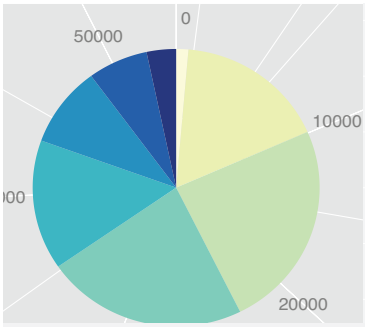
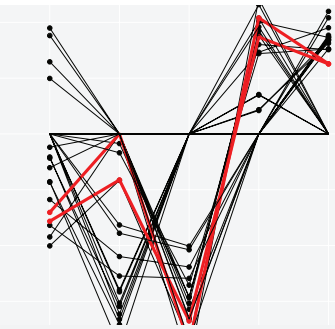
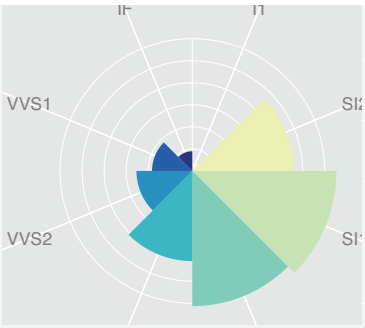
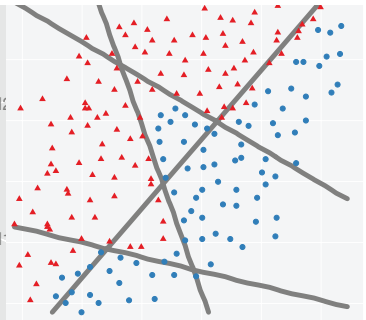
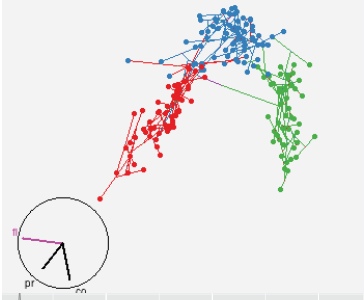


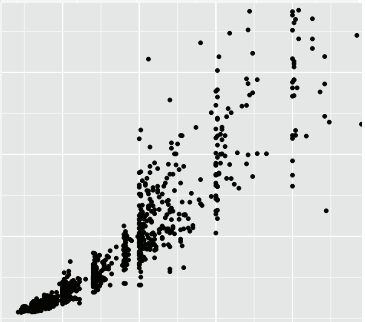


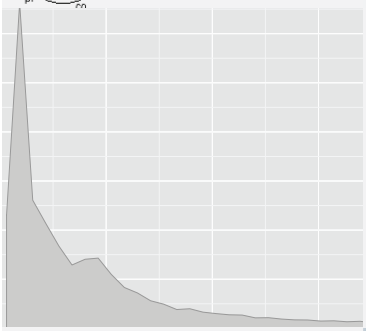


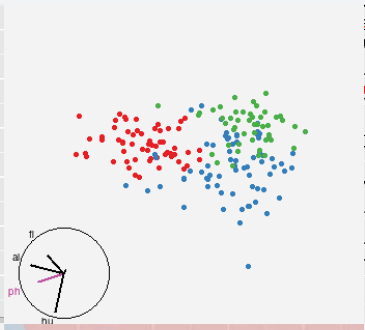


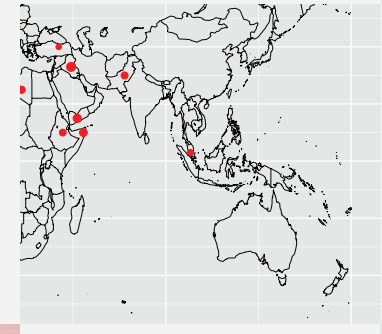


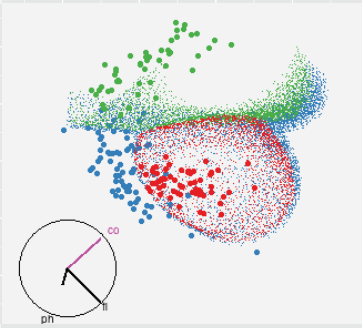
```
library(reshape)
ffm <- melt(french_fries, id=1:4,
na.rm=TRUE)
cast(ffm, subject ~ time, length)
cast(ffm, treatment ~ variable, mean)
cast(ffm, ... ~ rep)
```

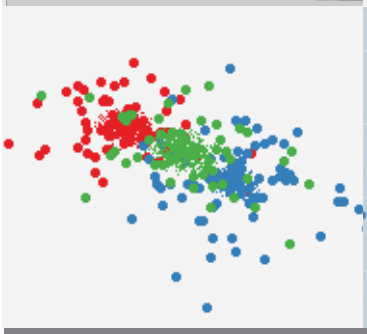


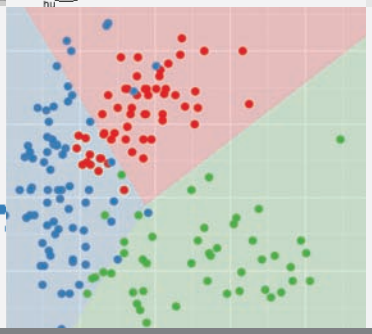


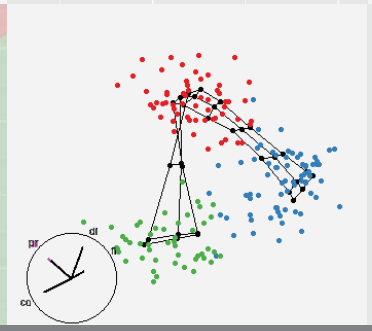


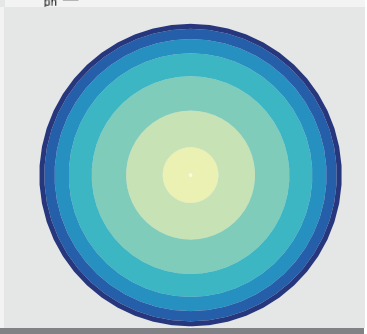












Practical tools for exploring data and models

Hadley Alexander Wickham

Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I would like to thank my major professors, Di Cook and Heike Hofmann. They have unstintingly shared their fantastic excitement about, and knowledge of, statistical graphics and data analysis. They have helped me to have a truly international education and have introduced me to many important figures in the field. They have kept me motivated throughout my PhD and have bought me far too many coffees and cakes. I can only hope that I can be as good towards my future students as they have been to me.

Many others have shaped this thesis. The generous support of Antony Unwin and Tony Rossini allowed me to spend two summers in Europe learning from masters of statistical graphics and computing. Discussions with Leland Wilkinson and Graham Wills (in person and over email) have substantially deepened my understanding of the grammar. Philip Dixon has never hesitated to share his encyclopaedic knowledge of statistical procedures, and has provided me with so many interesting problems through the AES consulting group. My entire committee have provided many interesting questions and have challenged me to think about my personal philosophy of statistics.

Last but not least, I'd like to thank my family for their unceasing love and support (not to mention the financial rescue packages!) and my friends, in Ames and in Auckland, for keeping me sane!

Chapter 1

Introduction

This thesis describes three families of tools for exploring data and models. It is organised in roughly the same way that you perform a data analysis. First, you get the data in a form that you can work with; Section 1.1 introduces the reshape framework for restructuring data, described fully in Chapter 2. Second, you plot the data to get a feel for what is going on; Section 1.2 introduces the layered grammar of graphics, described in Chapter 3. Third, you iterate between graphics and models to build a succinct quantitative summary of the data; Section 1.3 introduces strategies for visualising models, discussed in Chapter 4. Finally, you look back at what you have done, and contemplate what tools you need to do better in the future; Chapter 5 summarises the impact of my work and my plans for the future.

The tools developed in this thesis are firmly based in the philosophy of exploratory data analysis (Tukey, 1977). With every view of the data, we strive to be both curious and sceptical. We keep an open mind towards alternative explanations, never believing we have found the best model. Due to space limitations, the following papers only give a glimpse at this philosophy of data analysis, but it underlies all of the tools and strategies that are developed. A fuller data analysis, using many of the tools developed in this thesis, is available in Hobbs et al. (To appear).

