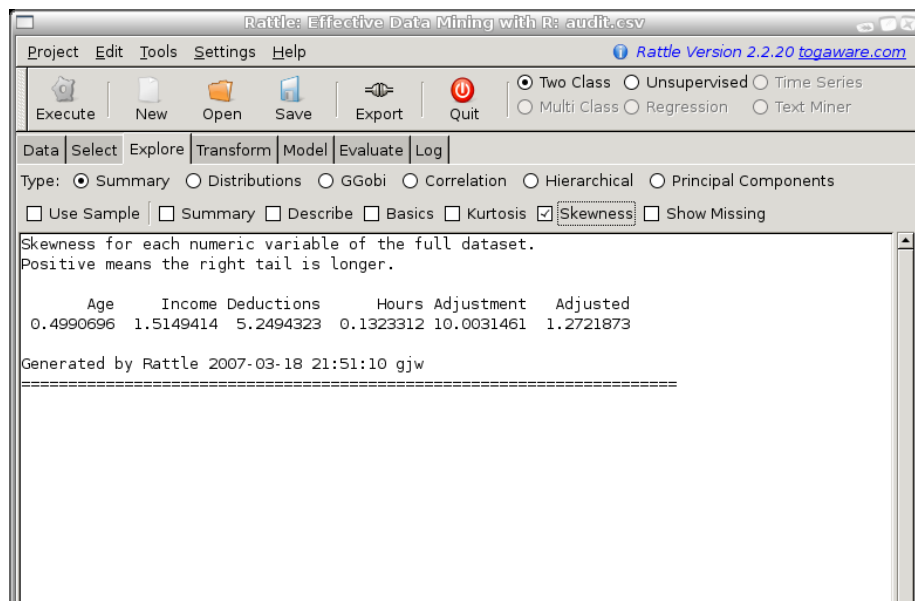
The **kurtosis** is a measure of the nature of the peaks in the distribution of the data. A larger value for the kurtosis will indicate that the distribution has a sharper peak, as we can see in comparing the distributions of Income and Adjustment. A lower kurtosis indicates a smoother peak.





4.1.5 Skewness

The **skewness** is a measure of how asymmetrical the data is distributed. Skewness indicates the assymetry of the distribution. A positive skew indicates that the tail to the right is longer, and a negative skew that the tail to the left is longer.



4.1 Summary Option

47

4.1.6 Missing

Rattle: Effective Data Mining with R: audit_missing.csv
Rattle Version 2.2.26 togaware.com

Project Edit Tools Settings Help

Execute New Open Save Export Quit

Two Class Unsupervised Time Series
Multi Class Regression Text Miner

Data Select Explore Transform Model Evaluate Log

Type: ☒ Summary ☐ Distributions ☐ GGobi ☐ Correlation ☐ Hierarchical ☐ Principal Components

☐ Use Sample ☐ Summary ☐ Describe ☐ Basics ☐ Kurtosis ☐ Skewness ☒ Show Missing

Missing Value Summary

	Marital	Deductions	Age	Income	Sex	Education	Occupation	Employment
1575	1	1	1	1	1	1	1	0
39	1	1	0	1	1	1	1	1
40	1	1	1	1	1	1	1	0
57	1	1	1	1	1	0	1	1
35	0	1	1	1	1	1	1	1
25	1	1	1	1	1	1	0	1
37	1	1	1	0	1	1	1	1
41	1	1	1	1	0	1	1	1
34	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	0	2
88	1	1	1	1	1	1	0	2
1	1	1	0	0	1	1	1	2
1	1	1	1	0	1	1	1	2
2	1	1	1	0	1	1	0	2
2	1	1	1	1	0	1	1	2
1	0	1	1	1	0	1	1	2
1	1	1	1	1	0	1	0	2
1	1	1	1	0	0	1	1	2
1	1	0	0	1	1	1	1	2
2	1	0	1	1	0	1	1	2
3	1	1	1	1	1	0	0	3
1	0	1	1	1	1	1	0	3
4	1	1	1	0	1	1	0	3
3	1	1	1	1	0	1	0	3
3	1	0	1	1	1	1	0	3
1	1	1	1	0	1	0	0	4
1	1	1	1	1	0	0	0	4
	37	40	42	47	52	62	133	147 560

Generated by Rattle 2007-03-22 21:25:55 gjw

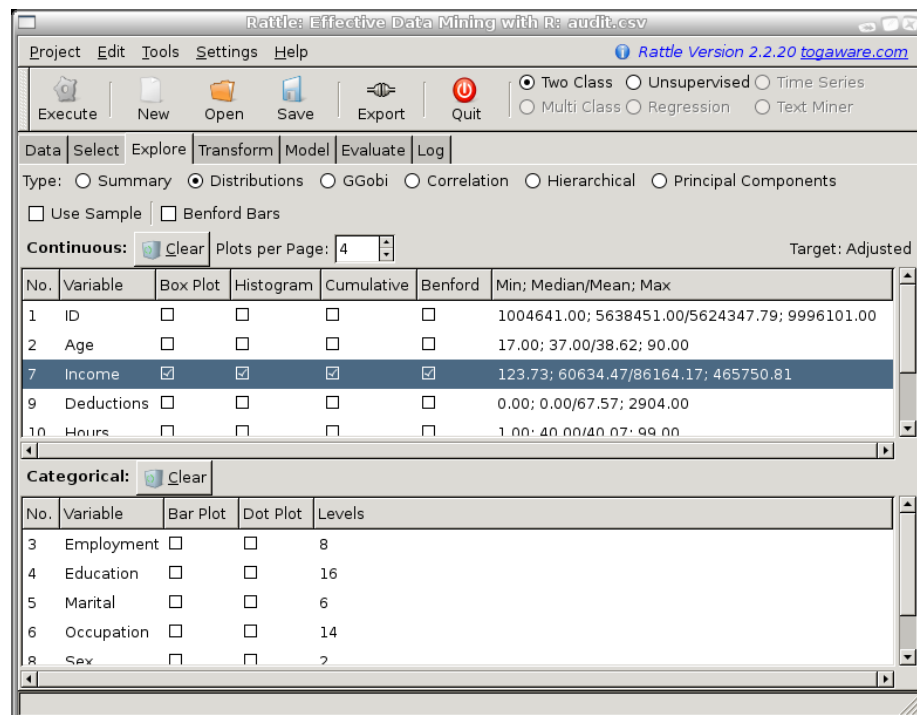
4.2 Distributions Option

It is usually a good idea to review the distributions of the values of each of the variables in your dataset. The Distributions option allows you to visually explore the distributions for specific variables.

Using graphical tools to visually investigate the data's characteristics can help our understanding the data, in error correction, and in variable selection and variable transformation.

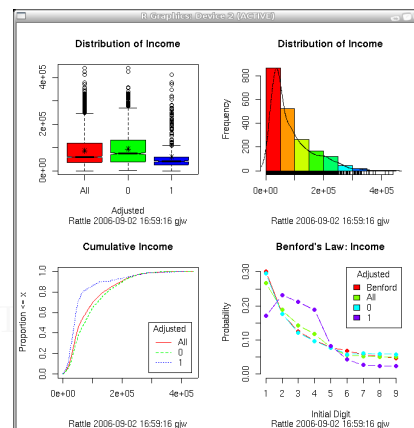
Graphical presentations are more effective for most people, and Rattle provides a graphical summary of the distribution of the data with the Distribution option of the Explore tab.

Visualising data has been an area of study within statistics for many years. A vast array of tools are available within R for presenting data visually and the topic is covered in detail in books in their own right, including [Cleveland \(1993\)](#) and Tufte.



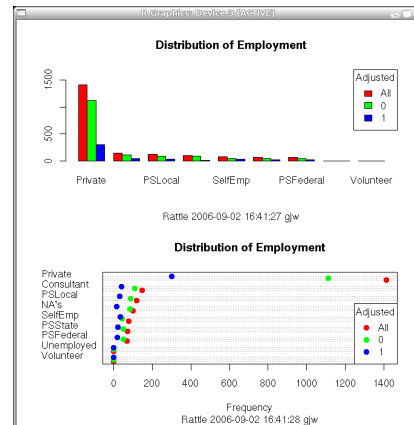
By choosing the Distributions radio button you can select specific variables of interest, and display various distribution plots. Selecting many variables will lead to many plots being displayed, and so it may be useful to display multiple plots per page (i.e., per window) by setting the appropriate value in the interface. By default, four plots will be displayed per page or window, but you can change this to anywhere from 1 plot per page to 9 plots per page.

Here we illustrate a window with the default four plots. Four plots per page are useful, for example, to display each of the four different types of plots for a single continuous variable. Clockwise, they are the Box Plot, the Histogram, a Cumulative Function Plot, and a Benford's Law Plot. Because we have identified a target variable the plots include the distributions for each subset of entities associated with each value of the target variable, wherever this makes sense to do so (e.g., not for the histogram).



The box and whiskers plot identifies the median and mean of the variable, the spread from the first quartile to the third, and indicates the outliers. The histogram splits the range of values of the variable into segments and shows the number of entities in each segment. The cumulative plot shows the percentage of entities below any particular value of the variable. And the Benford's Law plot compares the distribution of the first digit of the numbers against that which is expected according to Benford's Law. Each of the plots shown here is explained in more detail in the following sections.

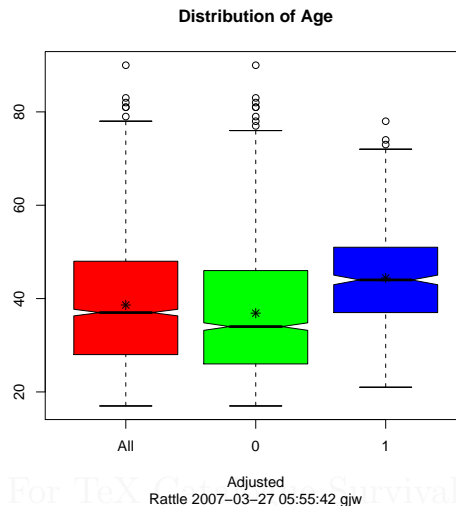
For categorical variables two types of plots are supported, more as alternatives than adding extra information: the Bar Plot and the Dot Plot. Each plot shows the number of entities that have a particular value for the chosen variable. Both are sorted from the most frequent to the least frequent value. For example, we can see that the value **Private** of the variable **Employment** is the most frequent, occurring over 1,400 times in this dataset.



A bar plot uses vertical bars while the dot plot uses dots placed horizontally. The dot plot has more of a chance to list the actual values for the variable, whilst a bar plot will have trouble listing all of the values (as illustrated here). Each of the plots shown here is explained in more detail in the following sections.

4.2.1 Box Plot

A **boxplot** (Tukey, 1977) (also known as a box-and-whisker plot) provides a graphical overview of how data is distributed over the number line. Rattle's Box Plot displays a graphical representation of the textual *summary* of data. It is useful for quickly ascertaining the skewness of the distribution of the data. If we have identified a Target variable, then the boxplot will also show the distribution of the values of the variable partitioned by values of the target variable, as we illustrate for the variable Age where Adjusted has been chosen as the Target variable.



The boxplot shows the **median** (which is also called the second **quartile** or the 50th **percentile**) as the thicker line within the box ($Age = 37$ over the whole population, as we can see from the Summary option's Summary check button). The top and bottom extents of the box (48 and 28 respectively) identify the upper quartile (the third quartile or the 75th percentile) and the lower quartile (the first quartile and the 25th percentile). The extent of the box is known as the **interquartile range** ($48 - 28 = 20$). The dashed lines extend to the maximum and minimum data points that are no more than 1.5 times the interquartile range from the median. Outliers (points further than 1.5 times the interquartile range from the median) are then individually plotted (at 79, 81, 82, 83, and 90). The **mean** (38.62) is also displayed as the asterisk.

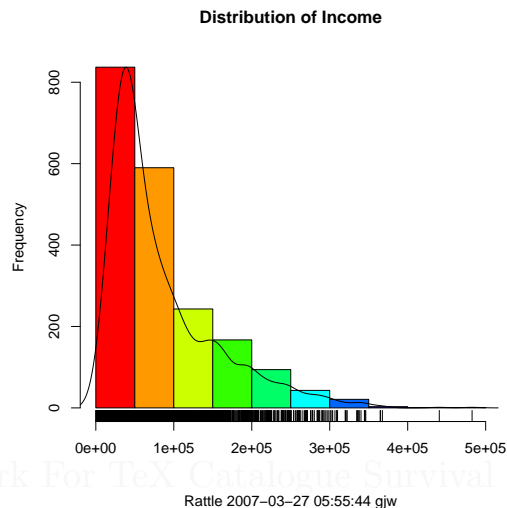
The notches in the box, around the median, indicate a level of confidence about the value of the median for the population in general. It is useful in comparing the distributions, and in this instance it allows us to say that all three distributions being presented here have significantly different means. In particular we can state that the positive cases (where

Adjusted = 1) are older than the negative cases (where *Adjusted* = 0).

Togaware Watermark For TeX Catalogue Survival

4.2.2 Histogram

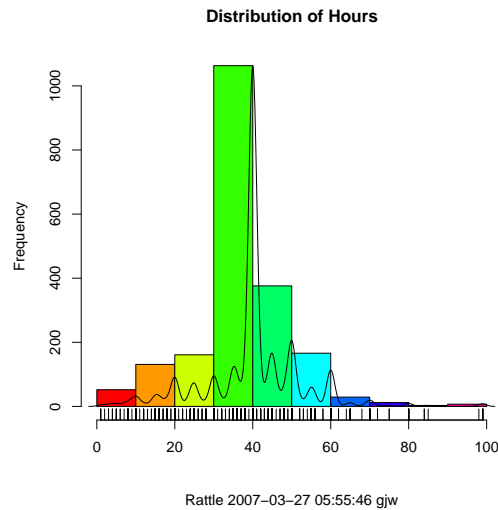
A **histogram** provides a quick and useful graphical view of the spread of the data. A histogram plot in Rattle includes three components. The first of these is obviously the coloured vertical bars. The continuous data in the example here (Distribution of Income) has been partitioned into ranges, and the frequency of each range is displayed as the bar. R is automatically choosing both the partitioning and how the x-axis is labelled here, showing x-axis points at 0, 10,000 (using scientific notation of $1e + 05$ which means 1×10^5 , or 10,000), and so on. Thus, we can see that the most frequent range of values is in the 0 – 5,000 partition. However, each partition spans quite a large range (a range of \$5,000).



The plot also includes a line plot showing the so called **density estimate** and is a more accurate display of the actual (at least estimated true) distribution of the data (the values of **Income**). It allows us to see that rather than values in the range 0 – 5,000 occurring frequently, in fact there is a much smaller range (perhaps 3,000 – 5,000) that occurs very frequently.

The third element of the plot is the so called *rug* along the bottom of the plot. The rug is a single dimension plot of the data along the number line. It is useful in seeing exactly where data points actually lay. For large collections of data with a relatively even spread of values the rug ends up being quite black, as is the case here, up to about \$25,000. Above about \$35,000 we can see that there is only a splattering of entities with such values. In fact, from the Summary option, using the Describe check button, we can see that the highest values are actually \$36,1092.60, \$38,0018.10, \$39,1436.70, \$40,4420.70, and \$42,1362.70.

This second plot, showing the distribution for the variable **Hours**, illustrates a more **normal distribution**. It is, roughly speaking, a distribution with a peak in the middle and diminishing on both sides, with regards the frequency. The density plot shows that it is not a very strong normal distribution, and the rug plot indicates that the data take on very distinct values (i.e., one would suggest that they are integer values, as is confirmed through viewing the textual summaries in the Summary option).



4.2.3 Cumulative Distribution Plot

CUMULATIVE PLOTS OF INCOME AND HOURS TO GO HERE.

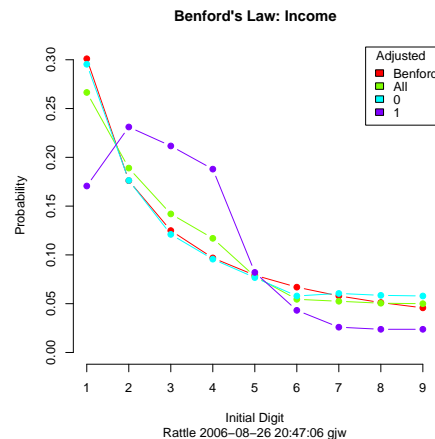
Togaware Watermark For TeX Catalogue Survival

4.2.4 Benford's Law

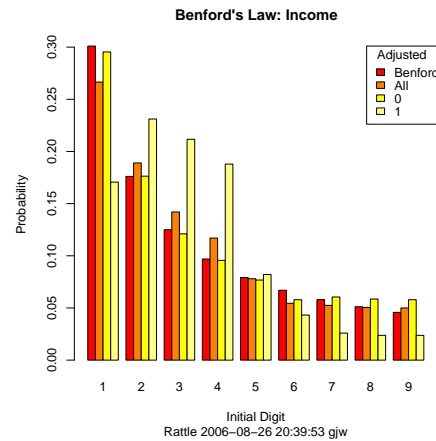
The use of **Benford's Law** has proven to be effective in identifying oddities in data. For example, it has been used for sample selection in fraud detection. Benford's law relates to the frequency of occurrence of the first digit in a collection of numbers. In many cases, the digit '1' appears as the first digit of the numbers in the collection some 30% of the time, whilst the digit '9' appears as the first digit less than 5% of the time. This rather startling observation is certainly found, empirically, to hold in many collections of numbers, such as bank account balances, taxation refunds, and so on. By plotting a collection of numbers against the expectation as based on Benford's law, we are able quickly ascertain any odd behaviour in the data.

Benford's law is not valid for all collections of numbers. For example, people's ages would not be expected to follow Benford's Law, nor would telephone numbers. So use the observations with care.

You can select any number of continuous variables to be compared with Benford's Law. By default, a line chart is used, with the red line corresponding to the expected frequency for each of the initial digits. In this plot we have requested that Income be compared to Benford's Law. A Target variable has been identified (in the Variables tab) and so not only is the whole population's distribution of initial digits compared to Benford's Law, but so are the distributions of the subsets corresponding to the different values of the target variable. It is interesting to observe here that those cases in this dataset that required an adjustment after investigation (*Adjustment* = 1) conformed much less to Benford's Law than those that were found to require no adjustment (*Adjustment* = 0). In fact, this latter group had a very close conformance to Benford's Law.

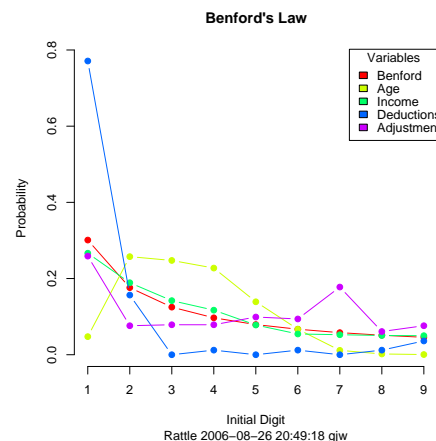


By selecting the Benford Bars option a bar chart will be used to display the same information. The expected distribution of the initial digit of the numbers under consideration, according to Benford's Law, is once again shown as the initial red bar in each group. This is followed by the population distribution, and then the distribution for each of the sub-populations corresponding to the value of the Target variable. The bar chart again shows a very clear differentiation between the adjusted and non-adjusted cases.



Some users find the bar chart presentation more readily conveys the information, whilst many prefer the less clutter and increased clarity of the line chart. Regardless of which you prefer, Rattle will generate a single plot for each of the variables that have been selected for comparison with Benford's Law.

In the situation where no target variable has been identified (either because, for the dataset being explored, there is no target variable or because the user has purposely not identified the target variable to Rattle) and where a line chart, rather than a bar chart, is requested, the distribution of all variables will be displayed on the one plot. This is the case here where we have chosen to explore Age, Income, Deductions, and Adjustment.



This particular exploration of Benford's Law leads to a number of interesting observations. In the first instance, the variable *Age* clearly does not conform. As mentioned, age is not expected to conform since it is a number series that is constrained

in various ways. In particular, people under the age of 20 are very much under-represented in this dataset, and the proportion of people over 50 diminishes with age.

The variable *Deductions* also looks particularly odd with numbers beginning with '1' being way beyond expectations. In fact, numbers beginning with '3' and beyond are very much under-represented, although, interestingly, there is a small surge at '9'. There are good reasons for this. In this dataset we know that people are claiming deductions of less than \$300, since this is a threshold in the tax law below which less documentation is required to substantiate the claims. The surge at '9' could be something to explore further, thinking perhaps that clients committing fraud may be trying to push their claims as high as possible (although there is really no need, in such circumstances, to limit oneself, it would seem, to less than \$1000).

By exploring this single plot (i.e., without partitioning the data according to whether the case was adjusted or not) we see that the interesting behaviours we observed with relation to *Income* have disappeared. This highlights a point that the approach of exploring Benford's Law may be of most use in exploring the behaviours of particular sub-populations.

Note that even when no target is identified (in the Variables tab) and the user chooses to produce Benford Bars, a new plot will be generated for each variable, as the bar charts can otherwise become quite full.

Benford's Law primarily applies to the first digit of the numbers. A similar, but much less strong, law also applies to the second, third and fourth digits. In particular, the second digit distributions are approximately XXXX. However, as we proceed to the third and fourth and so on, each has an expected frequency pretty close to 0.1 (or 10%), indicating they are all generally equally likely.

4.2.5 Bar Plot

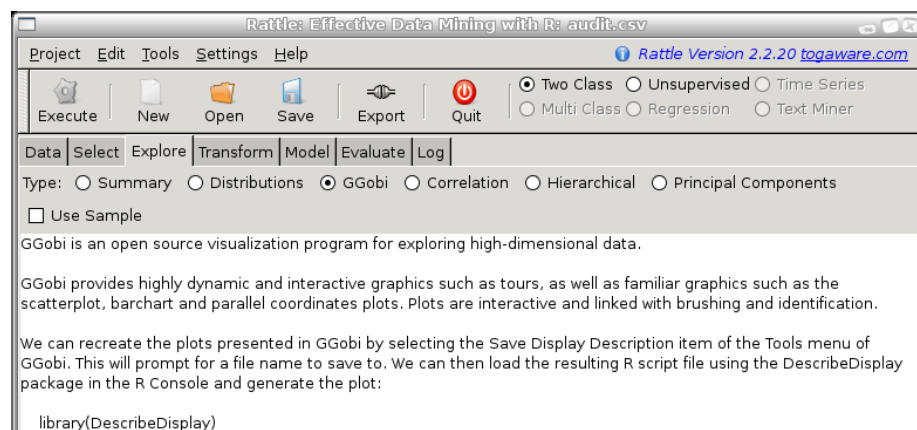
Togaware Watermark For TeX Catalogue Survival

4.2.6 Dot Plot

Togaware Watermark For TeX Catalogue Survival

4.3 GGobi Option

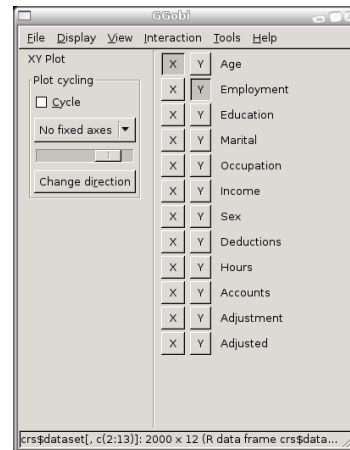
GGobi provides quite sophisticated visualisation tools as an adjunct to Rattle basic visualisations provided. To use the *GGobi* option the GGobi application will need to be installed on your system. GGobi runs under GNU/Linux, OS/X, and MS/Windows and is available for download from <http://www.ggobi.org/>.



GGobi is very powerful indeed, and here we only cover the basic functionality. With GGobi we are able to explore high-dimensional data through highly dynamic and interactive graphics such as tours, scatterplots, bar-charts and parallel coordinates plots. The plots are interactive and linked with brushing and identification. The available functionality is extensive, but includes being able to review entities with low or high values on particular variables and to view values for other variables through brushing in linked plots. Panning and zooming is supported. Data can be rotated in 3D, and we can tour high dimensional data through 1D, 2D, and 2x1D projections, with manual and automatic control of projection pursuits.

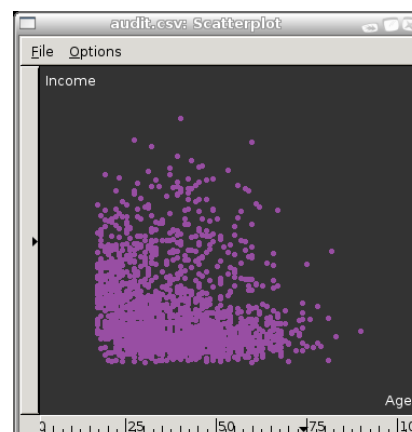
4.3.1 Scatterplot

When you startup GGobi (Execute the *GGobi* option) two windows will appear: one to control the visualisations and the other to display the default visualisation (a two variable scatterplot). The control window is as displayed to the right. It is a basic window with menus that provide the overall control of the visualisations. Below the menu bar you will see *XY Plot* which tells us that we are displaying a two variable scatterplot. On the right hand side is a list of the variables from your dataset, together with buttons to choose which variable to plot as the X and the Y.



By default, the first (*Age*) and second (*Employment*) are chosen. You can choose any of your variables to be the X or the Y by clicking the appropriate button. This will change what is displayed in the plot.

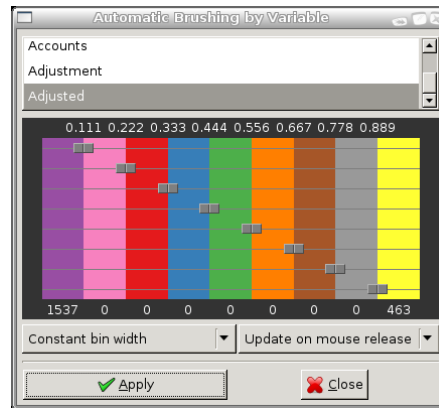
From the *Display* menu you can choose a *New Scatterplot Display* so that you can have two (or more) plots displayed at a time. At any one time just one plot is the current plot (as indicated in the title) and you can make a plot current by clicking in it.



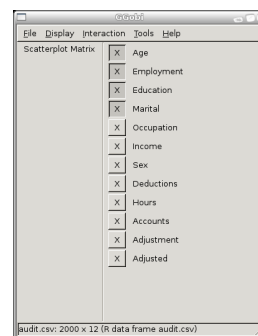
4.3 GGobi Option

63

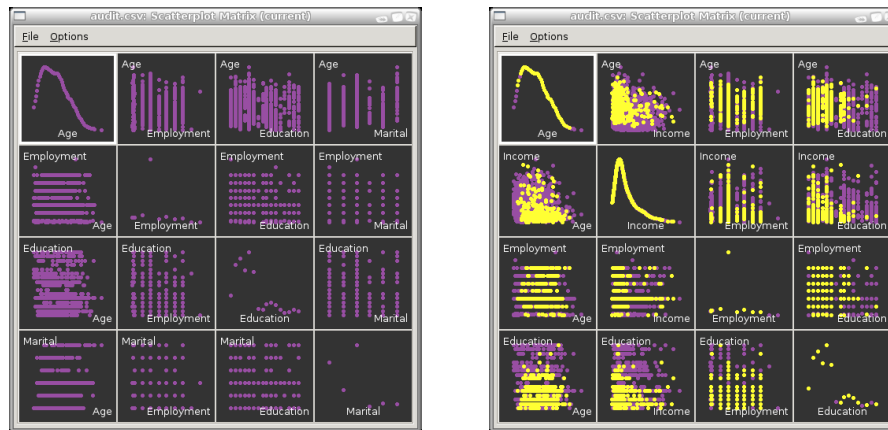
For our purposes we are usually most interested in the relationship between the values of the variables for entities that have an **Adjusted** value of 1 or 0. We can have these highlighted in different colours very easily. From the Tools menu choose Automatic Brushing. From the variables list at the top of the resulting popup window choose **Adjusted**. Now click on the Apply button and you will see that the 1,537 points that have a value of 0 for **Adjusted** remain purple, whilst those 463 entities that have a value of 1 are now yellow. This will apply to all displayed plots.



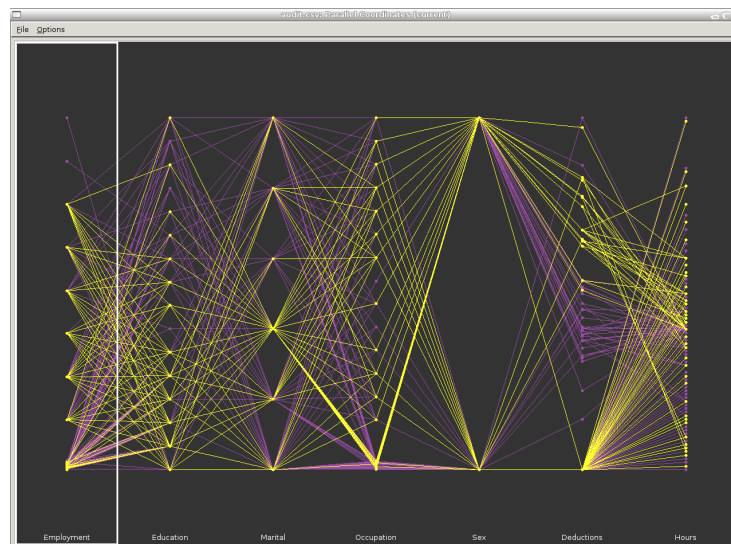
The Display menu provides a number of other options. The Scatterplot Matrix, for example, can be used to display a matrix of scatterplots across many variables at the one time. By default, the first four variables are displayed, as illustrated here, but we can add and remove variables by selecting the appropriate buttons in the control window (which is now displaying only the choice of X variables). You can also use the Automatic Brushing that we illustrated above to highlight



the adjusted cases. Such matrix scatterplots are effective in providing an overview of the distributions of our data.

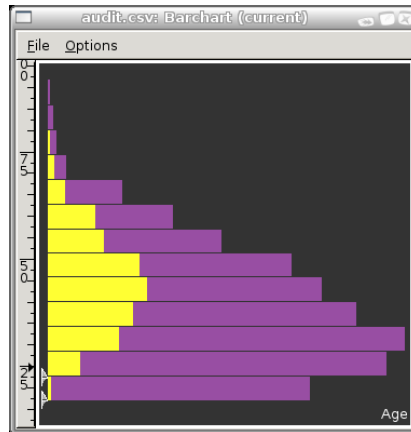


INCLUDE PARALLEL COORDINATES



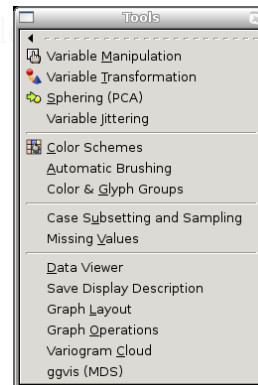
INCLUDE BAR CHART PLUS THE INTERACTIVE BINNING

The thing to note here is the two arrows down the bottom left side of the plot. Drag these around to get different width bars.

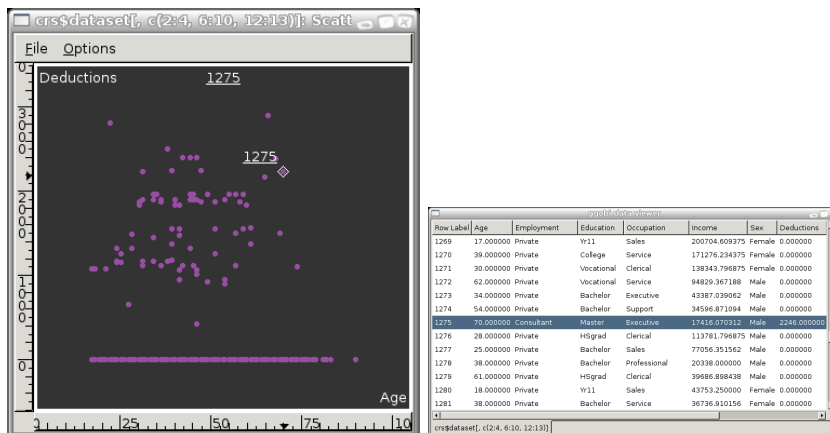
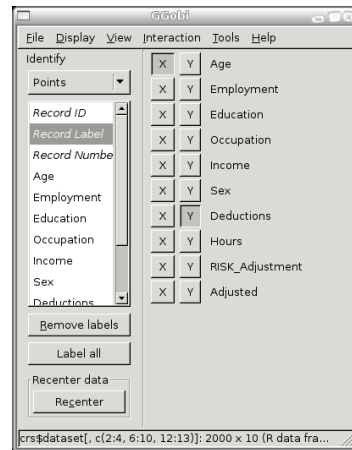


4.3.2 Data Viewer: Identifying Entities in Plots

Often we are interested in viewing the actual data/entities associated with points in our plots. The Tools menu provides many options and operations for visualising our data. In particular it provides access to a number of GGobiplugins. Some basic information about plugins is available from the Help menu, selecting the About Plugins item. The actual plugin we are interested in is the Data Viewer. Selecting this item from the Tools menu will display a textual view of the data, showing the row numbers and the column names, and allowing us to sort the rows by clicking on particular column names.



To make most use of the Data Viewer we will want to use the Identify option available under the Interaction menu. This will change the Control window to display the Identify controls as shown here. The Data Viewer is then linked to the Plot, allowing us to select a particular row in the Data Viewer to have the corresponding entity identified in the current Plot. Similarly, we can mouse over the Plot to have the individual points identified (with their row number) as well as displaying to the entity within the Data Viewer. Within the Plot display we can also right mouse button a point to have it's row number remain within the plot (useful when printing to highlight particular points). The right mouse button on the same point will remove the row number display.



4.3.3 Other Options

The Variable Manipulation option allows variables to be rescaled, cloned and manipulated and new variables to be created. The Transform Variables option allows us to view various transformations of the variables, including Log transforms, rescaling, and standardisation. Sphering performs a visual principal components analysis. The Color Scheme option

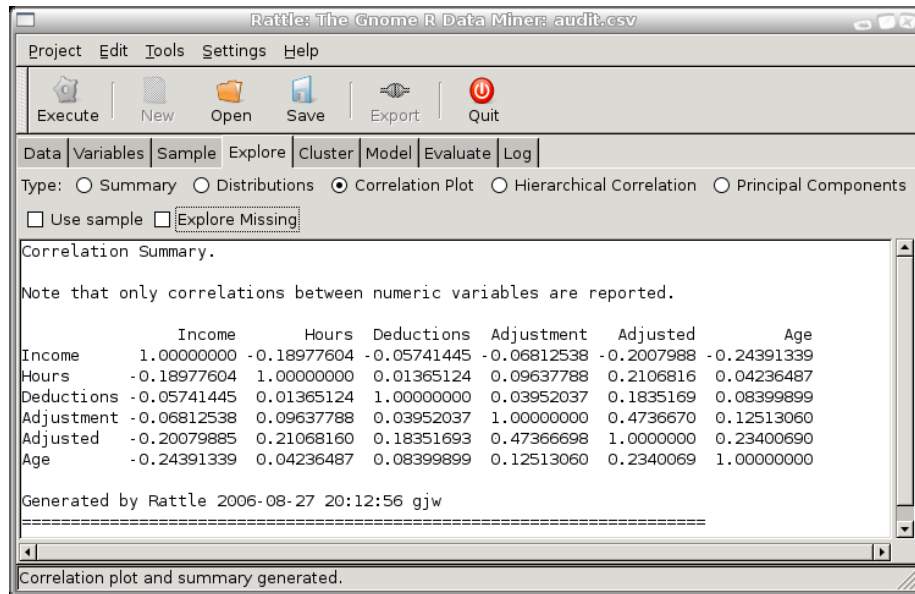
allows us to change the colours used in our plots. Automatic Brushing will colour various ranges of the values of each variable. Subsetting and Sampling allows us to choose subsets of the whole dataset using different methods, including random sampling, selection of blocks of data, every n th entity, and several others. There are also options to specify Graph Layout and Graph Options.