1.1 Reshaping data

While data is a crucial part of statistics, it is not often that the form of data itself is discussed. Most of our methods assume that the data is a rectangular matrix, with observations in the rows and variables in the columns. Unfortunately, this is rarely how people collect and store data. Client data is never in the correct format and often requires extensive work to get it into the right form for analysis.

Data reshaping is not a traditional statistical topic, but it is an important part of data analysis. Unfortunately it has largely been overlooked in the statistical literature. It is discussed in the computer science and database literature (Gray et al., 1997; Shoshani, 1997) but these communities fail to address particularly statistical concerns, such as missing data and the need to adjust the roles of rows and columns for particular analyses.

Chapter 2 describes a framework that encompasses statistical ideas of data reshaping and aggregating. The reshape framework divides the task into two components, first describing the structure of the input data (melting) and then the structure of the output

(casting). This framework is implemented in the `reshape` package and the chapter has been published in the Journal of Statistical Software (Wickham, 2007c).

1.2 Plotting data

Plotting data is a critical part of exploratory data analysis, helping us to see the bulk of our data, as well as highlighting the unusual. As Tukey once said: "numerical quantities focus on expected values, graphical summaries on unexpected values."

Unfortunately, current open-source systems for creating graphics are sorely lacking from a practical perspective. The R environment for statistical computing provides the richest set of graphical tools, split into two libraries: base graphics (R Development Core Team, 2007) and lattice (Sarkar, 2006). Base graphics has a primitive pen on paper model, and while lattice is a step up, it has fundamental limitations. Compared to base graphics, lattice takes care of many of the minor technical details that require manual tweaking in base graphics, in particular providing matching legends and maintaining common scales across multiple plots. However, attempting to extend lattice raises fundamental questions: why are there separate functions for scatterplot and dotplots when they seem so similar? Why can you only log transform scales and not use other functions? What makes adding error bars to a plot so complicated? Extending lattice also reveals another problem. Once a lattice plot object is created, it is very difficult to modify it in a maintainable way: the components of the lattice model of graphics (Becker et al., 1996) are designed for a very specific type of display, and do not generalise well to other graphics we may wish to produce.

To do better, we need a framework that incorporates a very wide range of graphics. There have been two main attempts to develop such a framework of statistical graphics, by Bertin and Wilkinson. Bertin (1983) focuses on geographical visualisation, but also lays out principles for sound graphical construction, including suggested mappings between different types of variables and visual properties. All graphics are hand drawn, and while the underlying principles are sound, the practice of drawing graphics on a computer is rather different. The Grammar of Graphics (Wilkinson, 2005) is more modern and presents a way to concisely and formally describe a graphic. Instead of coming up with a new name for your graphic, and giving a lengthy, textual description, you can instead describe the exact components which define your graphic. The grammar is composed of seven components, as follows:

- **Data**. The most important part of any plot. Data reshaping is the responsibility of the **algebra**, which consists of three operators (nesting, crossing and blending).
- **Transformation**s create new variables from functions of existing variables, e.g. log-transforming a variable.
- **Scale**s control the mapping between variables and aesthetic properties like colour and size.
- The geometric **element** specifies the type of object used to display the data, e.g. points, lines, bars.

- A **statistic** optionally summarises the data. Statistics are critical parts of certain graphics (e.g. the bar chart and histogram).
- The **coordinate system** is responsible for computing positions on the 2d plane of the plotting surface, which is usually the Cartesian coordinate system. A subset of the coordinate system is **facet**ting, which displays different subsets of the data in small multiples, generalisation of trellising (Becker et al., 1996) which allows for non-rectangular layout.
- **Guides**, axes and legends, enable the reading of data values from the graph.

Wilkinson's grammar successfully describes a broad range of graphics, but is hampered by a lack of an available implementation: we can not use the grammar or test its claims. These issues are discussed by Cox (2007), which provides a comprehensive review of the book.

To resolve these two problems, I implemented the grammar in R. This started as a direct implementation of the ideas in the book, but as I proceeded it became clear that there are areas in which the grammar could be improved. This lead to the development of a grammar of layered graphics, described in Chapter 3. The work extends and refines the work of Wilkinson, and is implemented in the R package `ggplot2` (Wickham, 2008). This chapter has been tentatively accepted by the Journal of Computational and Graphical Statistics, and a revised version will be resubmitted shortly.

1.3 Visualising models

Graphics give us a qualitative feel for the data, helping us to make sense of what's going on. That is often not enough: many times we also need a precise mathematical model which allows us to make predictions with quantifiable uncertainty. A model is also useful as a concise mathematical summary, succinctly describing the main features of the data.

To build a good model, we need some way to compare it to the data and investigate how well it captures the salient features. To understand the model and how well it fits the data, we need tools for exploratory model analysis Unwin et al. (2003); Urbanek (2004). Graphics and models make different assumptions and have different biases. Models are not prone to human perceptual biases caused by the simplifying assumptions we make about the world, but they do have their own set of simplifying assumptions, typically required to make mathematical analysis tractable. Using one to validate the other allows us to overcome the limitations of each.

Chapter 4 describes three strategies for visualising statistical models. These strategies emphasise displaying the model in the context of the data, looking at many models and exploring the process of model fitting, as well as the final result. This chapter pulls together my experience building visualisations for classification, clustering and ensembles of linear models, as implemented by the R packages `clusterfly` (Wickham, 2007b), `classifly` (Wickham, 2007a), and `meifly` (Wickham, 2007a). I plan to submit this paper to Computational Statistics.

Chapter 2

Reshaping data with the `reshape` package

Abstract

This paper presents the reshape package for R, which provides a common framework for many types of data reshaping and aggregation. It uses a paradigm of 'melting' and 'casting', where the data are 'melted' into a form which distinguishes measured and identifying variables, and then 'cast' into a new shape, whether it be a data frame, list, or high dimensional array. The paper includes an introduction to the conceptual framework, practical advice for melting and casting, and a case study.

2.1 Introduction

Reshaping data is a common task in real-life data analysis, and it's usually tedious and frustrating. You've struggled with this task in Excel, in SAS, and in R: how do you get your clients' data into the form that you need for summary and analysis? This paper describes version 0.8.1 of the reshape package for R (R Development Core Team, 2007), which presents a new approach that aims to reduce the tedium and complexity of reshaping data.

Data often has multiple levels of grouping (nested treatments, split plot designs, or repeated measurements) and typically requires investigation at multiple levels. For example, from a long term clinical study we may be interested in investigating relationships over time, or between times or patients or treatments. To make your job even more difficult, the data probably has been collected and stored in a way optimised for ease and accuracy of collection, and in no way resembles the form you need for statistical analysis. You need to be able to fluently and fluidly reshape the data to meet your needs, but most software packages make it difficult to generalise these tasks, and new code needs to be written for each new case.

While you're probably familiar with the idea of reshaping, it is useful to be a little more formal. Data reshaping involves a rearrangement of the form, but not the content, of the data. Reshaping is a little like creating a contingency table, as there are many ways to arrange the same data, but it is different in that there is no aggregation involved. The tools presented in this paper work equally well for reshaping, retaining all existing data, and aggregating, summarising the data, and later we will explore the connection between the two.

In R, there are a number of general functions that can aggregate data, for example

tapply, by and aggregate, and a function specifically for reshaping data, reshape. Each of these functions tends to deal well with one or two specific scenarios, and each requires slightly different input arguments. In practice, you need careful thought to piece together the correct sequence of operations to get your data into the form that you want. The reshape package grew out of my frustrations with reshaping data for consulting clients, and overcomes these problems with a general conceptual framework that uses just two functions: melt and cast.

The paper introduces this framework, which will help you think about the fundamental operations that you perform when reshaping and aggregating data, but the main emphasis is on the practical tools, detailing the many forms of data that melt can consume and that cast can produce. A few other useful functions are introduced, and the paper concludes with a case study, using reshape in a real-life example.

2.2 Conceptual framework

To help us think about the many ways we might rearrange a data set, it is useful to think about data in a new way. Usually, we think about data in terms of a matrix or data frame, where we have observations in the rows and variables in the columns. For the purposes of reshaping, we can divide the variables into two groups: identifier and measured variables.

1. Identifier (id) variables identify the unit that measurements take place on. Id variables are usually discrete, and are typically fixed by design. In ANOVA notation (Y_{ijk}), id variables are the indices on the variables (i, j, k); in database notation, id variables are a composite primary key.