4.3.4 Further GGobi Documentation

We have only really scratched the surface of using GGobi here. There is a lot more functionality available, and whilst the functionality that is likely to be useful for the data miner has been touched on, there is a lot more to explore. So do explore the other features of GGobi as some will surely be useful for new tasks.

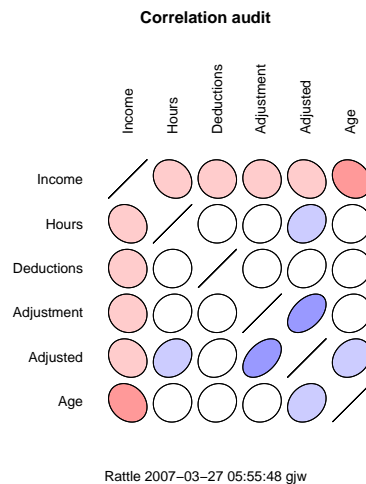
A full suite of documentation for GGobi is available from the GGobi web site at <http://www.ggobi.org/>, including a tutorial introduction and a complete book. These provide a much more complete treatise of the application.

4.4 Correlation Option



A correlation plot will display correlations between the values of variables in the dataset. In addition to the usual correlation calculated between values of different variables, the correlation between missing values can be explored by checking the Explore Missing check box.

The first thing to notice for this correlation plot is that only the numeric variables appear. Rattle only computes correlations between numeric variables at this time. The second thing to note about the graphic is that it is symmetric about the diagonal. The correlation between two variables is the same, irrespective of the order in which we view the two variables. The



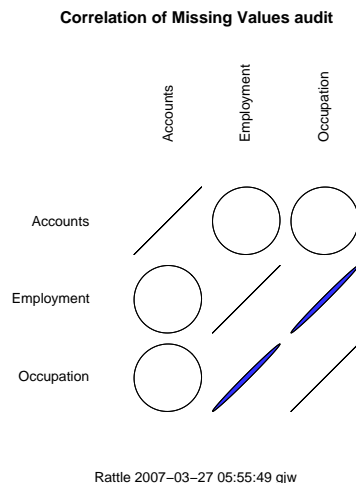
third thing to note is that the order of the variables does not correspond to the order in the dataset, but to the order of the strength of any correlations, from the least to the greatest. This is done simply to achieve a more pleasing graphic which is easier to take in.

We interpret the degree of any correlation by both the shape and colour of the graphic elements. Any variable is, of course, perfectly correlated with itself, and this is reflected as the diagonal lies on the diagonal of the graphic. Where the graphic element is a perfect circle, then there is no correlation between the variables, as is the case in the correlation between Hours and Deductions—although in fact there is a correlation, just a very weak one.

The colours used to shade the circles give another (if perhaps redundant) clue to the strength of the correlation. The intensity of the colour is maximal for a perfect correlation, and minimal (white) if there is no correlation. Shades of red are used for negative correlations and blue for positive correlations.

By selecting the Explore Missing check box you can obtain a correlation plot that will show any correlations between the missing values of variables. This is particularly useful to understand how missing values in one variable are related to missing values in another.

We notice immediately that only three variables are included in this correlation plot. Rattle has identified that the other variables in fact have no missing values, and so there is no point including them in the plot. We also notice that a categorical variable, Accounts, is included in the plot even though it was not included in the usual correlation plot. In this case we can obtain a correlation for categorical variables since we only measure missing and presence of a value, which is easily interpreted as numeric.

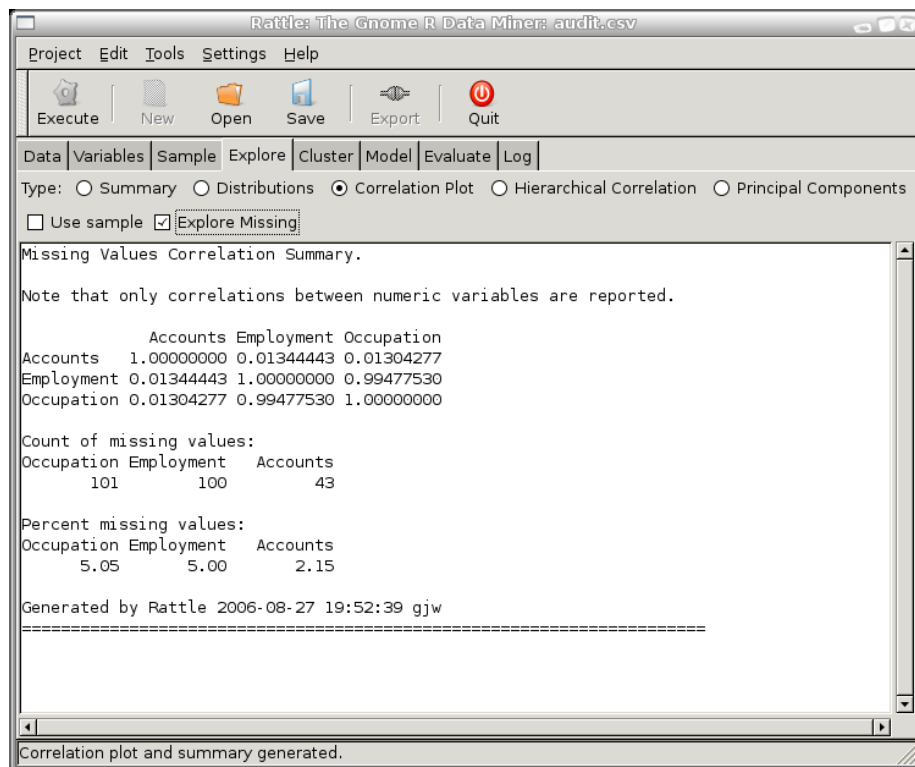


The graphic shows us that Employment and Occupation are highly correlated in their presence of missing values. That is, when Employment has a missing value, so does Occupation, and vice versa, at least in general. The actual correlation is 0.995 (which can be read from the Rattle text view window), which is very close to 1.

On the other hand, there is no (in fact very little at 0.013) correlation between Accounts and the other two variables, with regard missing values.

It is important to note that the correlations showing missing values may be based on very small samples, and this information is included in the text view of the Rattle window. For example, in this example we can see that there are only 100, 101, and 43 missing values, respectively, for each of the three variables having any missing values. This corresponds to approximately 5%, 5%, and 2% of the entities, respectively, having missing values for these variables.

Logaware Watermark For TeX Catalogue Survival



4.4 Correlation Option

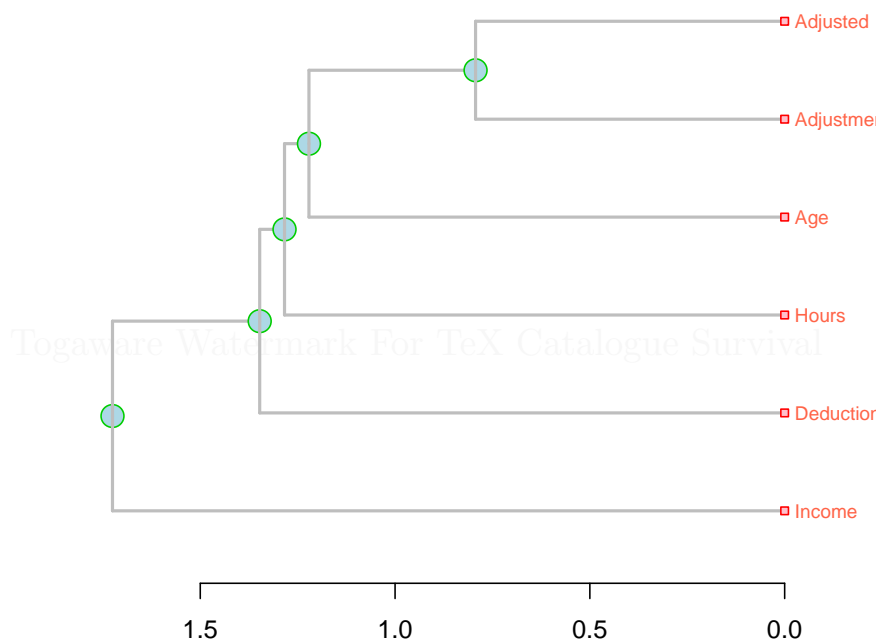
71

Rattle uses the default R correlation calculation known as Pearson's correlation, a common measure of correlation.

Togaware Watermark For TeX Catalogue Survival

4.4.1 Hierarchical Correlation

Variable Correlation Clusters audit.csv



Rattle 2007-03-27 06:04:59 gjw

4.4.2 Principal Components

4.5 Single Variable Overviews

Chapter 5

Transform

5.1 Impute

Fogaware Watermark For TeX Catalogue Survival

Imputation is the process of filling in the gaps (or missing values) in data. Often, data will contain missing values, and this can cause a problem for some modelling algorithms. For example, the random forest option silently removes any entity with any missing value! For datasets with a very large number of variables, and a reasonable number of missing values, this may well result in a small, unrepresentative dataset, or even no dataset at all!

There are many types of imputations available, and there is always discussion about whether it is a good idea or not. After all, we end up inventing data to suit the needs of the tool we are using. We won't discuss this issue so much, but concentrate on how we might impute values.

The simplest of imputation involves simply replacing all the missing values for a variable with a single value. This might be 0 for a numeric variable (and indeed, 0 may actually make sense in many cases—for example, number of children, if missing, is assumed to be 0). Or else it might be the variable mean or median (depending on how the data is distributed). Or we might be more sophisticated and use the average value of the k nearest neighbours, where the neighbours are determined by looking at the other variables.

When Rattle performs an imputation it will store the results in a variable of the dataset which has the same name as the variable that is imputed, but prefixed with `IMP_`. Such variables, whether they are imputed by Rattle or already existed in the dataset loaded into Rattle (e.g., a dataset from SAS), will be treated as input variables, and the original variable marked to be ignored.

5.1.1 Zero/Missing

The simplest approach to imputation is to replace missing values with a 0 (for numeric data) and the class *Missing* for categorical data.

5.1.2 Mean/Median

Often a simple, if not always satisfactory, choice for missing values is to use some “central” value of the variable. This is often the mean or median. We might choose to use the mean, for example, if the variable is otherwise generally normally distributed (and in particular does not have any skewness). If the data does exhibit some skewness though (e.g., there are a small number of very large values) then the median might be a better choice.

This is achieved in R with

```
crs$dataset[is.na(crs$dataset$Age), "Age"] <- mean(crs$dataset$Age, na.rm=T)
crs$dataset[is.na(crs$dataset$Age), "Age"] <- median(crs$dataset$Age, na.rm=T)
```

Whilst this is a simple and computationally quick approach, it is a very blunt approach to imputation and can lead to poor performance from the resulting models.

Refer to [Data Mining With R](#), from page 42, for more details.

5.1.3 Nearest Neighbours

Another approach to filling in the missing values is to look at the entities that are closest to the entity with a missing value, and to use the values for the missing variable of these nearby neighbours to fill in the missing

5.1 Impute

75

value for this entity. See Refer to [Data Mining With R](#), page 48 and following for example R code to do this.

Togaware Watermark For TeX Catalogue Survival

Togaware Watermark For TeX Catalogue Survival

Chapter 6

Building Models

In this chapter we present a framework within which we cast the task of data mining—the task being model building. We refer to an algorithm for building a model as a *model builder*. Rattle supports a number of model builders, including decision tree induction, boosted decision trees, random forests, support vector machines, logistic regression, kmeans, and association rules. In essence, the model builders differ in how they represent the models they build (i.e., the discovered knowledge) and how they find (or search for) the best model within this representation.

We can think of the discovered knowledge, or the *model*, as being expressed as sentences in a language. We are familiar with the fact that we express ourselves using sentences in our own specific human languages (whether that be English, French, or Chinese, for example). As we know, there is an infinite number of sentences that we can construct in our human languages.

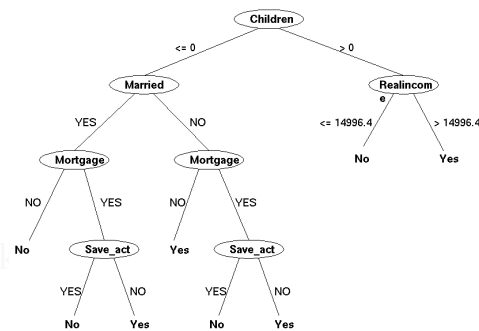
The situation is similar for the “sentences” we construct through using model builders—there is generally an infinite number possible sentences. In human language we are generally very well skilled at choosing sentences from this infinite number of possibilities to best represent what we would like to communicate. And so it is with model building. The skill is to express within the language chosen the best sentences that capture what it is we are attempting to model.

We will consider each of the model builders deployed in Rattle and char-

acterise them through the sentences they generate and how they search for the best sentences that capture or summarises what the data is indicating. We conclude the chapter with an overview of the issues around selecting the best model builder.

6.1 Decision Tree

One of the classic machine learning techniques, widely deployed in data mining, is decision tree induction. Using a simple algorithm and a simple knowledge structure, the approach has proven to be very effective. These simple tree structures represent a classification (and



regression) model. Starting at the root node, a simple question is asked (usually a test on a variable value, like *Age < 35*). The branches emanating from the node correspond to alternative answers. For example, with a test of *Age < 35* the alternatives would be *Yes* and *No*. Once a leaf node is reached (one from which no branches emanate) we take the decision or classification associated with that node. Some form of probability may also be associated with the nodes, indicating a degree of certainty for the decision. Decision tree algorithms handle mixed types of variables, handle missing values, are robust to outliers and monotonic transformations of the input, and robust to irrelevant inputs. Predictive power tends to be poorer than other techniques.

The model is expressed in the form of a simple decision tree (the knowledge representation). At each node of the tree we test the value of one of the variables, and depending on its value, we follow one of the branches emanating from that node. Thus, each branch can be thought of as having a test associated with it, for example *Age < 35*. This branch then leads to another node where there will be another variable to test, and so on, until we reach a leaf node of the tree. The leaf node represents the decision to be made. For example, it may be a *yes* or *no* for deciding

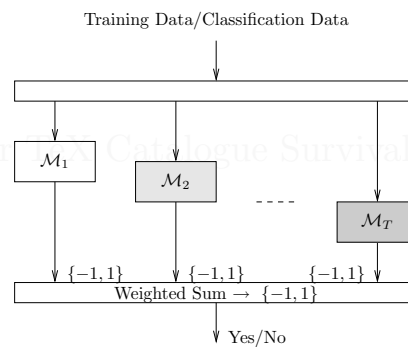
whether an insurance claim appears to be fraudulent.

In searching for a decision tree to best model our data, alternative decision trees are considered in a top-down fashion, beginning with the decision of the variable to initially partition the data (at the root node).

FURTHER DETAILS HERE

6.2 Boosting

The **Boosting** meta-algorithm is an efficient, simple, and easy to program learning strategy. The popular variant called **AdaBoost** (an abbreviation for Adaptive Boosting) has been described as the “best off-the-shelf classifier in the world” (attributed to Leo Breiman by [Hastie et al. \(2001, p. 302\)](#)). **Boosting** algorithms build multiple models from a dataset, using some other learning



algorithm that need not be a particularly good learner. Boosting associates weights with entities in the dataset, and increases (boosts) the weights for those entities that are hard to accurately model. A sequence of models is constructed and after each model is constructed the weights are modified to give more weight to those entities that are harder to classify. In fact, the weights of such entities generally oscillate up and down from one model to the next. The final model is then an additive model constructed from the sequence of models, each model’s output weighted by some score. There is little tuning required and little is assumed about the learner used, except that it should be a weak learner! We note that boosting can fail to perform if there is insufficient data or if the weak models are overly complex. Boosting is also susceptible to noise.

Boosting builds a collection of models using a “weak learner” and thereby reduces misclassification error, bias, and variance ([Bauer and Kohavi, 1999](#); [Schapire et al., 1997](#)). Boosting has been implemented in, for

example, **C5.0**. The term originates with **Freund and Schapire (1995)**.

The algorithm is quite simple, beginning by building an initial model from the training dataset. Those entities in the training data which the model was unable to capture (i.e., the model mis-classifies those entities) have their weights boosted. A new model is then built with these boosted entities, which we might think of as the problematic entities in the training dataset. This model building followed by boosting is repeated until the specific generated model performs no better than random. The result is then a panel of models used to make a decision on new data by combining the “expertise” of each model in such a way that the more accurate experts carry more weight.

As a meta learner Boosting employs some other simple learning algorithm to build the models. The key is the use of a weak learning algorithm—essentially any weak learner can be used. A weak learning algorithm is one that is only somewhat better than random guessing in terms of error rates (i.e., the error rate is just below 50%). An example might be decision trees of depth 1 (i.e., decision stumps).

6.3 Random Forest

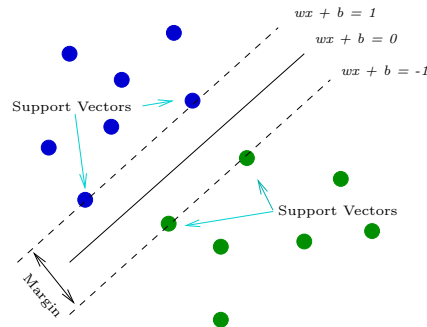
The approach taken by **random forests** is to build multiple decision trees (often many hundreds) from different subsets of entities from the dataset and from different subsets of the variables from the dataset, to obtain substantial performance gains over single tree classifiers. Each decision tree is built from a sample of the full dataset, and a random sample of the available variables is used for each node of each tree. Having built an ensemble of models the final decision is the majority vote of the models, or the average value for a regression problem. The generalisation error rate from random forests tends to compare favourably to boosting approaches, yet the approach is more robust to noise in the data.



FURTHER DETAILS OF THE SEARCH

6.4 Support Vector Machine

A Support Vector Machine (SMV) searches for so called *support vectors* which are data points that are found to lie at the edge of an area in space which is a boundary from one class of points to another. In the terminology of SVM we talk about the space between regions containing data points in different classes as being the margin between those classes. The support vectors are used to identify a hyperplane (when we are talking about many dimensions in the



data, or a line if we were talking about only two dimensional data) that separates the classes. Essentially, the maximum margin between the separable classes is found. An advantage of the method is that the modelling only deals with these support vectors, rather than the whole training dataset, and so the size of the training set is not usually an issue. If the data is not linearly separable, then kernels are used to map the data into higher dimensions so that the classes are linearly separable. Also, Support Vector Machines have been found to perform well on problems that are non-linear, sparse, and high dimensional. A disadvantage is that the algorithm is sensitive to the choice of parameter settings, making it harder to use, and time consuming to identify the best.

Support vector machines do not predict probabilities but rather produce normalised distances to the decision boundary. A common approach to transforming these decisions to probabilities is by passing them through a sigmoid function.

6.5 Logistic Regression

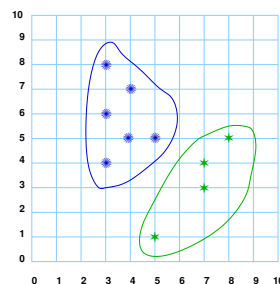
Logistic regression is a statistical model builder using traditional regression techniques but for predicting a 1/0 outcome. Logistic regression in fact builds a type of generalized linear model, using a so called logit function.



Togaware Watermark For TeX Catalogue Survival

6.6 KMeans

Clustering is one of the core tools used by the data miner. Clustering allows us, in a generally unguided fashion, to group entities according to how similar they are. This is done on the basis of a measure of distance between entities. The aim of clustering is to identify groups of entities that are close together but as a group are quite separate from other groups.

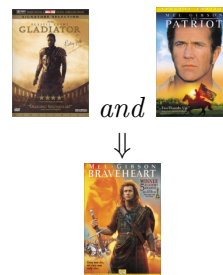


DETAILS REQUIRED

Togaware Watermark For TeX Catalogue Survival

6.7 Association Rules

Association analysis identifies relationships or affinities between entities and/or between variables. These relationships are then expressed as a collection of association rules. The approach has been particularly successful in mining very large transaction databases and is one of the core classes of techniques in data mining. A typical example is in the retail business where historic data might identify that customers who purchase the Gladiator DVD and the Patriot DVD also purchase the Braveheart DVD. The historic data might indicate that the first two DVDs are purchased by only 5% of all customers. But 70% of these then also purchase Braveheart. This is an *interesting* group of customers. As a business we may be able to take advantage of this observation by targetting advertising of the Braveheart DVD to those customers who have purchased both Gladiator and Patriot.



DETAILS OF REPRESENTATION AND SEARCH REQUIRED HERE.

6.8 Model Selection

The question that obviously now comes to mind is which model builder do we use. That is a question that has challenged us for a long time, and still there remains no definitive answer. It all depends on how well the model builder works on your data, and, in fact, how you measure the performance on the model. We review some of the insights that might help us choose the right model builder and, indeed, the right model, for our task.

Contrary to expectations, there are few comprehensive comparative studies of the performance of various model builders. A notable exception is the study by Caruana and Niculescu-Mizil, who compared most modern model builders across numerous datasets using a variety of performance measures. The key conclusion, they found, was that boosted trees and random forests generally perform the best, and that decision trees, logistic regression and boosted stumps generally perform the worst. Perhaps more importantly though, it often depends on what is being measured as the performance criteria, and on the characteristics of the data being modelled.

An overall conclusion from such comparative studies, then, is that often it is best to deploy different model builders over the dataset to investigate which performs the best. This is better than a single shot at the bullseye. We also need to be sure to select the appropriate criteria for evaluating the performance. The criteria should match the task at hand. For example, if the task is one of information retrieval then a Precision/Recall measure may be best. If the task is in the area of health, then the area under the ROC curve is an often used measure. For marketing, perhaps it is lift. For risk assessment, the Risk Charts are a good measure.

So, in conclusion, it is good to build multiple models using multiple model builders. The tricky bits are tuning the model builders (requiring an understanding of the sometimes very many and very complex model builder parameters) and selecting the right criteria to assess the performance of the model (a criteria to match the task at hand—noting that raw accuracy is not always, and maybe not often, the right criteria).

6.9 Bibliographic Notes

[Caruana and Niculescu-Mizil \(2006\)](#) present a comprehensive empirical comparison of many of the modern model builders. An older comparison is known as the Statlog comparison ([King et al., 1995](#)).

Togaware Watermark For TeX Catalogue Survival

Togaware Watermark For TeX Catalogue Survival

Chapter 7

Two Class Models

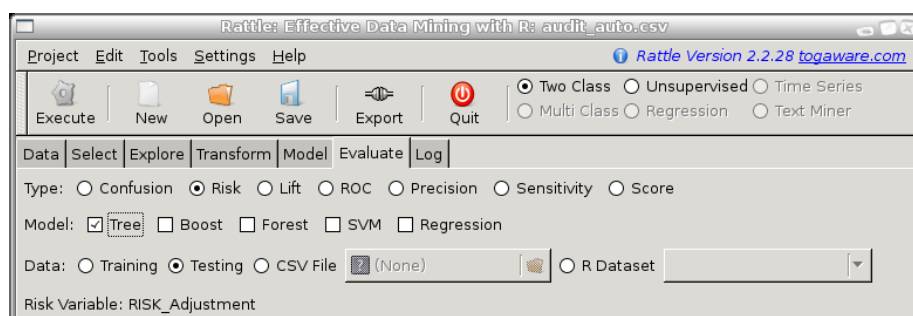
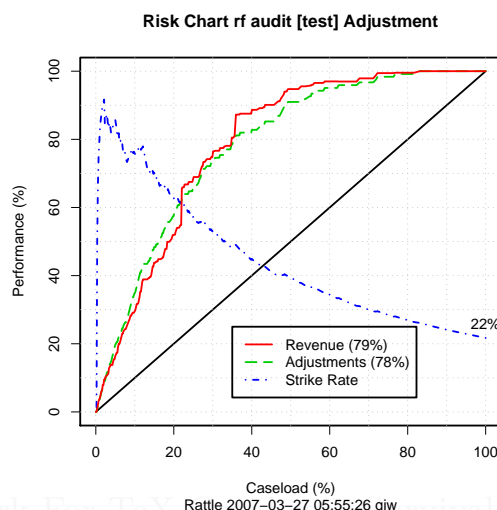
This chapter focuses on the common data mining task of binary (or two class) classification. This is the task of distinguishing between two classes of entities - whether they be high risk and low risk insurance clients, productive and unproductive audits, responsive and non-responsive customers, successful and unsuccessful security breaches, and many other similar examples.

Rattle provides a straight-forward interface to the collection of model builders commonly used in data mining. For each, a basic collection of the commonly used tuning parameters is exposed through the interface for fine tuning the model performance. Where possible, Rattle attempts to present good default values to allow the user to simply build a model with no or little tuning. This may not always be the right approach, but is certainly a good place to start.

The model builders provided by Rattle are: Decision Trees, Boosted Decision Trees, Random Forests, Support Vector Machines, and Logistic Regression.

7.1 Risk Charts

We have mentioned in passing, particularly in the previous chapter, the idea of a risk chart for evaluating the performance of our models. A risk chart is somewhat similar in concept to a number of the other evaluation approaches, particularly the ROC curves, which have also been mentioned and will be covered in detail in Chapter 12. We formally introduce the risk chart here (rather than in Chapter 12) in order to be able to discuss the model builders in practise, and in particular illustrate their performance. The risk charts can be displayed within Rattle, once we have built our models, by choosing the Risk option of the Evaluate tab.



A risk chart is particularly useful in the context of the *audit* dataset, and for risk analysis tasks in general. Already we have noted that this dataset has a two class target variable, **Adjusted**. We have also identified a so called *risk* variable, **Adjustment**, which is a measure of the size of the risk associated with each entity. Entities that have no adjustment following an audit (i.e., they are clients who have supplied the correct information) will of course have no risk associated with them (**Adjustment** = 0).

Entities that do have an adjustment will have a risk associated with them, and for convenience we simply identify the value of the adjustment as the magnitude of the risk

In particular, we can think of revenue (or tax) authorities, where the outcomes of audits include a dollar amount by which the tax obligation of the taxpayer has been changed (which may be a change in favour of the revenue authority or in favour of the taxpayer). For fraud investigations, the outcome might be the dollar amount recovered from the fraudster. In these situations it is often useful to see the tradeoff between the return on investment and the number of cases investigated.

Rattle introduces the idea of a risk chart to evaluate the performance of a model in the context of risk analysis.

A risk chart plots performance against caseload. Suppose we had a population of just 100 entities (audit cases). The case load is the percentage of these cases that we will actually ask our auditors to process. The remainder we will not consider any further, expecting them to be low risk, and hence, with limited resources, not requiring any action. The decision as to what percentage of cases are actually actioned corresponds to the X axis of the risk chart - the caseload. A 100% caseload indicates that we will action all audit cases. A 25% caseload indicates that we will action just one quarter of all cases.

For a given testing population we know how many cases resulted in adjustments. We also know the magnitude of those adjustments (the risk). For a population of 100 cases, if we were to randomly choose 50% of the cases for actioning, we might expect to recover just 50% of the cases that actually did require an adjustment and 50% of the risk (or in this case, the revenue) recovered from these adjusted cases. Similarly for every caseload value: for a random caseload of 25% of the population we might expect to recover 25% of the adjustments and revenue. The diagonal black line of the risk chart represents this random selection, and can be thought of as the baseline against which to compare the performance of our models.

Any model that Rattle builds in the two-class paradigm generates a risk score for each entity. This is generally the probability of the case requiring an adjustment. We can use this score to sort all of the cases in decreasing order of the score. In selecting cases to be actioned, we then start

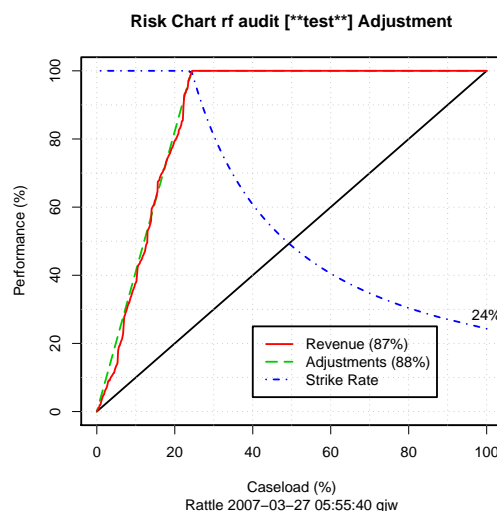
with those cases that have the highest score. Thus, in evaluating the performance of our model, the caseload axis represents the sorted list of cases, from the highest scored cases at the left (starting at 0% of the cases actioned), and the lowest at the right (ending with 100% of the cases actioned).

The green (dashed) line of a risk chart then records the percentage of adjusted cases that we would actually expect to identify for a given percentage of the caseload. In the risk chart above, for example, we can see that if we only actioned the top 20% of the cases, as scored by our model, we recover almost 60% of the cases that actually did require an adjustment. Similarly, for an 80% caseload we can recover essentially all of the cases that required adjustment. Hence our model can save us 20% of our caseload (i.e., 20% of our resources) whilst pretty much recovering all of the adjustments.

The red (solid) line similarly records the percentage of the total revenue (or risk) that is recovered for any particular caseload. In our example above we see that the red and green lines essentially follow each other. This is not always the case.

EXPLAIN THE REVENUE AND STRIKE RATES. EXPLAIN THE AUC.

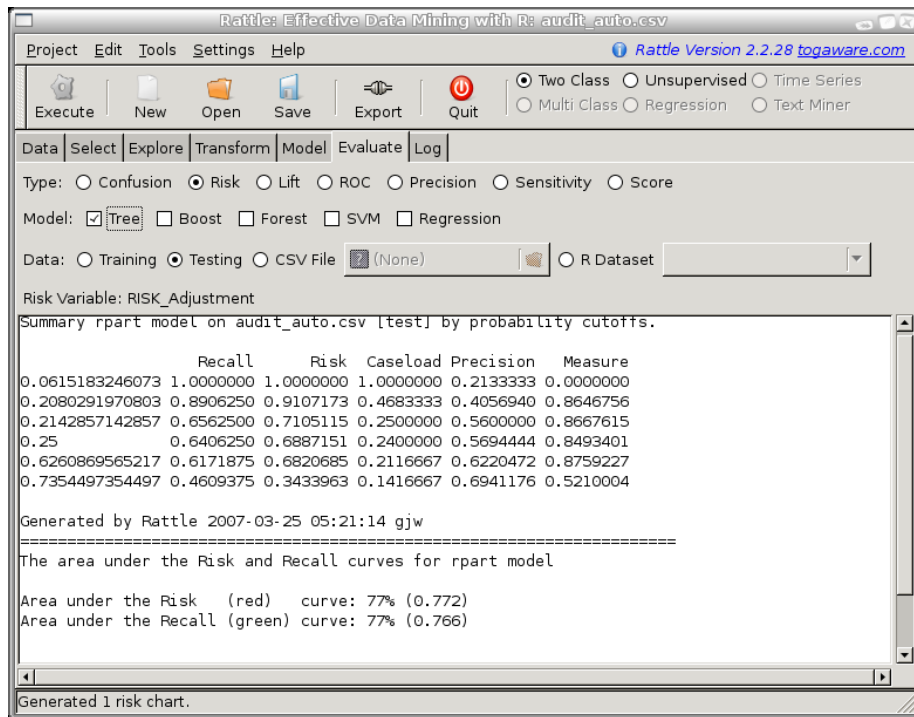
A perfect model performance assessed through a risk chart is then a risk chart that maximises the area under the two curves. We can illustrate such a risk chart by plotting the performance of a random forest model on the training data where a random forest often performs “perfectly,” as illustrated in this risk chart. The strike rate over the whole dataset in this case is 24% (as annotated at the right hand end of the strike rate line). That is, only 24% of



all of the cases in this dataset of audit cases actually required an adjustment. The perfect model accurately identifies all 24% of the cases (and 24% of the risk) with 24% of the caseload! Thus we see the performance plot having just two components: the essentially linear progression from a caseload of 0% and performance of 0% up to a caseload of 24% and performance of 100%, and then the flat 100% performance as we increase the caseload to 100%.

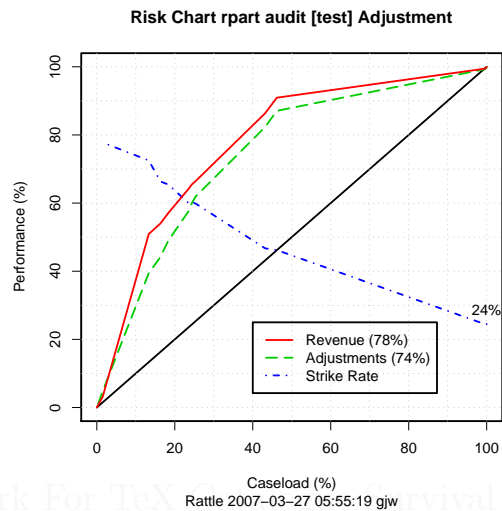
When a Risk Chart is generated the text window in **Rattle** will display the aggregated data that is used to construct the plot. This data consists of a row for each level of the probability distribution that is output from the model, ordered from the lowest probability value to a value of 1. For each row we record the model performance in terms of predicting a class of 1 if the probability cutoff was set to the corresponding value.

For example, we might choose a cutoff to be a probability of 0.28 so that anything predicted to be in class 1 with a probability of 0.28 or more will be regarded as in class 1. Then the number of predicted positives (or the Caseload) will be 30% (0.301667) of all cases. Amongst this 30% of cases are 69% of all true positives and they account for 79% of the total of the risk scores. The strike rate (number of true positives amongst the positives predicted by the model) is 61%. Finally, the measure reports the sum of the distances of the risk and recall from the baseline (the diagonal line). This measure can indicate the optimal caseload in terms of maximising both risk recovery and recall.

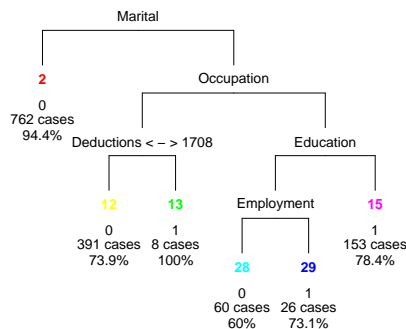


7.2 Decision Trees

Decision trees are the building blocks of data mining. Since their development back in the 1980's they have been the most widely deployed data mining model builder. The attraction lies in the simplicity of the resulting model, where a decision tree (at least one that is not too large) is quite easy to view, to understand, and, indeed, to explain to management! However, decision trees do not deliver the best performance in terms of the risk charts, and so there is a trade off between performance and simplicity of explanation and deployment.



Decision Tree audit \$ Adjusted



Rattle 2006-09-20 20:53:25 gjw

7.2.1 Priors

Sometimes the proportions of classes in a training set do not reflect their true proportions in the population. You can inform **Rattle** of the population proportions and the resulting model will reflect these.

The priors can be used to “boost” a particularly important class, by giving it a higher prior probability, although this might best be done through the Loss Matrix.

In **Rattle** the priors are expressed as a list of numbers that sum up to 1, and of the same length as the number of classes in the training dataset. An example for binary classification is `0.5,0.5`.

The default priors are set to be the class proportions as found in the training dataset.

7.2.2 Loss Matrix

The loss matrix is used to weight the outcome classes differently (the default is that all outcomes have the same loss of 1). The matrix will be constructed row-wise from the list of numbers we supply, and is of the same dimensions as the number of classes in the training dataset. Thus, for binary classification, four numbers must be supplied. The diagonal should be all zeros.

An example is: `0,10,1,0`, which might be interpreted as saying that an actual 1, predicted as 0 (i.e., a false negative) is 10 times more unwelcome than a false positive!

Rattle uses the loss matrix to alter the priors which will affect the choice of variable to split the dataset on at each node, giving more weight where appropriate.