2. Measured variables represent what is measured on that unit (Y).

It is possible to take this abstraction one step further and say there are only id variables and a value, where the id variables also identify what measured variable the value represents. For example, we could represent this data set, which has two id variables (subject and time):

	subject	time	age	weight	height
1	John Smith	1	33	90	2
2	Mary Smith	1			2

as:

	subject	time	variable	value
1	John Smith	1	age	33
2	John Smith	1	weight	90
3	John Smith	1	height	2
4	Mary Smith	1	height	2

where each row now represents one observation of one variable. This operation is called melting and produces 'molten' data. Compared to the original data set, the molten data has a new id variable 'variable', and a new column 'value', which represents the value of

that observation. We now have the data in a form in which there are only id variables and a value.

From this form, we can create new forms by specifying which variables should form the columns and rows. In the original data frame, the 'variable' id variable forms the columns, and all identifiers form the rows. We don't have to specify all the original id variables in the new form. When we don't, the combination of id variables will no longer identify one value, but many, and we will aggregate the data as well as reshaping it. The function that reduces these many numbers to one is called an aggregation function.

The following section describes the melting operation in detail, as implemented in the reshape package.

2.3 Melting data

Melting a data frame is a little trickier in practice than it is in theory. This section describes the practical use of the `melt` function in R.

In R, melting is a generic operation that can be applied to different data storage objects including data frames, arrays and matrices. This section describes the most common case, melting a data frame. Reshape also provides support for less common data structures, including high-dimensional arrays and lists of data frames or matrices. The built-in documentation for melt, `?melt`, lists all objects that can be melted, and provides links to more details. `?melt.data.frame` documents the most common case of melting a data frame.

The `melt` function needs to know which variables are measured and which are identifiers. This distinction should be obvious from your design: if you fixed the value, it is an id variable. If you don't specify them explicitly, `melt` will assume that any factor or integer column is an id variable. If you specify only one of measured and identifier variables, `melt` assumes that all the other variables are the other sort. For example, with the `smiths` dataset, as shown above, all the following calls have the same effect:

```
melt(smiths, id=c("subject","time"),
  measured=c("age","weight","height"))
melt(smiths, id=c("subject","time"))
melt(smiths, id=1:2)
melt(smiths, measured=c("age","weight","height"))
melt(smiths)
```

```
R> melt(smiths)
     subject time variable value
1 John Smith    1      age 33.00
2 Mary Smith    1      age    NA
3 John Smith    1   weight 90.00
4 Mary Smith    1   weight    NA
5 John Smith    1   height  1.87
6 Mary Smith    1   height  1.54
```

(If you want to run these functions yourself, the smiths dataset is included in the reshape package)

Melt doesn't make many assumptions about your measured and id variables: there can be any number, in any order, and the values within the columns can be in any order too. In the current implementation, there is only one assumption that `melt` makes: all measured values must be of the same type, e.g. numeric, factor, date. We need this assumption because the molten data is stored in a R data frame, and the value column can be only one type. Most of the time this isn't a problem as there are few cases where it makes sense to combine different types of variables in the cast output.

2.3.1 Melting data with id variables encoded in column names

A more complicated case is where the variable names contain information about more than one variable. For example, here we have an experiment with two treatments (A and B) with data recorded on two time points (1 and 2), and the column names represent both the treatment and the time at which the measurement was taken. It is important to make these implicit variables explicit as part of the data preparation process, as it ensures all id variables are represented in the same way.

```
R> trial <- data.frame(id = factor(1:4), A1 = c(1, 2, 1, 2),
+     A2 = c(2, 1, 2, 1), B1 = c(3, 3, 3, 3))
R> (trialm <- melt(trial))
   id variable value
1   1       A1     1
2   2       A1     2
3   3       A1     1
4   4       A1     2
5   1       A2     2
6   2       A2     1
7   3       A2     2
8   4       A2     1
9   1       B1     3
10  2       B1     3
11  3       B1     3
12  4       B1     3
```

To fix this we need to create a time and treatment column after melting:

```
R> (trialm <- cbind(trialm,
+     colsplit(trialm$variable, names = c("treatment", "time"))
+  ))
   id variable value treatment time
1   1       A1     1         A    1
2   2       A1     2         A    1
3   3       A1     1         A    1
4   4       A1     2         A    1
5   1       A2     2         A    2
```

```
6   2       A2      1         A     2
7   3       A2      2         A     2
8   4       A2      1         A     2
9   1       B1      3         B     1
10  2       B1      3         B     1
11  3       B1      3         B     1
12  4       B1      3         B     1
```

This uses the `colsplit` function described in `?colsplit`, which deals with the simple case where variable names are concatenated together with some separator. In general variable names can be constructed in many different ways and may need a custom regular expression to tease apart multiple components.

2.3.2 Already molten data

Sometimes your data may already be in molten form. In this case, all that is necessary is to ensure that the value column is named 'value'. See `?rename` for one way to do this.

2.3.3 Missing values in molten data

Finally, it's important to discuss what happens to missing values when you melt your data. Explicitly coded missing values usually denote sampling zeros rather than structural missings, which are usually implicit in the data. Clearly a structural missing depends on the structure of the data and as we are changing the structure of the data, we might expect some changes to structural missings. Structural missings change from implicit to explicit when we change from a nested to a crossed structure. For example, imagine a dataset with two id variables, sex (male or female) and pregnant (yes or no). When the variables are nested (ie. both on the same dimension) then the missing value 'pregnant male' is encoded by its absence. However, in a crossed view, we need to add an explicit missing as there will now be a cell which must be filled with something. This is illustrated below:

	sex	pregnant	value
1	male	no	10
2	female	no	14
3	female	yes	4

	sex	no	yes
1	female	14	4
2	male	10	

In this vein, one way to describe the molten form is that it is perfectly nested: there are no crossings. For this reason, it is possible to encode all missing values implicitly, by omitting that combination of id variables, rather than explicitly, with an `NA` value. However, you may expect these to be in the data frame, and it is a bad idea for a function to throw data away by default, so you need to explicitly state that implicit missing values are ok. In most cases it is safe to get rid of them, which you can do by using `na.rm = TRUE` in the call to `melt`. The two different results are illustrated below.

```
R> melt(smiths)
     subject time variable value
```

```
1 John Smith    1      age 33.00
2 Mary Smith    1      age    NA
3 John Smith    1   weight 90.00
4 Mary Smith    1   weight    NA
5 John Smith    1   height  1.87
6 Mary Smith    1   height  1.54

R> melt(smiths, na.rm = TRUE)
     subject time variable value
1 John Smith    1      age 33.00
2 John Smith    1   weight 90.00
3 John Smith    1   height  1.87
4 Mary Smith    1   height  1.54
```

If you don't use `na.rm = TRUE` you will need to make sure to account for possible missing values when aggregating (Section 2.4.4, page 28), for example, by supplying `na.rm = TRUE` to `mean`, `sum` or `var`.

2.4 Casting molten data

Once you have your data in the molten form, you can use `cast` to rearrange it into the shape that you want. The `cast` function has two required arguments:

- `data`: the molten data set to reshape
- `formula`: the casting formula which describes the shape of the output format (if you omit this argument, cast will return the data frame in the classic form with measured variables in the columns, and all other id variables in the rows)

Most of this section explains the different casting formulas you can use. It also explains the use of two other optional arguments to cast:

- `fun.aggregate`: aggregation function to use, if necessary
- `margins`: what marginal values should be computed

2.4.1 Basic use

The casting formula has this basic form `col_var_1 + col_var_2 ~ row_var_1 + row_var_2`. This describes which variables you want to appear in the columns and which in the rows. These variables need to come from the molten data frame or be one of the following special variables:

- `.` corresponds to no variable, useful when creating formulas of the form `. ~ x` or `x ~ .`, that is, a single row or column.

- `...` represents all variables not already included in the casting formula. Including this in your formula will guarantee that no aggregation occurs. There can be only one `...` in a cast formula.
- `result_variable` is used when your aggregation formula returns multiple results. See Section 2.4.6, page 29 for more details.

The first set of examples illustrate reshaping: all the original variables are used. The first two examples show two ways to put the data back into its original form, with measured variables in the columns and all other id variables in the rows. The second and third examples are variations of this form where we put subject and then time in the columns.

```
R> cast(smithsm, time + subject ~ variable)
  time    subject age weight height
1    1 John Smith  33     90   1.87
2    1 Mary Smith  NA     NA   1.54

R> cast(smithsm, ... ~ variable)
     subject time age weight height
1 John Smith    1  33     90   1.87
2 Mary Smith    1  NA     NA   1.54

R> cast(smithsm, ... ~ subject)
  time variable John Smith Mary Smith
1    1      age      33.00         NA
2    1   weight      90.00         NA
3    1   height       1.87       1.54

R> cast(smithsm, ... ~ time)
     subject variable     1
1 John Smith      age 33.00
2 John Smith   weight 90.00
3 John Smith   height  1.87
4 Mary Smith   height  1.54
```

Because of the limitations of R data frames, it is not possible to label the columns and rows completely unambiguously. For example, note the last three examples where the data frame has no indication of the name of the variable that forms the columns. Additionally, some data values do not make valid column names, e.g. `'John Smith'`. To use these within R, you often need to surround them with backticks, e.g. `df$‘John Smith‘`.

The following examples demonstrate aggregation. Aggregation occurs when the combination of variables in the cast formula does not identify individual observations. In this case an aggregation function reduces the multiple values to a single one. See Section 2.4.4, page 28 for more details. These examples use the french fries dataset included in the `reshape` package. It is data from a sensory experiment on french fries, where different

types of frier oil, `treatment`, were tested by different people, `subject`, over ten weeks `time`.

The most severe aggregation is reduction to a single number, described by the cast formula `. ~ .`

```
R> ffm <- melt(french_fries, id = 1:4, na.rm = TRUE)

R> cast(ffm, . ~ ., length)
  value (all)
1 (all)  3471
```

Alternatively, we can summarise by the values of a single variable, either in the rows or columns.

```
R> cast(ffm, treatment ~ ., length)
  treatment (all)
1         1  1159
2         2  1157
3         3  1155

R> cast(ffm, . ~ treatment, length)
  value    1    2    3
1 (all) 1159 1157 1155
```

The following casts show the different ways we can combine two variables: one each in row and column, both in row or both in column. When multiple variables appear in the column specification, their values are concatenated to form the column names.

```
R> cast(ffm, rep ~ treatment, length)
  rep   1   2   3
1   1 579 578 575
2   2 580 579 580

R> cast(ffm, treatment ~ rep, length)
  treatment   1   2
1         1 579 580
2         2 578 579
3         3 575 580

R> cast(ffm, treatment + rep ~ ., length)
  treatment rep (all)
1         1   1   579
2         1   2   580
3         2   1   578
```

```
4         2   2   579
5         3   1   575
6         3   2   580

R> cast(ffm, rep + treatment ~ ., length)
  rep treatment (all)
1   1         1   579
2   1         2   578
3   1         3   575
4   2         1   580
5   2         2   579
6   2         3   580

R> cast(ffm, . ~ treatment + rep, length)
  value 1_1 1_2 2_1 2_2 3_1 3_2
1 (all) 579 580 578 579 575 580
```

As illustrated above, the order in which the row and column variables are specified is very important. As with a contingency table, there are many possible ways of displaying the same variables, and the way that they are organised will reveal different patterns in the data. Variables specified first vary slowest, and those specified last vary fastest. Because comparisons are made most easily between adjacent cells, the variable you are most interested in should be specified last, and the early variables should be thought of as conditioning variables. An additional constraint is that displays have limited width but essentially infinite length, so variables with many levels may need to be specified as row variables.

2.4.2 High-dimensional arrays

You can use more than one `~` to create structures with more than two dimensions. For example, a cast formula of `x ~ y ~ z` will create a 3D array with x, y, and z dimensions. You can also still use multiple variables in each dimension: `x + a ~ y + b ~ z + c`. The following example shows the resulting dimensionality of various casting formulas. I don't show the actual output here because it is too large. You may want to verify the results for yourself:

```
R> dim(cast(ffm, time ~ variable ~ treatment, mean))
[1] 10  5  3

R> dim(cast(ffm, time ~ variable ~ treatment + rep, mean))
[1] 10  5  6

R> dim(cast(ffm, time ~ variable ~ treatment ~ rep, mean))
[1] 10  5  3  2
```

```
R> dim(cast(ffm, time ~ variable ~ subject ~ treatment ~ rep))
[1] 10  5 12  3  2

R> dim(cast(ffm, time ~ variable ~ subject ~ treatment ~
+     result_variable, range))
[1] 10  5 12  3  2
```

The high-dimensional array form is useful for sweeping out margins with `sweep`, or modifying with `iapply` (Section 2.5, page 30).

The ~ operator is a type of crossing operator, as all combinations of the variables will appear in the output table. Compare this to the + operator, where only combinations that appear in the data will appear in the output. For this reason, increasing the dimensionality of the output, i.e. using more ~s, will generally increase the number of structural missings. This is illustrated below:

```
R> sum(is.na(cast(ffm, ... ~ .)))
[1] 0

R> sum(is.na(cast(ffm, ... ~ rep)))
[1] 9

R> sum(is.na(cast(ffm, ... ~ subject)))
[1] 129

R> sum(is.na(cast(ffm, ... ~ time ~ subject ~ variable ~ rep)))
[1] 129
```

Margins of high-dimensional arrays are currently unsupported.

2.4.3 Lists

You can also use `cast` to produce lists. This is done with the | operator. Using multiple variables after | will create multiple levels of nesting.

```
R> cast(ffm, treatment ~ rep | variable, mean)
$potato
  treatment        1        2
1         1 6.772414 7.003448
2         2 7.158621 6.844828
3         3 6.937391 6.998276

$buttery
  treatment        1        2
1         1 1.797391 1.762931
```

```
2         2 1.989474 1.958621
3         3 1.805217 1.631034

$grassy
  treatment         1         2
1         1 0.4456897 0.8525862
2         2 0.6905172 0.6353448
3         3 0.5895652 0.7706897

$rancid
  treatment        1        2
1         1 4.283621 3.847414
2         2 3.712069 3.537069
3         3 3.752174 3.980172

$painty
  treatment        1        2
1         1 2.727586 2.439655
2         2 2.315517 2.597391
3         3 2.038261 3.008621
```

Space considerations necessitate only printing summaries of the following lists, but you can see `?cast` for full examples.

```
R> length(cast(ffm, treatment ~ rep | variable, mean))
[1] 5

R> length(cast(ffm, treatment ~ rep | subject, mean))
[1] 12

R> length(cast(ffm, treatment ~ rep | time, mean))
[1] 10

R> sapply(cast(ffm, treatment ~ rep | time + variable, mean),
+    length)
 1  2  3  4  5  6  7  8  9 10
 5  5  5  5  5  5  5  5  5  5
```

This form is useful for input to `lapply` and `sapply`, and completes the discussion of the different types of output you can create with reshape. The remainder of the section discusses aggregation.

2.4.4 Aggregation

Whenever there are fewer cells in the cast form than there were in the original data format, an aggregation function is necessary. This formula reduces multiple cells into one, and is supplied in the `fun.aggregate` argument, which defaults (with a warning) to `length`. Aggregation is a very common and useful operation and the case studies section (Section 2.6, page 31) contains further examples of aggregation.

The aggregation function will be passed the vector of a values for one cell. It may take other arguments, passed in through `...` in `cast`. Here are a few examples:

```
R> cast(ffm, . ~ treatment)
  value    1    2    3
1 (all) 1159 1157 1155

R> cast(ffm, . ~ treatment, function(x) length(x))
  value    1    2    3
1 (all) 1159 1157 1155

R> cast(ffm, . ~ treatment, length)
  value    1    2    3
1 (all) 1159 1157 1155

R> cast(ffm, . ~ treatment, sum)
  value      1      2      3
1 (all) 3702.4 3640.4 3640.2

R> cast(ffm, . ~ treatment, mean)
  value        1        2        3
1 (all) 3.194478 3.146413 3.151688

R> cast(ffm, . ~ treatment, mean, trim = 0.1)
  value        1        2        3
1 (all) 2.595910 2.548112 2.589081
```

You can also display margins and use functions that return multiple results. See the next two sections for details.