7.2.3 Complexity

The complexity parameter (`cp`) is used to control the size of the decision tree and to select the optimal tree size. If the cost of adding another variable to the decision tree from the current node is above the value of

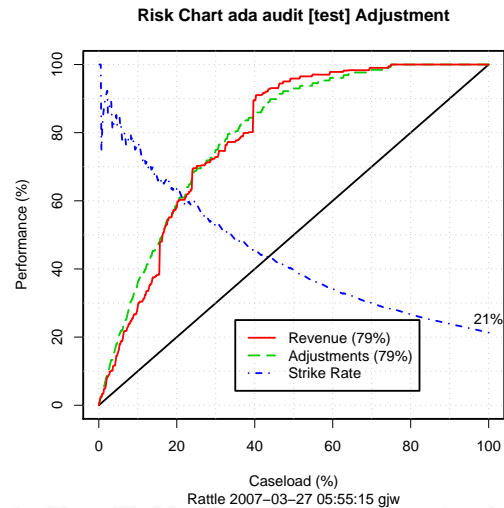
cp, then tree building does not continue. We could also say that tree construction does not continue unless it would decrease the overall lack of fit by a factor of cp.

Setting this to zero will build a tree to its maximum depth (and perhaps will build a very, very, large tree). This is useful if you want to look at the values for CP for various tree sizes. This information will be in the text view window. You will look for the number of splits where the sum of the xerror (cross validation error, relative to the root node error) and xstd is minimum. This is usually early in the list.

Togaware Watermark For TeX Catalogue Survival

7.3 Boosting

The ensemble approaches build upon the decision tree model builder by building many decision trees through sampling the training dataset in various ways. The *ada* boosting algorithm is deployed by *Rattle* to provide its boosting model builder. With the default settings a very reasonable model can be built. At a 60% caseload we are recovering 98% of the cases that required adjustment and 98% of the revenue.



The Boost functionality in *Rattle* allows the ensemble of trees to be added to. To do so, simply increase the Number of Trees, and click the Continue button.

7.4 Random Forests

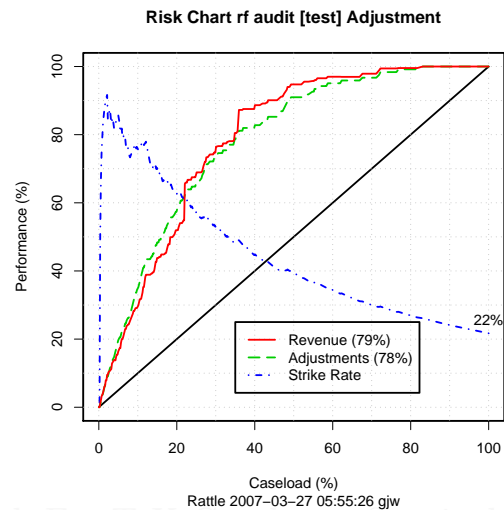
Random forests have generally been found to produce very reasonable models. A random forest, by building a large number of decision trees (from 500 or more), tends to be a very stable model, and not suffering the sensitivity that single decision tree induction suffers from.

Note and illustrate with the audit data how building a random forest and evaluate on training dataset

gives "perfect" results (that might make you wonder about it overfitting) but on test data you get a realistic (and generally good still) performance.

In RF if there are many noise variables, increase the number of variables considered at each node.

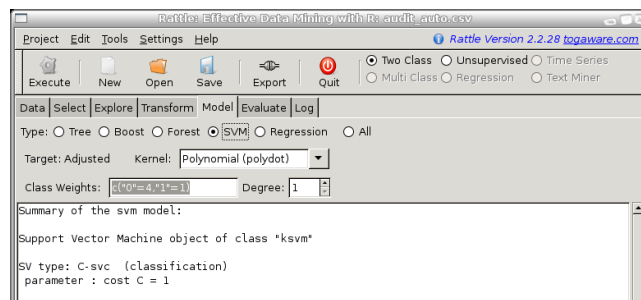
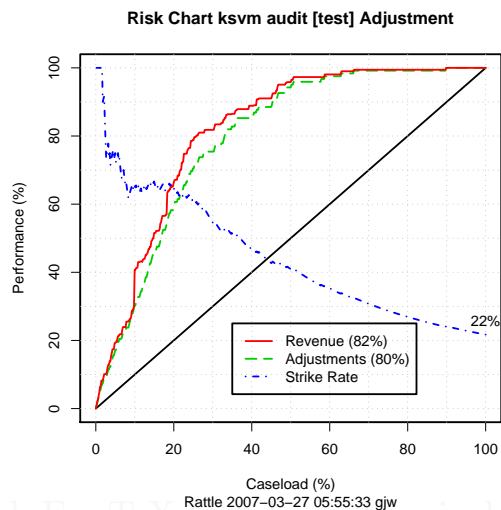
Also note that the Breiman-Cutler implementation as used in R appears to produce better results than those produced by the Weka implementation.



7.5 Support Vector Machines

Rattle supports the building of support vector machine (SVM) models using the *kernelab* package for R. This package provides an extensive collection of kernel functions, and a variety of tuning options. The trick with support vector machines is to use the right combination of kernel function and kernel parameters—and this can be quite tricky. Some experimentation with the

audit dataset, exploring different kernel functions and parameter settings, identified that a polynomial kernel function with the class weights set to `c("0"=4, "1"=1)` resulted in the best risk chart. For a 50% caseload we are recovering 94% of the adjustments and 96% of the revenue.



There are other general parameters that can be set for *ksvm*, including the cost of constraint violation (the trade-off between the training error and margin? could be from 1 to 1000, and is 1 by default).

For best results it is often a good idea to scale the numeric variables to have a mean of 0 and a standard deviation of 1.

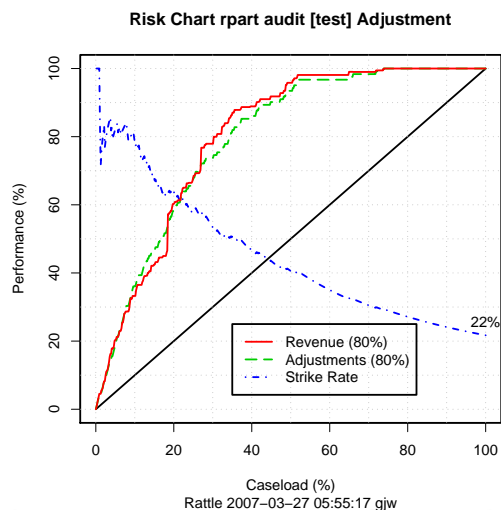
For polynomial kernels you can choose the degree of the polynomial. The default is 1. For the audit data as you increase the degree the model gets much more accurate on the training data but the model generalises less well, as exhibited by its performance on the test dataset.

Another parameter often needing setting for a radial basis function kernel is the sigma value. **Rattle** uses automatic sigma estimation (`sigest`) for this kernel, to find the best sigma, and so the user need not set a sigma value. If we wanted to experiment with various sigma values we can copy the R code from the Log tab and paste it into the R console, add in the additional settings, and run the model. Assuming the results are assigned into the variable `crs$ksvm`, as in the Log, we can evaluate the performance of this new model using the Evaluate tab.

Togaware Watermark For TeX Catalogue Survival

7.6 Logistic Regression

Logistic regression is the traditional statistical approach and indeed it can still produce good models as evidenced in the risk chart here. As noted in Section 6.8 though, logistic regression has not always been found to produce good models. Nonetheless, here we see a very good model that gives us an area under the curve of 80% for both Revenue and Adjustments, and at the 50% caseload we are recovering 94% of the cases requiring adjustment and 95% of the revenue associated with the cases that are adjusted.



For best results it is often a good idea to scale the numeric variables to have a mean of 0 and a standard deviation of 1.

7.7 Bibliographic Notes

The *ada* package for boosting was implemented by Mark Culp, Kjell Johnson, and George Michailidis, and is described in [Culp et al. \(2006\)](#).

Togaware Watermark For TeX Catalogue Survival

Togaware Watermark For TeX Catalogue Survival

Chapter 8

Unsupervised Modelling

8.1 Associate

Togaware Watermark For TeX Catalogue Survival

Association rules are one of the more common types of techniques most associated with data mining. Rattle supports association rules through the Associate tab of the Unsupervised paradigm.

Two types of association rules are supported. Rattle will use either the Ident and Target variables for the analysis if a market basket analysis is requested, or else will use the Input variables for a rules analysis.

8.1.1 Basket Analysis

The simplest association analysis is often referred to as market basket analysis. Within Rattle this is enabled when the Baskets button is checked. In this case, the data is thought of as representing shopping baskets (or any other type of collection of items, such as a basket of medical tests, a basket of medicines prescribed to a patient, a basket of stocks held by an investor, and so on). Each basket has a unique identifier, and the variable specified as an Ident variable in the Variables tab is taken as the identifier of a shopping basket. The contents of the basket are then the items contained in the column of data identified as the target variable. For market basket analysis, these are the only two variables

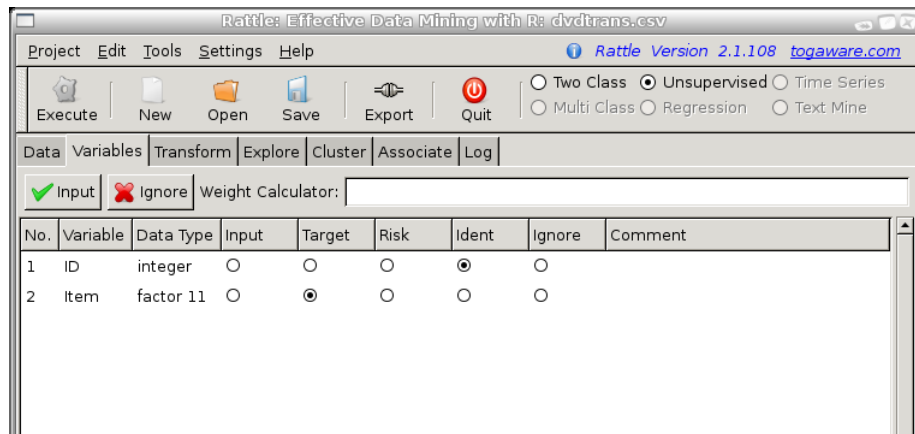
used.

To illustrate market basket analysis with **Rattle**, we will use a very simple dataset consisting of the DVD movies purchased by customers. Suppose the data is stored in the file `dvdtrans.csv` and consists of the following:

```
ID,Item
1,Sixth Sense
1,L0TR1
1,Harry Potter1
1,Green Mile
1,L0TR2
2,Gladiator
2,Patriot
2,Braveheart
3,L0TR1
3,L0TR2
4,Gladiator
4,Patriot
4,Sixth Sense
5,Gladiator
5,Patriot
5,Sixth Sense
6,Gladiator
6,Patriot
6,Sixth Sense
7,Harry Potter1
7,Harry Potter2
8,Gladiator
8,Patriot
9,Gladiator
9,Patriot
9,Sixth Sense
10,Sixth Sense
10,L0TR
10,Galdiator
10,Green Mile
```

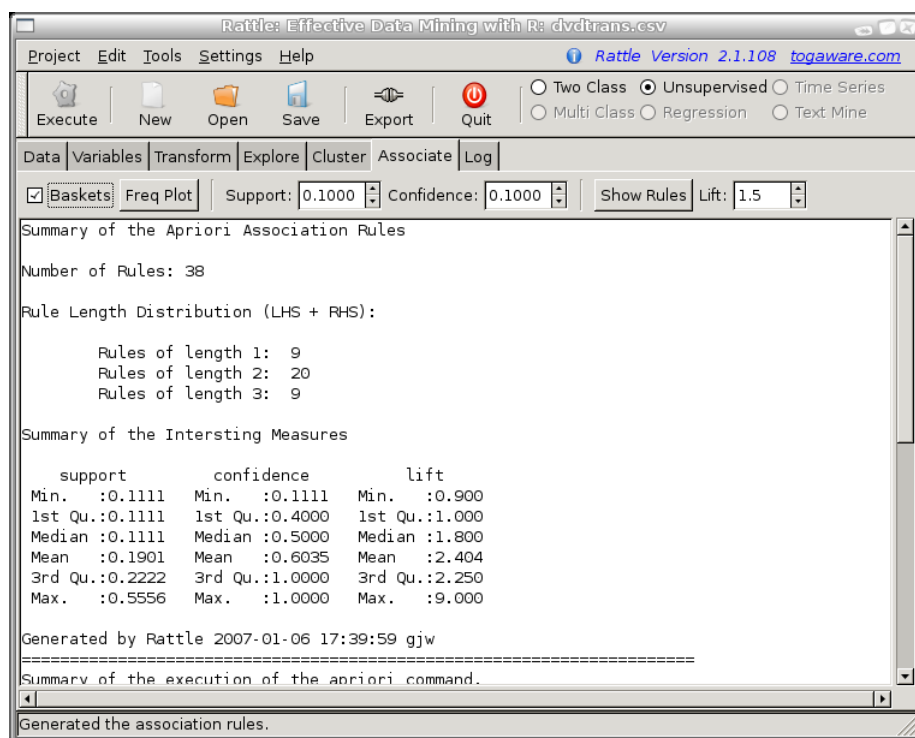
We load this data into **Rattle** and choose the appropriate variable roles. In this case it is quite simple: On the Associate tab (of the Unsupervised paradigm) ensure the Baskets check button is checked. Click the Execute button to identify the associations: Here we see a summary of the associations found. There were 38 association rules that met the criteria of having a minimum support of 0.1 and a minimum confidence of 0.1. Of these, 9 were of length 1 (i.e., a single item that has occurred frequently enough in the data), 20 were of length 2 and another 9 of length 3. Across the rules the support ranges from 0.11 up to 0.56. Confidence ranges from 0.11 up to 1.0, and lift from 0.9 up to 9.0.

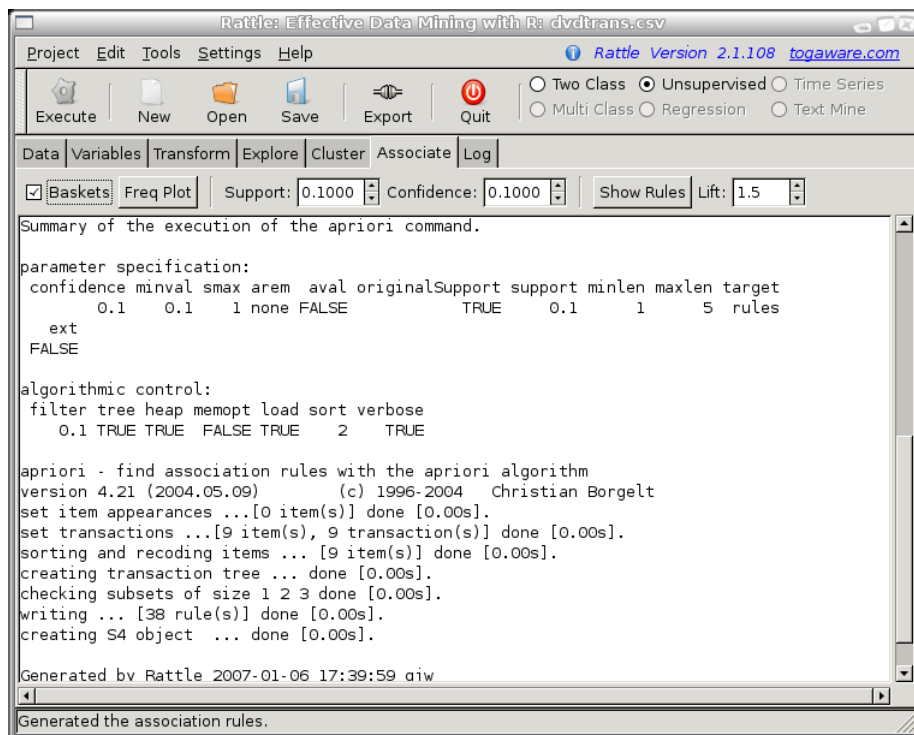
The lower part of the same textview contains information about the



running of the algorithm: We can see the parameter settings used, noting that Rattle only provides access to a smaller set of settings (support and confidence). The output includes timing information for the various phases of the algorithm. For such a small dataset, the times are of course essentially 0!

8.1.2 General Rules





Togaware Watermark For TeX Catalogue Survival

Chapter 9

Multi Class Models

Togaware Watermark For TeX Catalogue Survival

Togaware Watermark For TeX Catalogue Survival

Chapter 10

Regression Models

Togaware Watermark For TeX Catalogue Survival

Togaware Watermark For TeX Catalogue Survival

Chapter 11

Text Mining

Togaware Watermark For TeX Catalogue Survival

Togaware Watermark For TeX Catalogue Survival

Chapter 12

Evaluation and Deployment

Evaluating the performance of model building is important. We need to measure how any model we build will perform on previously unseen cases. A measure will also allow us to ascertain how well a model performs in comparison to other models we might choose to build, either using the same model builder, or a very different model builder. A common approach is to measure the error rate as the proportional number of cases that the model incorrectly (or equivalently, correctly) classifies. Common methods for presenting and estimating the empirical error rate include confusion matrices and cross-validation.

The various approaches to measuring performance include Lift, area under the ROC curve, the F-score, average precision, precision/recall, squared error, and risk charts.

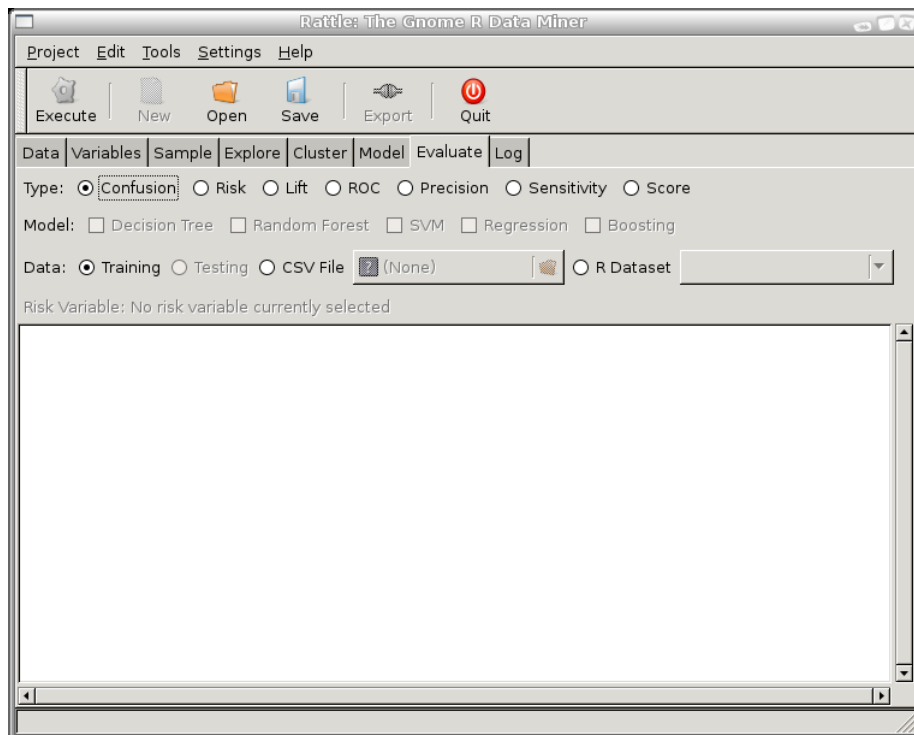
In this chapter we explore **Rattle**'s various tools for reporting the performance of a model and its various approaches to evaluating the output of data mining. We include the confusion matrix (using underneath the *table* function) for producing confusion matrices, **Rattle**'s new Risk Chart for effectively displaying model performance including a measure of the success of each case, and we explore the use of the *ROCR* package for the graphical presentation of numerous evaluations, including those common approaches included in **Rattle**. *Moving in to R* illustrates how to fine the presentations for your own needs.

This chapter also touches on issues around Deployment of our models,

and in particular Rattle's Scoring option, which allows us to load a new dataset and apply our model to that dataset, and to save the scores, together with the identity data, to a file for actioning.

12.1 The Evaluate Tab

The Evaluate Tab displays all the options available for evaluating the performance of our models, and for deploying the model over new datasets.



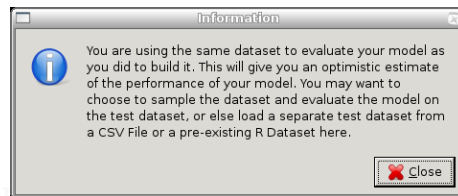
The range of different types of evaluations is presented as a series of radio buttons, allowing just a single evaluation type to be chosen at any time. Each type of evaluation is presented in the following sections of this chapter.

Below the row of evaluation types is a row of check buttons to choose the model types we wish to evaluate. The check buttons are only sensitive once a model has been built, and so on a fresh start of Rattle no model

check button can be checked. As models are built, they will become sensitive, and as we move from the Model tab to this Evaluate tab the most recently built model will be automatically checked. This corresponds to a common pattern of behaviour, in that often we will build and tune a model, then want to explore its performance by moving to this Evaluate tab. The check status of each of the other tabs does not otherwise change except by our clicking them.

To evaluate a model we need to identify a dataset on which to perform the evaluation.

The first option (but not the best option) is to evaluate our model on the training dataset. This is generally not a good idea, and the information dialogue shown here will be displayed each time we perform an evaluation on a training dataset. The output of any evaluation on the training dataset will also highlight this fact. The problem is that we have built our model on this training dataset, and it is often the case that the model will perform very well on that dataset! It should, because we've tried hard to make sure it does. But this does not give us a very good idea of how well the model will perform in general, on previously unseen data.



For a better guide to how well the model will perform in general, that is, on new and previously unseen data, we need to apply the model to such data and obtain an error rate. This error rate, and not the error rate from the training dataset, will then be a better estimate of how well the model will perform.

We discussed the concept of a training set in Section 3.3.1, presenting the Sample tab which provides a simple but effective mechanism for identifying a part of the dataset to be held separately from the training dataset, and to be used explicitly as the testing dataset. As indicated there, the default in Rattle is to use 70% of the dataset for training, and 30% for testing.

The final piece of information displayed in the common area of the Evaluate tab is the Risk Variable. The concept of the Risk Variable has been discussed in Section 3.3. It is used as a measure of how significant each

case is, with a typical example recording the dollar value of the fraud related to the case. The Risk Chart makes use of this variable if there is one, and it is included in the common area of the Evaluation tab for information purposes only.

12.2 Confusion Matrix

12.2.1 Measures

True positives (TPs) are those records which are correctly classified by a model as positive instances of the concept being modelled (e.g., the model identifies them as a case of fraud, and they indeed are a case of fraud). **False positives** (FPs) are classified as positive instances by the model, but in fact are known not to be. Similarly, **true negatives** (TNs) are those records correctly classified by the model as not being instances of the concept, and **false negatives** (FNs) are classified as not being instances, but are in fact known to be. These are the basic measures of the performance of a model. These basic measures are often presented in the form of a **confusion matrix**, produced using a **contingency table**.

12.2.2 Graphical Measures

ROC graphs, sensitivity/specificity curves, lift charts, and precision/recall plots are useful in illustrating specific pairs of performance measures for classifiers. The *ROCR* package creates 2D performance curves from any two of over 25 standard performance measures. Curves from different cross-validation or bootstrapping runs can be averaged by different methods, and standard deviations, standard errors or box plots can be used to visualize the variability across the runs. See **demo(ROCR)** and <http://rocr.bioinf.mpi-sb.mpg.de/> for examples.

12.2.3 Issues

Overfitting

Overfitting is more of a problem when training on smaller datasets.

Imbalanced Decisions

Model accuracy is not such an appropriate measure of performance when the data has a very imbalanced distribution of outcomes. For example, if positive cases account for just 1% of all cases, as might be the case in an insurance dataset recording cases of fraud, then the most accurate, but most useless, of models is one that predicts no fraud in all cases. It will be 99% accurate!

Togaware Watermark For TeX Catalogue Survival

12.3 Lift

Togaware Watermark For TeX Catalogue Survival

12.4 ROC Curves

Togaware Watermark For TeX Catalogue Survival

Area Under Curve

Togaware Watermark For TeX Catalogue Survival

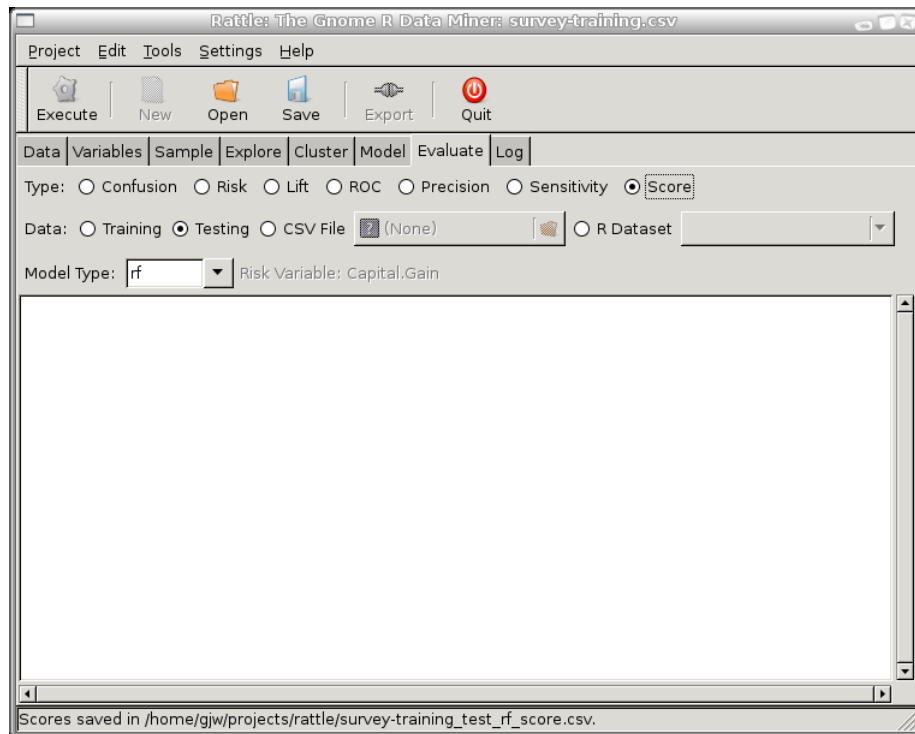
12.5 Precision versus Recall

Togaware Watermark For TeX Catalogue Survival

12.6 Sensitivity versus Specificity

Togaware Watermark For TeX Catalogue Survival

12.7 Score Option



Often you will want to apply a model to a dataset to generate the scores for each entity in the dataset. Such scores may be useful for further exploration in other tools, or for actual deployment.

The Score radio button allows you to score (i.e., to generate probabilities for each entry in) a dataset. The specific dataset which is scored is that which is identified with the Data option. In the above example, the model will be used to score the Testing dataset. You can score the actual Training dataset, a dataset loaded from a CSV data file, or from a dataset already loaded into R.

Rattle will generate a CSV file containing the “scores” for the dataset. Each line of the CSV file will consist of a comma separated list of all of the variables that have been identified as *Idents* in the Variables tab, followed by the score itself. This score will be a number between 0 and 1.

Note the status bar in the sample screenshot has identified that the score file has been saved to the suitably named file. The file name is derived from name of the dataset (perhaps a source data csv filename or the name of an R data frame), whether it is a test or training dataset, the type of model and the type of score.

The output looks like:

```
ID,predict
98953270,0.104
12161980,NA
96316627,0.014
54464140,0.346
57742269,0.648
19307037,0.07
61179245,0.004
36044473,0.338
19156946,0.33
```

12.8 Calibration Curves

Chapter 13

Moving into R

R is a statistical programming language together with an extensive collection of packages (of which **Rattle** is one) that provide a comprehensive toolkit for, amongst other things, data mining. Indeed, R is extensively deployed in bio-informatics, epidemiology, geophysics, agriculture and crop science, ecology, oceanography, fisheries, risk analysis, process engineering, pharmaceutical research, customer analytics, sociology, political science, psychology, and more.

One of the goals of **Rattle** is to ease a user's transition into using R directly, and thereby unleashing the full power of the language. R is a relatively simple language to learn, but has its own idiosyncrasies that emerge in any language that has such a long history, as R does.

In this chapter we discuss how we can access the internal data structures used by **Rattle**. This then allows us to smoothly switch between using **Rattle** and R in a single session. We significantly augment the functionality encapsulated in **Rattle** through this direct interaction with R, whilst not losing the ability to quickly explore the results in **Rattle**.

13.1 The Current Rattle State

Internally, **Rattle** uses the variable `crs` to store the current rattle state. We can have a look at the structure of this variable at any time using

the *str* function in the R Console.

<code>crs\$ident</code>	List of variable names treated as identifiers.
<code>crs\$input</code>	List of variable names for input to modelling.
<code>crs\$target</code>	The name of the variable used as target for modelling.

Togaware Watermark For TeX Catalogue Survival

13.2 Data

Our example of loading data into Rattle from a CSV file simply uses the R function *read.csv*.

13.3 Samples

Rattle uses a simple approach to generating a partitioning of our dataset into training and testing datasets with the *sample* function.

```
crs$sample <- sample(nrow(crs$dataset), floor(nrow(crs$dataset)*0.7))
```

The first argument to *sample* is the top of the range of integers you wish to choose from, and the second is the number to choose. In this example, corresponding to the *audit* dataset, 1400 (which is 70% of the 2000 entities in the whole dataset) random numbers between 1 and 2000 will be generated. This list of random numbers is saved in the corresponding Rattle variable, *crs\$sample* and used throughout Rattle for selecting or excluding these entities, depending on the task.

To use the chosen 1400 entities as a training dataset, we index our dataset with the corresponding Rattle variable:

```
crs$dataset[crs$sample,]
```

This then selects the 1400 rows from *crs\$dataset* and all columns.

Similarly, to use the other 600 entities as a testing dataset, we index our dataset using the same Rattle variable, but in the negative!

```
crs$dataset[-crs$sample,]
```

Each call to the *sample* function generates a different random selection. In Rattle, to ensure we get repeatable results, a specific seed is used each time, so that with the same seed, we obtain the same random selection, whilst also providing us with the opportunity to obtain different random selections. The *set.seed* function is called immediately prior to the *sample* call to specify the user chosen seed. The default seed used in Rattle is arbitrarily the number 123:

```
set.seed(123)
crs$sample <- sample(nrow(crs$dataset), floor(nrow(crs$dataset)*0.7))
```

In moving into R we might find the *sample.split* function of the *caTools* package handy. It will split a

vector into two subsets, two thirds in one and one third in the other, maintaining the relative ratio of the different categorical values represented in the vector. Rather than returning a list of indices, it works with a more efficient Boolean representation:

```
> library(caTools)
> mask <- sample.split(crs$dataset$Adjusted)

> head(mask)
[1] TRUE TRUE TRUE FALSE TRUE TRUE

> table(crs$dataset$Adjusted)
 0    1
1537 463

> table(crs$dataset$Adjusted[mask])
 0    1
1025 309

> table(crs$dataset$Adjusted[!mask])
 0    1
512 154
```

Perhaps it will be more convincing to list the proportions in each of the groups of the target variable (rounding these to just two digits):

```
> options(digits=2)

> table(crs$dataset$Adjusted)/
  length(crs$dataset$Adjusted)
 0    1
0.77 0.23

> table(crs$dataset$Adjusted[mask])/
  length(crs$dataset$Adjusted[mask])
 0    1
0.77 0.23

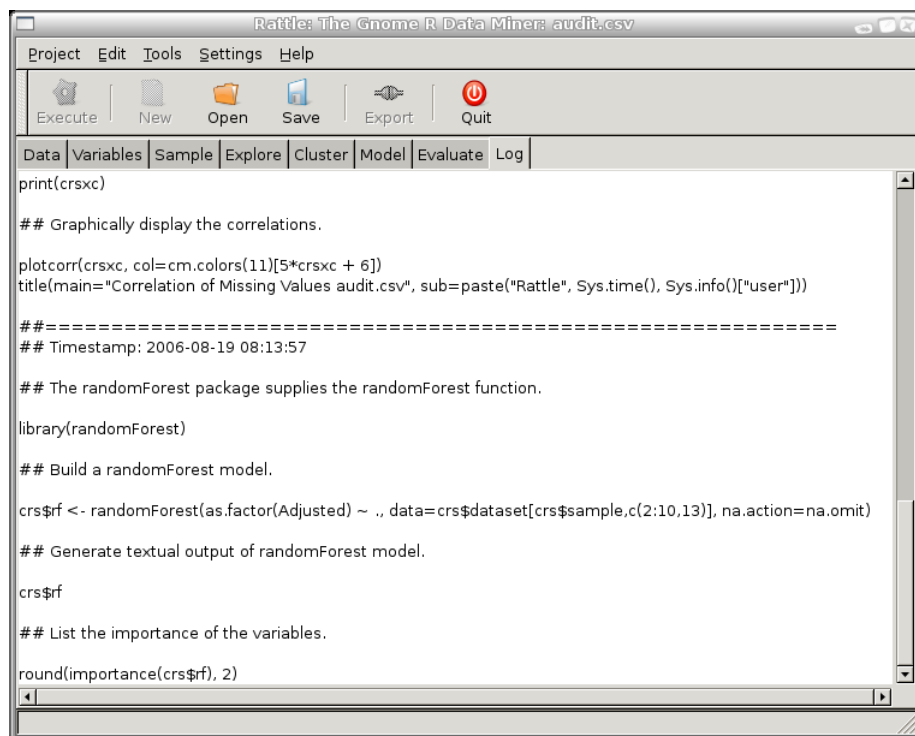
> table(crs$dataset$Adjusted[!mask])/
  length(crs$dataset$Adjusted[!mask])
 0    1
0.77 0.23
```

Thus, using this approach, both the training and the testing datasets will have the same distribution of the target variable.

13.4 Projects

We have illustrated in this chapter how to save Rattle's current state of affairs to a `.rattle` file. This file is in fact a standard R file, and is simply saving a single R object called `crs` (for Current Rattle State). The object has many slots, storing the dataset, the models, and various option settings.

13.5 The Rattle Log



All R commands that Rattle runs underneath are exposed through the text view of the Log tab. The intention is that the R commands be available for copying into the R console so that where Rattle only exposes a limited number of options, further options can be tuned via the R console.

The Log tab aims to be educational as much as possible. Informative comments are included to describe the steps involved.

Also, the whole log can be saved to a script file (with a R filename extension) and in principle, loaded into R to repeat the exact steps of the Rattle interactions. In general, you may want to review the steps and fine tune them to suit your purposes. After pasting the contents of the Log text view into a file, perhaps with a filename of `audit-rf-risk.R`, you can have the file execute as a script in R with:

```
> source("audit-rf-risk.R")
```

Internally, Rattle uses a variable called `crs` to store its current state, and you can modify this variable directly. Generally, changes you make will be reflected within Rattle and vice versa.

13.6 Further Tuning Models Catalogue Survival

One of the goals of Rattle is to keep things simple for the user. Consequently, not all options available for many of the functions provided by R are exposed through the Rattle user interface. This is not meant to be a limitation though, and Rattle is quite at ease working with modifications you make to the `crs` data structure within the R Console, at least to quite some extent.

Suppose for example that you wish to build an ada model using the `x` and `y` arguments rather than the formula argument. First, within Rattle, build the normal ada model and go to the Log tab to highlight and copy the command used:

```
crs$ada <- ada(Adjusted ~ .,
               data=crs$dataset[crs$sample,c(2:4,6:10,13)],
               control=rpart.control(maxdepth=30, cp=0.010000,
                                     minsplit=20, xval=10),
               iter=50)
```

Now past that into the R Console, but modify it appropriately:

```
crs$ada <- ada(crs$dataset[crs$sample,c(2:4,6:10)],
               crs$dataset[crs$sample,c(13)],
               control=rpart.control(maxdepth=30, cp=0.010000,
                                     minsplit=20, xval=10),
               iter=50)
```

You can now go back to the **Rattle** window's Evaluate tab and evaluate the performance of this new model. Indeed, you can, if you choose, save the models to different variables in the R Console, and selectively copy them into `crs$ada` and then evaluate them with **Rattle**. Of course, the alternative is to copy the R commands for the evaluation from the Log tab of **Rattle** and paste them into the R console and perform the evaluation programmatically.

Togaware Watermark For TeX Catalogue Survival

Togaware Watermark For TeX Catalogue Survival

Chapter 14

Troubleshooting

Rattle is open source, freely available, software, and benefits from user feedback and user contributions. It is also under constant development and improvement. Whilst every effort is made to ensure Rattle is error and bug free, there is always the possibility that we will find a new bug. This is then the opportunity for us to review the code and to improve the code. At the very least though, if you find a problem with Rattle, contact the author to report the problem. At best, feel free to hunt down the source of the problem in the R code and provide the solution to the author, for all to benefit.