2.4.5 Margins

It's often useful to be able to add statistics to the margins of your tables, for example, as suggested by Chatfield (1995). You can tell `cast` to display all margins with `margins = TRUE`, or list individual variables in a character vector, `margins=c("subject","day")`. There are two special margins, `"grand_col"` and `"grand_row"`, which display margins for the overall columns and rows respectively. Margins are indicated with '(all)' as the value of the variable that was margined over.

These examples illustrate some of the possible ways to use margins. I've used sum as the aggregation function so that you can check the results yourself. Note that changing the order and position of the variables in the cast formula affects the margins that can be computed.

```
R> cast(ffm, treatment ~ ., sum, margins = TRUE)
  treatment   (all)
1         1  3702.4
2         2  3640.4
3         3  3640.2
4     (all) 10983.0

R> cast(ffm, treatment ~ ., sum, margins = "grand_row")
  treatment   (all)
1         1  3702.4
2         2  3640.4
3         3  3640.2
4     (all) 10983.0

R> cast(ffm, treatment ~ rep, sum, margins = TRUE)
  treatment      1      2   (all)
1         1 1857.3 1845.1  3702.4
2         2 1836.5 1803.9  3640.4
3         3 1739.1 1901.1  3640.2
4     (all) 5432.9 5550.1 10983.0

R> cast(ffm, treatment + rep ~ ., sum, margins = TRUE)
   treatment   rep   (all)
1          1     1  1857.3
2          1     2  1845.1
3          1 (all)  3702.4
4          2     1  1836.5
5          2     2  1803.9
6          2 (all)  3640.4
7          3     1  1739.1
8          3     2  1901.1
9          3 (all)  3640.2
10     (all) (all) 10983.0
```

2.4.6 Returning multiple values

Occasionally it is useful to aggregate with a function that returns multiple values, e.g. `range` or `summary`. This can be thought of as combining multiple `casts` each with an aggregation function that returns one variable. To display this we need to add an extra variable, `result_variable` that differentiates the multiple return values. By default, this

new id variable will be shown as the last column variable, but you can specify the position manually by including result_variable in the casting formula.

```
R> cast(ffm, treatment ~ ., summary)
  treatment Min. X1st.Qu. Median  Mean X3rd.Qu. Max.
1         1    0        0    1.6 3.194      5.4 14.9
2         2    0        0    1.4 3.146      5.4 14.9
3         3    0        0    1.5 3.152      5.7 14.5

R> cast(ffm, treatment ~ ., quantile, c(0.05, 0.5, 0.95))
  treatment X5. X50. X95.
1         1   0  1.6 11.0
2         2   0  1.4 10.7
3         3   0  1.5 10.6

R> cast(ffm, treatment ~ rep, range)
  treatment 1_X1 1_X2 2_X1 2_X2
1         1    0 14.9    0 14.3
2         2    0 14.9    0 13.7
3         3    0 14.5    0 14.0
```

You can also supply a vector of functions:

```
R> cast(ffm, treatment ~ rep, c(min, max))
  treatment 1_min 1_max 2_min 2_max
1         1     0  14.9     0  14.3
2         2     0  14.9     0  13.7
3         3     0  14.5     0  14.0

R> cast(ffm, treatment ~ result_variable + rep, c(min, max))
  treatment min_1 min_2 max_1 max_2
1         1     0     0  14.9  14.3
2         2     0     0  14.9  13.7
3         3     0     0  14.5  14.0
```

2.5 Other convenience functions

There are many other problems encountered in practical analysis that can be painful to overcome without some handy functions. This section describes some of the functions that reshape provides to make dealing with data a little bit easier. More details are provided in the respective documentation.

2.5.1 Factors

- `combine_factor` combines levels in a factor. For example, if you have many small levels you can combine them together into an 'other' level.
- `reorder_factor` reorders a factor based on another variable. For example, you can order a factor by the average value of a variable for each level.

2.5.2 Data frames

- `rescaler` performs column-wise rescaling of data frames, with a variety of different scaling options including rank, common range and common variance. It automatically preserves non-numeric variables.
- `merge.all` merges multiple data frames together, an extension of `merge` in base R. It assumes that all columns with the same name should be equated.
- `rbind.fill` `rbind`s two data frames together, filling in any missing columns in the second data frame with missing values.

2.5.3 Miscellaneous

- `round_any` allows you to round a number to any degree of accuracy, e.g. to the nearest 1, 10, or any other number.
- `iapply` is an idempotent version of the `apply` function. It is idempotent in the sense that `iapply(x, a, function(x) x)` is guaranteed to return `x` for any value of a. This is useful when dealing with high-dimensional arrays as it will return the array in the same shape that you sent it. It also supports functions that return matrices or arrays in a sensible manner.

2.6 Case study: French fries

These data are from a sensory experiment investigating the effect of different frying oils on the taste of French fries over time. There are three different types of frying oils (treatment), each in two different fryers (rep), tested by 12 people (subject) on 10 different days (time). The sensory attributes recorded, in order of desirability, are potato, buttery, grassy, rancid, painty flavours. The first few rows of the data are shown in Table 2.1.

We first melt the data to use in subsequent analyses.

```
R> ffm <- melt(french_fries, id = 1:4, na.rm = TRUE)
R> head(ffm)
  time treatment subject rep variable value
1    1         1       3   1   potato   2.9
2    1         1       3   2   potato  14.0
3    1         1      10   1   potato  11.0
4    1         1      10   2   potato   9.9
5    1         1      15   1   potato   1.2
```

time	trt	subject	rep	potato	buttery	grassy	rancid	painty
1	1	3	1.00	2.90	0.00	0.00	0.00	5.50
1	1	3	2.00	14.00	0.00	0.00	1.10	0.00
1	1	10	1.00	11.00	6.40	0.00	0.00	0.00
1	1	10	2.00	9.90	5.90	2.90	2.20	0.00
1	1	15	1.00	1.20	0.10	0.00	1.10	5.10
1	1	15	2.00	8.80	3.00	3.60	1.50	2.30

Table 2.1: First few rows of the French fries dataset

```
6    1          1       15   2   potato   8.8
```

2.6.1 Investigating balance

One of the first things we might be interested in is how balanced this design is, and if there are many different missing values. We are interested in missingness, so we removed explicit missing values to put structural and non-structural missings on an equal footing.

We can investigate balance using length as our aggregation function:

```
R> cast(ffm, subject ~ time, function(x) 30 - length(x),
+    margins=T)
   subject   1   2   3   4   5   6   7   8   9  10 (all)
1        3  30  30  30  30  30  30  30  30  30  NA   270
2       10  30  30  30  30  30  30  30  30  30  30   300
3       15  30  30  30  30  25  30  30  30  30  30   295
4       16  30  30  30  30  30  30  30  29  30  30   299
5       19  30  30  30  30  30  30  30  30  30  30   300
6       31  30  30  30  30  30  30  30  30  NA  30   270
7       51  30  30  30  30  30  30  30  30  30  30   300
8       52  30  30  30  30  30  30  30  30  30  30   300
9       63  30  30  30  30  30  30  30  30  30  30   300
10      78  30  30  30  30  30  30  30  30  30  30   300
11      79  30  30  30  30  30  30  29  28  30  NA   267
12      86  30  30  30  30  30  30  30  30  NA  30   270
13   (all) 360 360 360 360 355 360 359 357 300 300  3471
```

Each subject should have had 30 observations at each time, so by displaying the difference we can more easily see where the data are missing.

```
R> cast(ffm, subject ~ time, function(x) 30 - length(x))
   subject 1 2 3 4 5 6 7 8  9 10
1        3 0 0 0 0 0 0 0 0  0 NA
2       10 0 0 0 0 0 0 0 0  0  0
3       15 0 0 0 0 5 0 0 0  0  0
```

```
4       16 0 0 0 0 0 0 0 1  0  0
5       19 0 0 0 0 0 0 0 0  0  0
6       31 0 0 0 0 0 0 0 0 NA  0
7       51 0 0 0 0 0 0 0 0  0  0
8       52 0 0 0 0 0 0 0 0  0  0
9       63 0 0 0 0 0 0 0 0  0  0
10      78 0 0 0 0 0 0 0 0  0  0
11      79 0 0 0 0 0 0 1 2  0 NA
12      86 0 0 0 0 0 0 0 0 NA  0
```