In this chapter we record some known issues that you may come across in using Rattle. These aren't bugs as such but rather known issues that a data miner is likely to come across in building models.

14.1 A factor has new levels

This occurs when the training dataset does not contain examples of all of the levels of particular factor and the testing set contains examples of these other levels.

Togaware Watermark For TeX Catalogue Survival

Bibliography

- Aggarwal, C. C. and Yu, P. S. (2001), Outlier detection for high dimensional data, in *Proceedings of the 27th ACM SIGMOD International Conference on Management of Data (SIGMOD01)*, pp. 37–46. [360](#)
- Agrawal, R. and Srikant, R. (1994), Fast algorithms for mining association rules in large databases, in J. B. Bocca, M. Jarke and C. Zaniolo, eds, *Proceedings of the 20th International Conference on Very Large Databases (VLDB94)*, Morgan Kaufmann, pp. 487–499. <http://citeseer.ist.psu.edu/agrawal94fast.html>. [403](#)
- Barnett, V. and Lewis, T. (1994), *Outliers in Statistical Data*, John Wiley. [359](#)
- Bauer, E. and Kohavi, R. (1999), ‘An empirical comparison of voting classification algorithms: Bagging, boosting, and variants’, *Machine Learning* **36**(1-2), 105–139. <http://citeseer.ist.psu.edu/bauer99empirical.html>. [79](#), [412](#)
- Beyer, K. S., Goldstein, J., Ramakrishnan, R. and Shaft, U. (1999), When is “nearest neighbor” meaningful?, in *Proceedings of the 7th International Conference on Database Theory (ICDT99)*, Jerusalem, Israel, pp. 217–235. <http://citeseer.ist.psu.edu/beyer99when.html>. [360](#)
- Bhandari, I., Colet, E., Parker, J., Pines, Z., Pratap, R. and Ramanujam, K. (1997), ‘Advance scout: data mining and knowledge discovery in nba data’, *Data Mining and Knowledge Discovery* **1**(1), 121–125. [402](#)
- Blake, C. and Merz, C. (1998), ‘UCI repository of machine learning databases’. <http://www.ics.uci.edu/~mlearn/MLRepository.html>. [194](#)

- Breiman, L. (1996), ‘Bagging predictors’, *Machine Learning* **24**(2), 123–140. <http://citeseer.ist.psu.edu/breiman96bagging.html>. 406
- Breiman, L. (2001), ‘Random forests’, *Machine Learning* **45**(1), 5–32. 454
- Breunig, M. M., Kriegel, H., Ng, R. and Sander, J. (1999), OPTICS-OF: Identifying local outliers, in *Proceedings of the XXXXth Conference on Principles of Data Mining and Knowledge Discovery (PKDD99)*, Springer-Verlag, pp. 262–270. 359
- Breunig, M. M., Kriegel, H., Ng, R. and Sander, J. (2000), LOF: Identifying density based local outliers, in *Proceedings of the 26th ACM SIGMOD International Conference on Management of Data (SIGMOD00) Proceedings of the 26th ACM SIGMOD International Conference on Management of Data (SIGMOD00) (2000)*. 359
- Caruana, R. and Niculescu-Mizil, A. (2006), An empirical comparison of supervised learning algorithms, in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA. 85
- Cendrowska, J. (1987), ‘An algorithm for inducing modular rules’, *International Journal of Man-Machine Studies* **27**(4), 349–370. 410
- Cleveland, W. S. (1993), *Visualizing Data*, Hobart Press, Summit, New Jersey. 48, 233
- Culp, M., Johnson, K. and Michailidis, G. (2006), ‘ada: An r package for stochastic boosting’, *Journal of Statistical Software* **17**(2). <http://www.jstatsoft.org/v17/i02/v17i02.pdf>. 101
- Cypher, A., ed. (1993), *Watch What I Do: Programming by Demonstration*, The MIT Press, Cambridge, Massachusetts. <http://www.acypher.com/wwid/WWIDToC.html>. 139
- Dalgaard, P. (2002), *Introductory Statistics with R*, Statistics and Computing, Springer, New York. xxxiv
- Freund, Y. and Schapire, R. E. (1995), A decision-theoretic generalization of on-line learning and an application to boosting, in *Proceedings of the 2nd European Conference on Computational Learning Theory (Eurocolt95)*, Barcelona, Spain, pp. 23–37. <http://citeseer.ist.psu.edu/freund95decisiontheoretic.html>. 80, 412, 414, 421

- Friedman, J. H. (2001), ‘Greedy function approximation: A gradient boosting machine’, *Annals of Statistics* **29**(5), 1189–1232. <http://citeseer.ist.psu.edu/46840.html>. 419
- Friedman, J. H. (2002), ‘Stochastic gradient boosting’, *Computational Statistics and Data Analysis* **38**(4), 367–378. <http://citeseer.ist.psu.edu/friedman99stochastic.html>. 419
- Hahsler, M., Grün, B. and Hornik, K. (2005), *A Computational Environment for Mining Association Rules and Frequent Item Sets*, R Package, Version 0.2-1. 395, 397
- Hastie, T., Tibshirani, R. and Friedman, J. (2001), *The elements of statistical learning: Data mining, inference, and prediction*, Springer Series in Statistics, Springer-Verlag, New York. xxxiv, 21, 79, 411, 414, 421
- Hawkins, D. (1980), *Identification of Outliers*, Chapman and Hall, London. 358, 359
- Jin, W., Tung, A. K. H. and Han, J. (2001), Mining top-n local outliers in large databases, in *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining (KDD01)*. 359
- King, R. D., Feng, C. and Sutherland, A. (1995), ‘Statlog: Comparison of classification algorithms on large real-world problems’, *Applied Artificial Intelligence* **9**(3), 289–333. 85
- Knorr, E. and Ng, R. (1998), Algorithms for mining distance based outliers in large databases, in *Proceedings of the 24th International Conference on Very Large Databases (VLDB98)*, pp. 392–403. 359
- Knorr, E. and Ng, R. (1999), Finding intensional knowledge of distance-based outliers, in *Proceedings of the 25th International Conference on Very Large Databases (VLDB99) Proceedings of the 25th International Conference on Very Large Databases (VLDB99) (1999)*, pp. 211–222. 359
- Kohavi, R. (1996), Scaling up the accuracy of naive-Bayes classifiers: A decision tree hybrid, in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD96)*, Portland, OR, pp. 202–207. <http://citeseer.ist.psu.edu/kohavi96scaling.html>. 410

- Lin, W., Orgun, M. A. and Williams, G. J. (2000), Temporal data mining using multilevel-local polynomial models, in *Proceedings of the 2nd International Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2000)*, Hong Kong, Vol. 1983 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 180–186. 360
- Lin, W., Orgun, M. A. and Williams, G. J. (2001), Temporal data mining using hidden markov-local polynomial models, in D. W.-L. Cheung, G. J. Williams and Q. Li, eds, *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD01)*, Hong Kong, Vol. 2035 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 324–335. 360
- Mingers, J. (1989), ‘An empirical comparison of selection measures for decision-tree induction’, *Machne Learning* **3**(4), 319–342. 370
- Proceedings of the 25th International Conference on Very Large Databases (VLDB99)* (1999). 529, 531 X Catalogue Survival
- Proceedings of the 26th ACM SIGMOD International Conference on Management of Data (SIGMOD00)* (2000), ACM Press. 528, 530
- Provost, F. J., Jensen, D. and Oates, T. (1999), Efficient progressive sampling, in *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD99)*, San Diego, CA, ACM Press, pp. 23–32. <http://citeseer.ist.psu.edu/provost99efficient.html>. 487
- Quinlan, J. R. (1993), *C4.5: Programs for machine learning*, Morgan Kaufmann. 487
- R D (2005), *R Data Import/Export*, version 2.1.1 edn. 192
- Ramaswamy, S., Rastogi, R. and Kyuseok, S. (2000), Efficient algorithms for mining outliers from large data sets, in *Proceedings of the 26th ACM SIGMOD International Conference on Management of Data (SIGMOD00)* *Proceedings of the 26th ACM SIGMOD International Conference on Management of Data (SIGMOD00)* (2000), pp. 427–438. 359
- Schafer, J. L. (1997), *Analysis of Incomplete Multivariate Data*, Chapman and Hall, London. 346

- Schapire, R. E., Freund, Y., Bartlett, P. and Lee, W. S. (1997), Boosting the margin: a new explanation for the effectiveness of voting methods, in *Proceedings of the 14th International Conference on Machine Learning (ICML97)*, Morgan Kaufmann, pp. 322–330. <http://citeseer.ist.psu.edu/schapire97boosting.html>. 79, 412
- Soares, C., Brazdil, P. B. and Kuba, P. (2004), ‘Meta-learning method to select the kernel width in support vector regression’, *Machine Learning* **54**(3), 195–209. 457, 460
- Tukey, J. W. (1977), *Exploratory data analysis*, Addison-Wesley. 51, 271, 332
- Venables, W. N. and Ripley, B. D. (2002), *Modern Applied Statistics with S*, Statistics and Computing, 4th edn, Springer, New York. xxxiv
- Viveros, M. S., Nearhos, J. P. and Rothman, M. J. (1999), Applying data mining techniques to a health insurance information system., in *Proceedings of the 25th International Conference on Very Large Databases (VLDB99) Proceedings of the 25th International Conference on Very Large Databases (VLDB99)* (1999), pp. 286–294. <http://www.informatik.uni-trier.de/~ley/vldb/ViverosNR96/Article.PS>. 402
- Williams, G. J. (1987), ‘Some experiments in decision tree induction.’, *Australian Computer Journal* **19**(2), 84–91. 519
- Williams, G. J. (1988), Combining decision trees: Initial results from the MIL algorithm, in J. S. Gero and R. B. Stanton, eds, *Artificial Intelligence Developments and Applications*, Elsevier Science Publishers B.V. (North-Holland), pp. 273–289. 519
- Williams, G. J. (1991), Inducing and combining decision structures for expert systems, PhD thesis, Australian National University. <http://togaware.redirectme.net/papers/gjwthesis.pdf>. 519
- Yamanishi, K., ichi Takeuchi, J., Williams, G. J. and Milne, P. (2000), On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms, in *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD00)*, pp. 320–324. <http://citeseer.ist.psu.edu/446936.html>. 359

Togaware Watermark For TeX Catalogue Survival

Index

- .Machine, 165
- .Platform, 163, 164
- R Console, 12
- R, 7
- Access
 - Import data into R, 203
- ada (R package), 96, 101
- AdaBoost, 79, 411–422
- adaboost (R function), 416
- Adjusted, 63, 88
- Adjustment, 88
- Advance Scout, 402
- Age, 62
- aggregate (R function), 254
- amap (R package), 425, 430, 437, 439
- AMD64, 167
- analysis of variance, 289
- analysis of variance, 289
- ANOVA, 289, *see* analysis of variance
- apply, *see* lapply, mapply, sapply
- apply (R function), 346, 352
- Apriori, 403
- apriori, 387
- apriori (R function), 391, 395, 398, 399
- array (R function), 180
- arrows (R function), 214
- Artificial neural networks, *see* Neural networks
- arules (R package), 388, 391, 392, 395
- as (R function), 398
- as.Date (R function), 185
- as.integer (R function), 256
- as.logical (R function), 349
- as.matrix (R function), 168
- association analysis
 - Apriori, 387–403
- associations, 518
- at, 282
- attach (R function), 184, 191, 211, 291
- attr (R function), 162
- audit (Dataset), 14, 27, 88, 98, 129
- available.packages (R function), 156
- Bagging, 405
- bagging, 405–406
- barchart, 300
- barchart (R function), 291, 292
- barplot (R function), 245, 321
- bayesian analysis
 - Bayes theorem, 408
- bbox (R function), 206
- believeNRows, 201
- Benford's Law, 56
- binning (R function), 355
- bitmap (R function), 226
- bmp (R function), 226

- boost (R package), [416](#), [419](#)
- Boosting, [79–80](#), [411–422](#), [517](#)
- bootstrap aggregating, [405](#)
- Bootstrapping, [423–424](#)
- Borgelt, [467–468](#)
- box and whisker plot, *see* box plot
- boxplot, [51](#), [271](#), [332](#)
- boxplot (R function), [211](#), [271–274](#), [332–334](#), [346](#)
- breaks, [247](#)
- bxp (R function), [274](#), [275](#)
- c (R function), [179](#), [181](#), [372](#)
- C4.5, [485–487](#)
- capabilities (R function), [165](#)
- cast (R function), [253](#)
- caTools (R package), [130](#), [223](#), [342](#), [416](#)
- chron (R package), [313](#)
- class (R function), [347](#)
- Classification
 - C4.5, [485–487](#)
 - Conditional trees, [433–435](#)
 - Decision trees, [78–79](#), [361–370](#), [485–487](#)
 - K-nearest neighbour, [445–446](#)
 - Kernel methods, [457–458](#)
 - Neural networks, [451–452](#)
 - Support vector machine (SVM), [457–460](#)
- classification
 - Naïve Bayes, [407](#), [410](#)
- classwt, [453](#)
- Clementine, [489](#)
- clipboard, [160](#), [193](#), [203](#), [204](#), [350](#)
- closure, [175](#)
- Clustering
 - Hierarchical, [430–431](#), [437](#)
 - K-means, [425–430](#), [439–443](#)
- cm.colors (R function), [217](#), [260](#), [324](#)
- col, [215](#), [217](#), [236](#)
- colnames (R function), [181](#), [183](#), [235](#), [316](#), [349](#)
- color, [246](#)
- colour (R function), [217](#)
- colSums (R function), [352](#)
- comment, [151](#)
- complete.cases (R function), [346](#)
- complex (R function), [223](#)
- complex numbers, [176](#)
- compress, [190](#)
- Conditional trees, [433–435](#)
- confidence, [389](#)
- confusion matrix, [118](#), [373](#)
- confusion matrix, [118](#), [373](#)
- contingency table, [118](#), [373](#)
- contingency table, [118](#), [373](#)
- continue, [166](#)
- cor (R function), [260](#), [324](#)
- correlation, [260](#), [324](#)
- cost, [368](#)
- ctree (R function), [434](#)
- cut (R function), [397](#)
- Data, [175](#)
- data, [356](#)
- data import
 - csv, [202](#)
 - txt, [26](#)
- data sources, *see* data import
- data (R function), [193](#)
- data cleaning, [342–349](#)
- data frame, [183–184](#)
- data import
 - Access, [22](#)
 - arff, [22](#), [27–28](#)
 - csv, [22](#), [24–26](#)

- DB2, [22](#)
- Excel, [22](#), [202](#)
- missing values, [26](#)
- MySQL, [22](#)
- ODBC, [22](#), [29–30](#), [202](#)
- Oracle, [22](#)
- SQL Server, [22](#)
- Teradata, [22](#)
- txt, [22](#)
- data linking, [350–351](#)
- data transformation, [352–355](#)
 - aggregation, [352](#)
 - Sum of columns, [352](#)
- data types, [175](#)
 - Data frame, [183–184](#)
 - Date, [185](#)
 - Matrix, [181–183](#)
 - String, [177–178](#)
 - Vector, [179–180](#)
- Datasets
 - audit, [14](#), [27](#), [88](#), [98](#), [129](#)
 - iris, [145](#), [190](#), [211](#)
 - survey, [197](#), [349](#), [373](#)
 - wine, [195](#), [234](#), [236](#), [239](#), [245](#), [253](#), [254](#), [256](#), [260](#), [315](#), [321](#), [324](#), [417](#)
- date, [185](#)
- dd.load (R function), [315](#)
- Debian, [143](#)
- Decision trees, [78–79](#), [361–370](#)
- density estimate, [53](#)
- DescribeDisplay (R package), [315](#)
- Design (R package), [449](#)
- detach (R function), [155](#), [211](#)
- dev.copy (R function), [229](#)
- dev.cur (R function), [226](#)
- dev.list (R function), [226](#)
- dev.next (R function), [226](#)
- dev.off (R function), [225](#)
- dev.prev (R function), [226](#)
- dev.set (R function), [226](#)
- digits, [130](#), [166](#)
- dim (R function), [235](#), [316](#), [349](#)
- distribution, [419](#)
- distributions
 - normal, [54](#)
- divide by zero, [187](#)
- do.call (R function), [183](#)
- download.file (R function), [194](#)
- download.packages (R function), [156](#)
- dprep (R package), [356](#)
- duplicated (R function), [344](#)
- e1071 (R package), [456](#), [460](#)
- EDA, *see* Exploratory data analysis
- exploratory data analysis
 - analysis
- Eddelbuettel, Dirk, [143](#)
- edit (R function), [153](#), [182](#)
- ellipse (R package), [260](#), [324](#)
- Employment, [62](#)
- Enterprise Miner, [501–504](#)
- Equbits Foresight, [491](#)
- evaluation
 - risk chart, [88](#)
- example (R function), [153](#)
- Excel, *see* data import
- exploratory data analysis, [39](#), [233](#)
- false negative, [118](#), [373](#)
- false positive, [118](#), [373](#)
- feature selection, [355](#)
- fields (R package), [223](#)
- fig (R function), [226](#)
- file (R function), [203](#)
- file.choose (R function), [196](#)
- finco (R function), [356](#)
- fix (R function), [182](#)

- Flavanoids, 256
- floor (R function), 129
- for (R function), 349
- format (R function), 191, 352
- format.df (R function), 191
- formatC (R function), 191
- Fujitsu, 493–495
- functional, 151
- functional language, 151
- gbm (R function), 419
- gbm (R package), 416, 419
- gc (R function), 169
- gcinfo (R function), 169
- get (R function), 162
- getOption (R function), 165
- GGobi, 61, 62
- ggplot (R package), 276
- GhostMiner, 493–495
- GNU/Linux, 143
- gplots (R package), 159, 288
- graphics
 - barchart, 300
- graphics.off (R function), 225
- gray (R function), 217
- grep (R function), 177
- gsub (R function), 178
- head (R function), 188, 235, 316
- Health Insurance Commission, 402
- help, 152, 158
- help (R function), 152, 159
- help.search (R function), 153
- help.start (R function), 152
- Hierarchical clustering, 430–431, 437
- hist (R function), 211, 247
- histogram, 53
- histogram (R function), 246
- Hmisc (R package), 191
- holdout method, 375
- horizontal, 272, 333
- htmlhelp, 152
- hyperedges, 394
- IBM
 - Advance Scout, 402
- if (R function), 179
- image (R function), 223
- InductionEngine, 497
- inspect (R function), 395
- install
 - GTK+, 8
 - R, 8
 - R packages, 11
 - Rattle, 11
 - RGtk2, 10
- install.packages (R function), 143, 156
- installed.packages (R function), 156
- interpreted language, 151
- interquartile range, 51, 271, 332
- invisible (R function), 160, 182
- iris (Dataset), 145, 190, 211
- is.factor (R function), 349
- is.integer (R function), 349
- is.logical (R function), 349
- is.na (R function), 346
- is.numeric (R function), 348, 349
- itemsets, 394
- join, *see* merge
- jpeg (R function), 226
- JPG, 209
- K-means, 425–430, 439–443
- K-Nearest Neighbour, 445
- K-nearest neighbour, 445–446
- Kernel Methods, 457, 458
- Kernel methods, 457

- kernlab (R package), 98, 456–458, 460
- kurtosis, 44
- lapply (R function), 256, 347, 349
- latex (R function), 191
- lattice (R package), 230, 246, 291, 299
- layout (R function), 281
- legend (R function), 215, 236
- length (R function), 130
- levels (R function), 186
- library (R function), 152, 155, 158
- load (R function), 190
- locator (R function), 230
- log (R function), 151
- Logistic regression, 449–450
- LogitBoost (R function), 416
- logitboost (R function), 416
- loss, 368
- lty, 215, 236
- mapply (R function), 161
- maptree (R package), 370
- matplot (R function), 236
- matrix, 181–183
- matrix (R function), 181, 352
- matrix scatterplot, 259
- max (R function), 349
- maxdepth, 417
- mean, 51, 268, 269, 328, 330
- mean (R function), 162, 253, 269, 330
- median, 51, 268, 271, 328, 332
- median (R function), 397
- merge (R function), 350
- Meta algorithms
 - AdaBoost, 411–422
 - Boosting, 79–80, 411–422
 - Bootstrapping, 423–424
- meta algorithms
 - bagging, 405–406
- methods (R function), 153
- mfrow, 274, 282
- min (R function), 349
- mode (R function), 176
- modelling
 - supervised, 17
 - unsupervised, 17
- mvpart (R function), 362
- mvpart (R package), 362, 370
- na.omit (R function), 346
- naïve Bayes classifier, 407–410
- nchar (R function), 177, 191
- ncol (R function), 235, 316, 349
- Neural networks, 451–452
- new, 228
- normal distribution, 54
- nrow (R function), 129, 145, 190, 235, 316, 352
- nsl (R function), 167
- object.size (R function), 168
- ODBC, *see* data import
- odbcClose (R function), 202, 203
- odbcConnect (R function), 200
- odbcConnectAccess (R function), 203
- odbcConnectExcel (R function), 202
- ODM, 499–500
- ODMiner, 499–500
- OLAP, 518
- on.exit (R function), 166
- options (R function), 130, 156, 165, 166
- Oracle, 499–500
- order (R function), 189
- ordered (R function), 186, 397

- outlier analysis, 358–360
- packageStatus (R function), 156, 157
- palette (R function), 217, 256
- par (R function), 229, 230, 274
- parms, 368
- party (R package), 434
- paste (R function), 177
- pch, 215, 256
- PDF, 209
- pdf (R function), 226, 229
- percentile, 51, 271, 332
- Phenols, 256
- pie (R function), 211, 239
- pie chart, 239
- pinktoe (R package), 370
- pivot table
 - reshape, 253
- plot (R function), 144, 153, 204, 211, 212, 228, 256, 259, 363
- plot.rpart (R function), 153
- plotcorr (R function), 260, 324
- plotmeans (R function), 288
- plotNetwork (R function), 284
- plots
 - matrix scatterplot, 259
 - Scatterplot, 211
 - scatterplot, 256, 259
- pmatch (R function), 177
- PNG, 209
- png (R function), 226
- PostScript, 209
- postscript (R function), 226, 229
- predict (R function), 366, 372, 373
- PredictionWorks, 497
- printep (R function), 363, 371
- prior, 368
- proc.time (R function), 166
- prompt, 166
- prompt (R function), 207
- q (R function), 145
- qplot (R function), 276
- Quantian live CD, 142
- quantile (R function), 274
- quartile, 51, 271, 332
- quartz (R function), 225
- R, 471–472
- R functions, 162
 - adaboost, 416
 - aggregate, 254
 - apply, 346, 352
 - apriori, 391, 395, 398, 399
 - array, 180
 - arrows, 214
 - as, 398
 - as.Date, 185
 - as.integer, 256
 - as.logical, 349
 - as.matrix, 168
 - attach, 184, 191, 211, 291
 - attr, 162
 - available.packages, 156
 - barchart, 291, 292
 - barplot, 245, 321
 - bbox, 206
 - binning, 355
 - bitmap, 226
 - bmp, 226
 - boxplot, 211, 271–274, 332–334, 346
 - bxp, 274, 275
 - c, 179, 181, 372
 - capabilities, 165
 - cast, 253
 - class, 347
 - cm.colors, 217, 260, 324

- colnames, 181, 183, 235, 316, 349
- colour, 217
- colSums, 352
- complete.cases, 346
- complex, 223
- cor, 260, 324
- ctree, 434
- cut, 397
- data, 193
- dd_load, 315
- detach, 155, 211
- dev.copy, 229
- dev.cur, 226
- dev.list, 226
- dev.next, 226
- dev.off, 225
- dev.prev, 226
- dev.set, 226
- dim, 235, 316, 349
- do.call, 183
- download.file, 194
- download.packages, 156
- duplicated, 344
- edit, 153, 182
- example, 153
- fig, 226
- file, 203
- file.choose, 196
- finco, 356
- fix, 182
- floor, 129
- for, 349
- format, 191, 352
- format.df, 191
- formatC, 191
- gbm, 419
- gc, 169
- gcinfo, 169
- get, 162
- getOption, 165
- graphics.off, 225
- gray, 217
- grep, 177
- gsub, 178
- head, 188, 235, 316
- help, 152, 159
- help.search, 153
- help.start, 152
- hist, 211, 247
- histogram, 246
- if, 179
- image, 223
- inspect, 395
- install.packages, 143, 156
- installed.packages, 156
- invisible, 160, 182
- is.factor, 349
- is.integer, 349
- is.logical, 349
- is.na, 346
- is.numeric, 348, 349
- jpeg, 226
- lapply, 256, 347, 349
- latex, 191
- layout, 281
- legend, 215, 236
- length, 130
- levels, 186
- library, 152, 155, 158
- load, 190
- locator, 230
- log, 151
- LogitBoost, 416
- logitboost, 416
- mapply, 161
- matplot, 236
- matrix, 181, 352

- max, 349
- mean, 162, 253, 269, 330
- median, 397
- merge, 350
- methods, 153
- min, 349
- mode, 176
- mvpart, 362
- na.omit, 346
- nchar, 177, 191
- ncol, 235, 316, 349
- nrow, 129, 145, 190, 235, 316, 352
- nsl, 167
- object.size, 168
- odbcClose, 202, 203
- odbcConnect, 200
- odbcConnectAccess, 203
- odbcConnectExcel, 202
- on.exit, 166
- options, 130, 156, 165, 166
- order, 189
- ordered, 186, 397
- packageStatus, 156, 157
- palette, 217, 256
- par, 229, 230, 274
- paste, 177
- pdf, 226, 229
- pie, 211, 239
- plot, 144, 153, 204, 211, 212, 228, 256, 259, 363
- plot.rpart, 153
- plotcorr, 260, 324
- plotmeans, 288
- plotNetwork, 284
- pmatch, 177
- png, 226
- postscript, 226, 229
- predict, 366, 372, 373
- printcp, 363, 371
- proc.time, 166
- prompt, 207
- q, 145
- qplot, 276
- quantile, 274
- quartz, 225
- rainbow, 338
- randomForest, 346
- rbind, 182, 183
- read.arff, 28
- read.csv, 129, 185, 194, 197
- read.transactions, 392
- read.xls, 202
- readShapePoly, 204
- rect, 220
- relief, 356
- remove.packages, 156
- rep, 352
- require, 155
- rescaler, 354
- return, 160
- rev, 189, 349
- rnorm, 304, 334
- rownames, 181, 235, 316
- rpart, 168, 345, 362, 368, 373, 413, 416, 417
- rpart.control, 417
- Rprof, 166
- RSiteSearch, 153
- runif, 179
- sample, 35, 129, 190, 341
- sample.split, 130, 342
- sapply, 270, 348
- save, 190, 195, 203
- scale, 352
- scan, 193, 204
- search, 155
- seq, 180, 349

- sessionInfo, 164
- set.seed, 35
- show.settings, 230
- sprintf, 191
- sqlFetch, 200, 202, 203
- sqlQuery, 202, 203
- sqlTables, 200, 202, 203
- stopifnot, 159
- str, 128, 152, 235, 317
- strftime, 185
- strsplit, 177
- strwidth, 220
- sub, 177, 178
- subset, 184, 282
- substr, 177
- sum, 352
- summary, 51, 157, 236, 268, 269, 271, 317, 329, 330, 332, 343, 393, 398, 399
- sunflower, 286
- svm, 456, 460
- Sys.info, 164
- Sys.sleep, 166
- system, 163
- system.time, 166, 169, 352
- t, 182
- table, 115, 130, 371, 373
- tail, 188
- text, 220, 245, 321
- tim.colors, 223
- traceback, 170
- trellis.par, 230
- trellis.par.get, 230
- trellis.par.set, 291
- typeof, 175
- unique, 189
- unlist, 256
- unstack, 184
- update.packages, 156
- UseMethod, 162
- vector, 179
- vignette, 152, 153, 395
- which, 187, 418
- win.metafile, 226, 228
- window, 19
- windows, 225
- with, 187, 212
- write.gif, 223
- write.table, 194, 195
- x11, 19, 225
- R options
 - at, 282
 - believeNRows, 201
 - breaks, 247
 - classwt, 453
 - col, 215, 217, 236
 - color, 246
 - compress, 190
 - continue, 166
 - cost, 368
 - digits, 130, 166
 - distribution, 419
 - help, 152, 158
 - horizontal, 272, 333
 - htmlhelp, 152
 - hyperedges, 394
 - itemsets, 394
 - loss, 368
 - lty, 215, 236
 - maxdepth, 417
 - mfrow, 274, 282
 - new, 228
 - parms, 368
 - pch, 215, 256
 - prior, 368
 - prompt, 166
 - rules, 394
 - sampsize, 453, 454

- scipen, 231
- strata, 453
- type, 373
- usr, 210, 229, 282
- weight, 413
- weights, 368, 369
- where, 365
- width, 166
- xlim, 204
- xpd, 210, 230, 245, 321
- y, 245, 321
- ylim, 204
- R packages
 - ada, 96, 101
 - amap, 425, 430, 437, 439
 - arules, 388, 391, 392, 395
 - boost, 416, 419
 - caTools, 130, 223, 342, 416
 - chron, 313
 - DescribeDisplay, 315
 - Design, 449
 - dprep, 356
 - e1071, 456, 460
 - ellipse, 260, 324
 - fields, 223
 - gbm, 416, 419
 - ggplot, 276
 - gplots, 159, 288
 - Hmisc, 191
 - kernlab, 98, 456–458, 460
 - lattice, 230, 246, 291, 299
 - maptree, 370
 - mvpart, 362, 370
 - party, 434
 - pinktoe, 370
 - randomForrest, 453
 - rattle, 148, 368
 - Rcmdr, 148, 150
 - reshape, 253, 354
 - rggobi, 9
 - ROCR, 115, 118, 371, 376
 - RODBC, 167, 200, 201
 - rpart, 370, 416
 - sudoku, 170
 - survey, 173
 - tree, 370
 - ttdda, 461
 - xlsReadWrite, 202
 - zoo, 238, 313
- R variables
 - .Machine, 165
 - .Platform, 163, 164
 - version, 164
- rainbow (R function), 338
- random forests, 80
- randomForest (R function), 346
- randomForrest (R package), 453
- Rattle, 473–474
 - install, 11
 - Options
 - GGobi, 61, 62
 - start up, 12
- rattle (R package), 148, 368
- rbind (R function), 182, 183
- Rcmdr (R package), 148, 150
- read.arff (R function), 28
- read.csv (R function), 129, 185, 194, 197
- read.transactions (R function), 392
- read.xls (R function), 202
- readShapePoly (R function), 204
- recall, 374
- rect (R function), 220
- Regression
 - Logistic regression, 449–450
 - Neural networks, 451–452
 - Support vector machine (SVM), 457–460

- relief (R function), 356
- remove.packages (R function), 156
- rep (R function), 352
- require (R function), 155
- rescaler (R function), 354
- reshape (R package), 253, 354
- return (R function), 160
- rev (R function), 189, 349
- rggobi (R package), 9
- Ridgeway, Greg, 419
- risk analysis, 89
- rnorm (R function), 304, 334
- ROCR (R package), 115, 118, 371, 376
- RODBC (R package), 167, 200, 201
- rownames (R function), 181, 235, 316
- rpart (R function), 168, 345, 362, 368, 373, 413, 416, 417
- rpart (R package), 370, 416
- rpart.control (R function), 417
- Rprof (R function), 166
- RSiteSearch (R function), 153
- rug, 53
- rules, 394
- runif (R function), 179
- Salford Systems, 511
- sample (R function), 35, 129, 190, 341
- sample.split (R function), 130, 342
- sampsize, 453, 454
- sapply (R function), 270, 348
- SAS, 501–504
- save (R function), 190, 195, 203
- scale (R function), 352
- scan (R function), 193, 204
- scatterplot, 211, 256, 259
- scipen, 231
- script file, 140, 144
- search (R function), 155
- selection, *see* clipboard
- seq (R function), 180, 349
- sessionInfo (R function), 164
- set.seed (R function), 35
- shapefiles, 204
- show.settings (R function), 230
- skewness, 46
- sprintf (R function), 191
- SPSS, 489
- sqlFetch (R function), 200, 202, 203
- sqlQuery (R function), 202, 203
- sqlTables (R function), 200, 202, 203
- Statistica, 505–509
- StatSoft, 505–509
- Stem-and-leaf, 240, 318
- stopifnot (R function), 159
- str (R function), 128, 152, 235, 317
- strata, 453
- strftime (R function), 185
- string, 177–178
- strsplit (R function), 177
- strwidth (R function), 220
- sub (R function), 177, 178
- subset (R function), 184, 282
- substr (R function), 177
- sudoku (R package), 170
- sum (R function), 352
- summary (R function), 51, 157, 236, 268, 269, 271, 317, 329, 330, 332, 343, 393, 398, 399
- sunflower (R function), 286
- support, 389
- Support vector machine (SVM), 457–460
- survey (Dataset), 197, 349, 373
- survey (R package), 173
- SVM, 457–460

- svm (R function), 456, 460
- Sys.info (R function), 164
- Sys.sleep (R function), 166
- system (R function), 163
- system.time (R function), 166, 169, 352
- t (R function), 182
- table (R function), 115, 130, 371, 373
- tail (R function), 188
- temporal analysis, 360
- test set, 375
- text (R function), 220, 245, 321
- tim.colors (R function), 223
- Togaware, 473–474
- Tools
 - Borgelt, 467–468
 - Clementine, 489
 - Enterprise Miner, 501–504
 - Equbits Foresight, 491
 - Ghostminer, 493–495
 - InductionEngine, 497
 - ODM, 499–500
 - ODMiner, 499–500
 - R, 471–472
 - Rattle, 473–474
 - SAS Enterprise Miner, 501–504
 - Statistica, 505–509
 - TreeNet, 511
 - Virtual Predict, 513–514
 - Weka, 477–479
- traceback (R function), 170
- training set, 375
- tree (R package), 370
- TreeNet, 511
- trellis.par (R function), 230
- trellis.par.get (R function), 230
- trellis.par.set (R function), 291
- true negative, 118, 373
- true positive, 118, 373
- true positive rate, 374
- ttdda (R package), 461
- type, 373
- typeof (R function), 175
- unique (R function), 189
- University of Magdeburg, 467–468
- University of Waikato, 477–479
- unlist (R function), 256
- unstack (R function), 184
- update.packages (R function), 156
- UseMethod (R function), 162
- usr, 210, 229, 282
- variable selection, 355–356
- Variables
 - Adjusted, 63, 88
 - Adjustment, 88
 - Age, 62
 - Employment, 62
- variance, 268, 328, 517
- vector, 179–180
- vector (R function), 179
- Vendors
 - Equbits, 491
 - Fujitsu, 493–495
 - Oracle, 499–500
 - PredictionWorks, 497
 - Salford Systems, 511
 - SAS, 501–504
 - SPSS, 489
 - StatSoft, 505–509
 - Togaware, 473–474
 - University of Magdeburg, 467–468
 - University of Waikato, 477–479
 - Virtual Genetics, 513–514

version, [164](#)
vignette (R function), [152](#), [153](#), [395](#)
Virtual Genetics, [513–514](#)
Virtual Predict, [513–514](#)

weight, [413](#)
weights, [368](#), [369](#)
Weka, [477–479](#)
where, [365](#)
which (R function), [187](#), [418](#)
Wickham, Hadley, [354](#)
width, [166](#)
win.metafile (R function), [226](#), [228](#)
window (R function), [19](#)
windows (R function), [225](#)
wine (Dataset), [195](#), [234](#), [236](#), [239](#),
[245](#), [253](#), [254](#), [256](#), [260](#), [315](#),
[321](#), [324](#), [417](#)
with (R function), [187](#), [212](#)
write.gif (R function), [223](#)
write.table (R function), [194](#), [195](#)

x11 (R function), [19](#), [225](#)
xlim, [204](#)
xlsReadWrite (R package), [202](#)
xpd, [210](#), [230](#), [245](#), [321](#)

y, [245](#), [321](#)
ylim, [204](#)

zoo (R package), [238](#), [313](#)