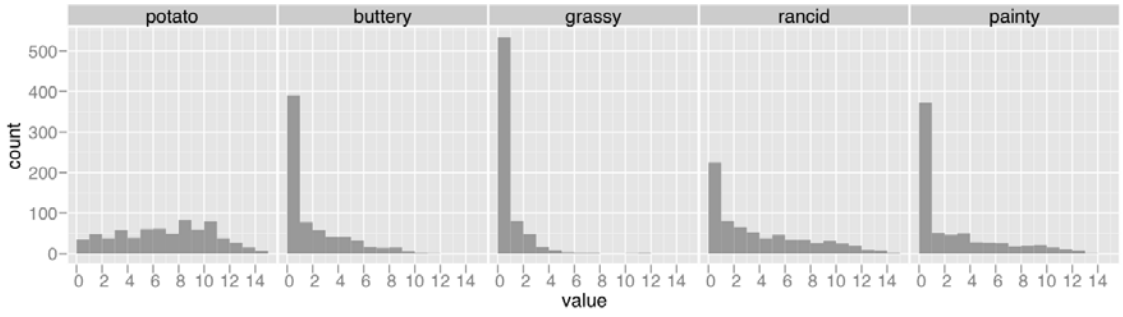
There are two types of missing observations here: a non-zero value, or a missing value. A missing value represents a subject with no records at a given time point; they did not turn up on that day. A non-zero value represents a subject who did turn up, but perhaps due to a recording error, missed some observations.

We can also easily see the range of values that each variable takes:

```
R> cast(ffm, variable ~ ., c(min, max))
  variable min  max
1   potato   0 14.9
2  buttery   0 11.2
3   grassy   0 11.1
4   rancid   0 14.9
5   painty   0 13.1
```

Better than just looking at the ranges is to look at the distribution, with a histogram. Here we use the molten data directly, facetting (aka conditioning or trellising) by the measured variable.

```
R> qplot(value, data=ffm, geom="histogram",
+    facets = . ~ variable, binwidth=1)
```



2.6.2 Tables of means

When creating these tables, it is a good idea to restrict the number of digits displayed. You can do this globally, by setting `options(digits=2)`, or locally, by using `round_any`.

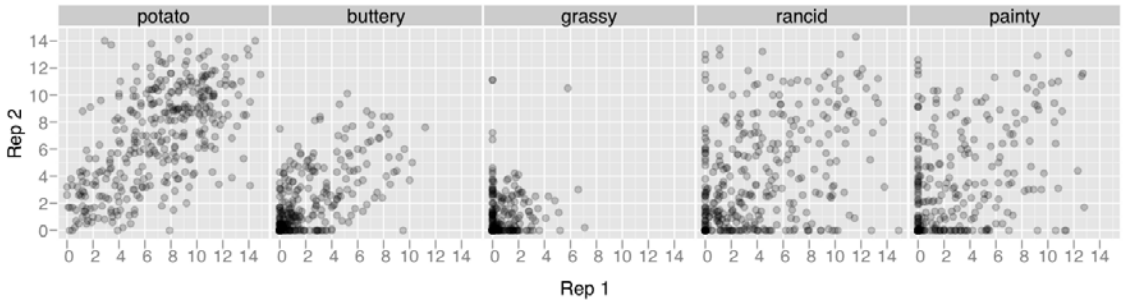
Since the data are fairly well balanced, we can do some (crude) investigation as to the effects of the different treatments. For example, we can calculate the overall means for each sensory attribute for each treatment:

```
R> options(digits = 2)
R> cast(ffm, treatment ~ variable, mean,
+    margins = c("grand_col", "grand_row"))
  treatment potato buttery grassy rancid painty (all)
1         1    6.9     1.8   0.65    4.1    2.6   3.2
2         2    7.0     2.0   0.66    3.6    2.5   3.1
3         3    7.0     1.7   0.68    3.9    2.5   3.2
4     (all)    7.0     1.8   0.66    3.9    2.5   3.2
```

It doesn't look like there is any effect of treatment. This could be confirmed using a more formal analysis of variance.

2.6.3 Investigating inter-rep reliability

Since we have a repetition over treatments, we might be interested in how reliable each subject is: are the scores for the two repetitions highly correlated? We can explore this graphically by reshaping the data and plotting the data. Our graphical tools work best when the things we want to compare are in different columns, so we'll cast the data to have a column for each rep. In quality control literature, this is known as a Youden plot (Youden, 1959).



The correlation looks strong for potatoey and buttery, and poor for rancid and painty. Grassy has many low values, particularly zeros, as seen in the histogram. This reflects the training the participants received for the trial: they were trained to taste potato and buttery flavours, their grassy reference flavour was parsley (very grassy compared to French fries), and they were not trained on rancid or painty flavours.

If we wanted to explore the relationships between subjects or times or treatments we could follow similar steps.

2.7 Where to go next

You can find a quick reference and more examples in `?melt` and `?cast`. You can find some additional information on the reshape website http://had.co.nz/reshape, including copies of presentations and papers related to reshape.

I would like to include more case studies of reshape in use. If you have an interesting example, or there is something you are struggling with please let me know: h.wickham@gmail.com.

2.8 Acknowledgements

I'd like to thank Antony Unwin for his comments about the paper and package, which have lead to a significantly more consistent and user-friendly interface. The questions and comments of the users of the reshape package, Kevin Wright, François Pinard, Dieter Menne, Reinhold Kleigl, and many others, have also contributed greatly to the development of the package.

This material is based upon work supported by the National Science Foundation under Grant No. 0706949.

Chapter 3

A layered grammar of graphics

Abstract

A grammar of graphics is a tool which enables us to concisely describe the components of a graphic. A grammar of graphics allows us to move beyond named graphics and gain insight into the deep structure that underlies statistical graphics. This paper builds on Wilkinson (2005), describing extensions and refinements developed while building an open source implementation of the grammar of graphics for R, ggplot2.

The topics in this paper include an introduction to the grammar by working through the process of creating a plot, and discussing the components that we need. The grammar is then presented formally and compared to Wilkinson's grammar, highlighting the hierarchy of defaults, and the implications of embedding a graphical grammar into a programming language. The power of the grammar is illustrated with a selection of examples that explore different components, and their interactions, in more detail. The paper concludes by discussing some perceptual issues, and thinking about how we can build on the grammar to learn how to create graphical "poems".

3.1 Introduction

What is a graphic? How can we succinctly describe a graphic? And how can we create the graphic that we have described? These are important questions for the field of statistical graphics.

One way to answer these questions is to develop a grammar, "the fundamental principles or rules of an art or science" (OED Online, 1989). A good grammar will allow us to gain insight into the composition of complicated graphics, and reveal unexpected connections between seemingly different graphics (Cox, 1978). A grammar provides a strong foundation for understanding a diverse range of graphics. A grammar may also help guide us on what a well-formed or correct graphic looks like, but there will still be many grammatically correct but nonsensical graphics. This is easy to see by analogy to the English language: good grammar is just the first step in creating a good sentence.

The seminal work in graphical grammars is "The Grammar of Graphics" by Wilkinson et al. (2005), which proposes a grammar which can be used to describe and construct a wide range of graphics. This paper proposes an alternative parameterisation of the grammar, based around the idea of building up a graphic from multiple layers of data. The

grammar differs from Wilkinson's in its arrangement of the components, the development of a hierarchy of defaults, and in that it is embedded inside another programming language. These three sections form the core of the paper, and compare and contrast to Wilkinson's grammar. These sections are followed by some implications of the grammar, a discussion of perceptual issues otherwise not mentioned by the grammar, and finally some ideas for building higher level tools to support data analysis.

The ideas presented in this paper have been implemented in the open-source R package, ggplot2, available from CRAN. More details about the grammar and implementation, including a comprehensive set of examples, can be found on the package website http://had.co.nz/ggplot2. Ggplot2 is the analogue of GPL, the implementation of Wilkinson's grammar in SPSS.

3.2 How to build a plot

When creating a plot we start with data. We will use the trivial dataset shown in Table 3.1 as an example. It has four variables, A, B, C, and D, and four observations.

A	B	C	D
2	3	4	a
1	2	1	a
4	5	15	b
9	10	80	b

Table 3.1: Simple dataset.

Let's draw a scatterplot of A vs. C. What exactly is a scatterplot? One way to describe it is that we're going to draw a point for each observation, and we will position the point horizontally according to the value of A, and vertically according to C. For this example, we will also map categorical variable D to the colour of the points. The first step in making this plot is to create a new dataset which reflects the mapping of x-position to A, y-position to C and colour to D. x-position, y-position and colour are examples of aesthetics, things that we can perceive on the graphic. We will also remove all other variables that do not appear in the plot. This is shown in Table 3.2.

x	y	*colour*
2	4	a
1	1	a
4	15	b
9	80	b

Table 3.2: Simple dataset with variables named according to the aesthetic that they use.

We can create many different types of plots using this same basic specification. For example, if we were to draw lines instead of points we would get a line plot. If we used bars, we'd get a bar plot. Bars, lines and points are all examples of geometric objects.

The next thing we need to do is to convert these numbers measured in data units to numbers measured in physical units, things that the computer can display. To do that we need to know two things: that we're going to use linear scales, and a Cartesian coordinate system. We can then convert the data units to aesthetic units, which have meaning to the underlying drawing system. For example, to convert from a continuous data value to a horizontal pixel coordinate, we need a function like the following:

$$\text{floor}(\frac{x - \min(x)}{\text{range}(x)} * screen\ width)$$

In this example, we will scale the x-position to $[0, 200]$ and the y-position to $[0, 300]$. The procedure is similar for other aesthetics, such as colour: here we map "a" to red, and "b" to blue. The results of these scalings are shown in Table 3.3. These transformations are the responsibility of *scales*, described in detail in Section 3.3.2.

x	y	$colour$
25	11	red
0	0	red
75	53	blue
200	300	blue

Table 3.3: Simple dataset with variables mapped into aesthetic space.

In general, there is another step that we've skipped in this simple example: a statistical transformation. Here we are using the identity transformation, but there are many others that are useful, such as binning or aggregating. Statistical transformations, or stats, are described in detail in Section 3.3.1.

Finally, we need to render this data to create the graphical objects that are displayed on the screen. To create a complete plot we need to combine graphical objects from three sources: the data, represented by the point geom; the scales and coordinate system, which generates axes and legends so that we can read values from the graph; and the plot annotations, such as the background and plot title. These components are shown in Figure 3.1. Combining and displaying these graphical objects produces the final plot, as in Figure 3.2.

3.2.1 A more complicated plot

Now that you are acquainted with drawing a simple plot, we will create a more complicated plot. The big difference with this plot is that we'll use faceting. Faceting is also known as conditioning, trellising and latticing, and produces small multiples showing different subsets of the data. If we facet the previous plot by D we will get a plot that looks like Figure 3.3, where each value of D is displayed in a different panel.

Faceting splits the original dataset into a dataset for each subset, so the data that underlies Figure 3.3 looks like Table 3.4.

The first steps of plot creation proceed as before, but new steps are necessary when we get to the scales. Scaling actually occurs in three parts: transforming, training and mapping.

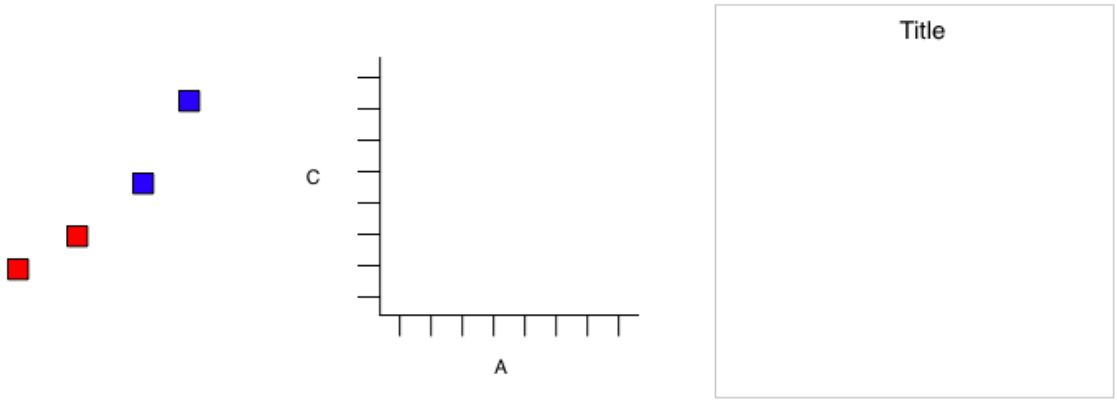


Figure 3.1: Graphics objects produced by (from left to right): geometric objects, scales and coordinate system, plot annotations.

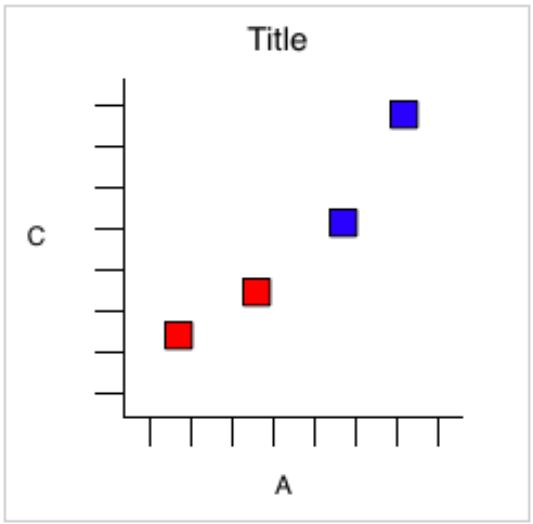


Figure 3.2: The final graphic, produced by combining the pieces in Figure 3.1.

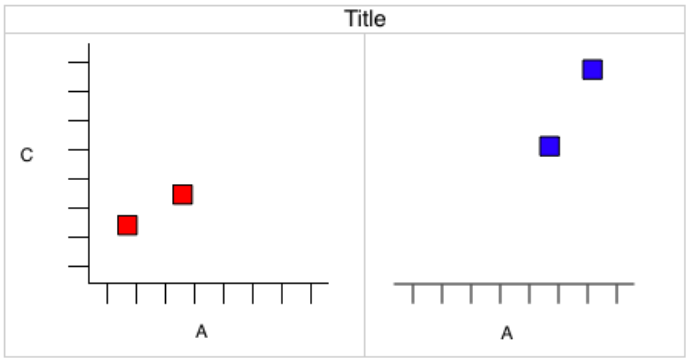


Figure 3.3: A more complicated plot, which is faceted by variable D. Here the faceting uses the same variable that is mapped to colour so that there is some redundancy in our visual representation. This allows us to easily see how the data has been broken into panels.

	x	y	$colour$
a	2	4	red
a	1	1	red
b	4	15	blue
b	9	80	blue

Table 3.4: Simple dataset faceted into subsets.

- Scale transformation occurs before statistical transformation so that statistics are computed on the scale-transformed data. This ensures that a plot of $log(x)$ vs $log(y)$ on linear scales looks the same as x vs y on log scales. See Section 3.6.3 for more details. Transformation is only necessary for non-linear scales, because all statistics are location-scale invariant.

- After the statistics are computed, each scale is trained on every faceted dataset (a plot can contain multiple datasets, e.g. raw data and predictions from a model). The training operation combines the ranges of the individual datasets to get the range of the complete data. If scales were applied locally, comparisons would only be meaningful within a facet. This is shown in Table 3.5.

- Finally the scales map the data values into aesthetic values. This gives Table 3.6 which is essentially identical to Table 3.2 apart from the structure of the datasets. Given that we end up with an essentially identical structure you might wonder why we don't simply split up the final result. There are several reasons for this. It makes writing statistical transformation functions easier, as they only need to operate on a single facet of data, and some need to operate on a single subset, for example, calculating a percentage. Also, in practice we may have a more complicated training scheme for the position scales so that different columns or rows can have different x and y scales.

	x	y	$colour$
a	200	300	red
a	0	0	red
b	0	0	red
b	200	300	red

Table 3.5: Local scaling, where data are scaled independently within each facet. Note that each facet occupies the full range of positions, and only uses one colour. Comparisons across facets are not necessarily meaningful.

	x	y	$colour$
a	25	11	red
a	0	0	red
b	75	53	blue
b	200	300	blue

Table 3.6: faceted data correctly mapped to aesthetics. Note the similarity to Table 3.3.

3.2.2 Summary

In the examples above, we have seen some of the components that make up a plot:

- data and aesthetic mappings,
- geometric objects,
- scales,
- and faceting.

We have also touched on two other components:

- statistical transformations,
- and the coordinate system.

Together, the data, mappings, statistical transformation and geometric object form a layer. A plot may have multiple layers, for example, when we overlay a scatterplot with a smoothed line.

3.3 Components of the layered grammar

To be precise, the layered grammar defines the components of a plot as:

- A default dataset and set of mappings from variables to aesthetics.
- One or more layers, each composed of a geometric object, a statistical transformation, and a position adjustment, and optionally, a dataset and aesthetic mappings.

- One scale for each aesthetic mapping used.
- A coordinate system.
- A faceting specification.

These high-level components are quite similar to those of Wilkinson's grammar, as shown in Figure 3.4. In both grammars, the components are independent, meaning that we can generally change a single component in isolation. There are more differences within the individual components, which are described in the details which follow.

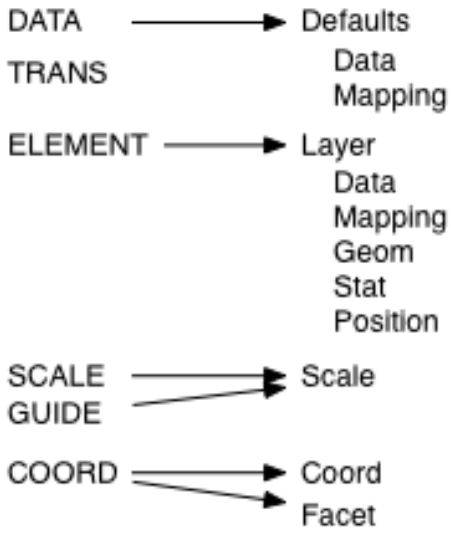


Figure 3.4: Mapping between components of Wilkinson's grammar (left) and the layered grammar (right)

The layer component is particularly important as it determines the physical representation of the data, with the combination of stat and geom defining many familiar named graphics: the scatterplot, histogram, contourplot, and so. In practice, many plots have (at least) three layers: the data, context for the data, and a statistical summary of the data. For example, to visualise a spatial point process, we might display the points themselves, a map giving some context to the locations of points, and contours of a 2d density estimate.

This grammar is useful for both the user and the developer of statistical graphics. For the user, it makes it easier to iteratively update a plot, changing a single feature at a time. The grammar is also useful because it suggests the high level aspects of a plot that *can* be changed, giving us a framework to think about graphics, and hopefully shortening the distance from mind to paper. It also encourages the use of graphics customised to a particular problem, rather than relying on generic named graphics.

For the developer, it makes it much easier to add new capabilities. You only need to add the one component that you need, and continue to use the all the other existing components. For example, you can add a new statistical transformation, and continue to use the existing scales and geoms. It is also useful for discovering new types of graphics, as the grammar effectively defines the parameter space of statistical graphics.

3.3.1 Layers

Layers are responsible for creating the objects that we perceive on the plot. A layer is composed of four parts:

- data and aesthetic mapping,
- a statistical transformation (stat),
- a geometric object (geom)
- and a position adjustment.

These parts are described in detail below.

Usually all the layers on a plot have something in common, which is typically that they are different views of the same data, e.g. a scatterplot with overlaid smoother.

A layer is the equivalent of Wilkinson's ELEMENT. However, the parameterisation is rather different. In Wilkinson's grammar, all the parts of an element are intertwined, while in the layered grammar they are separate, as shown by Figure 3.5. This makes it possible to omit parts from the specification and rely on defaults: if the stat is omitted, the geom will supply a default; if the geom is omitted, the stat will supply a default; if the mapping is omitted, the plot default will be used. These defaults are discussed further in Section 3.4. In Wilkinson's grammar, the dataset is implied by the variable names, while in the layered grammar it can be specified separately.

```
line(position(smooth.linear(x * y)), colour(z))
layer(aes(x = x, y = y, colour = z), geom="line", stat="smooth")
```

Figure 3.5: Difference between GPL (top) and `ggplot2` (bottom) parameterisations.

Data and mapping

Data is obviously a critical part of the plot, but it is important to remember that it is independent from the other components: we can construct a graphic that can be applied to multiple datasets. Data is what turns an abstract graphic into a concrete graphic.

Along with the data, we need a specification of which variables are mapped to which aesthetics. For example, we might map weight to x position, height to y position and age to size. The details of the mapping are described by the scales, Section 3.3.2. Choosing a good mapping is crucial for generating a useful graphic, as described in Section 3.8.

Statistical transformation

A statistical transformation, or **stat**, transforms the data, typically by summarising it in some manner. For example, a useful stat is the smoother, which calculates the mean of y, conditional on x, subject to some restriction that ensures smoothness. Table 3.7 lists some of the stats available in ggplot2. To make sense in a graphic context a stat must be location-scale invariant: $\mathrm{f}(x + a) = \mathrm{f}(x) + a$ and $\mathrm{f}(b \cdot x) = b \cdot \mathrm{f}(x)$. This ensures that the

Name	Description
bin	Divide continuous range into bins and count
boxplot	Compute statistics necessary for boxplot
contour	Calculate contour lines
density	Compute 1d density estimate
identity	Identity transformation, $f(x) = x$
jitter	Jitter values by adding small random value
qq	Calculate values for quantile-quantile plot
quantile	Quantile regression
smooth	Smoothed conditional mean of y given x
summary	Aggregate values of y for given x
sortx	Sort values in order of ascending x
unique	Remove duplicated observations

Table 3.7: Some statistical transformations provided by ggplot2. The user is able to supplement this list in a straight forward manner.

transformation is invariant under translation and scaling, which are common operations on a graphic.

A stat takes a dataset as input and returns a dataset as output, and so a stat can add new variables to the original dataset. It is possible to map aesthetics to these new variables. For example, one way to describe a histogram is as a binning of a continuous variable, plotted with bars whose height is proportional to the number of points in each bin, as described in Section 3.6.1. Another useful example is mapping the size of the lines in a contour plot to the height of the contour.

The actual statistical method used by a stat is conditional on the coordinate system. For example, a smoother in polar coordinates should use circular regression, and in 3d should return a 2d surface rather than a 1d curve. However, many statistical operations have not been derived for non-Cartesian coordinates and we so we generally fall back to Cartesian coordinates for calculation, which, while not strictly correct, will normally be a fairly close approximation. This issue is not discussed in the Wilkinson's Grammar.

Geometric object

Geometric objects, or **geom**s for short, control the type of plot that you create. For example, using a point geom will create a scatterplot, while using a line geom will create a line plot. We can classify geoms by their dimensionality:

- 0d: point, text
- 1d: path, line (ordered path)
- 2d: polygon, interval

Geometric objects are an abstract component and can be rendered in different ways. Figure 3.6 illustrates four possible renderings of the interval geom.

Figure 3.6: Four representations of an interval geom. From left to right: as a bar, as a line, as a error bar, and (for continuous x) as a ribbon.

Geoms are mostly general purpose, but do require certain outputs from a statistic. For example, the boxplot geom requires the position of the upper and lower fences, upper and lower hinges, the middle bar and the outliers. Any statistic used with the boxplot needs to provide these values.

Every geom has a default statistic, and every statistic a default geom. For example, the bin statistic defaults to using the bar geom to produce a histogram. Over-riding these defaults will still produce valid plots, but they may violate graphical conventions.

Each geom can only display certain aesthetics. For example, a point geom has position, colour, and size aesthetics. A bar geom has all those, plus height, width and fill colour. Different parameterisations may be useful. For example, instead of location and dimension, we could parameterise the bar with locations representing the four corners. Parameterisations which involve dimension (e.g. height and width) only make sense for Cartesian coordinate systems. For example, height of a bar geom in polar coordinates corresponds to radius of a segment. For this reason location based parameterisations are used internally.

Position adjustment

Sometimes we need to tweak the position of the geometric elements on the plot, when otherwise they would obscure each other. This is most common in bar plots, where we stack or dodge (place side-by-side) the bar to avoid overlaps. In scatterplots with few unique x and y values, we sometimes randomly jitter (Chambers et al., 1983) the points to reduce overplotting. Wilkinson calls these collision modifiers.

3.3.2 Scales

A **scale** controls the mapping from data to aesthetic attributes, and so we need one scale for each aesthetic property used in a layer. Scales are common across layers to ensure a consistent mapping from data to aesthetics. Some scales are illustrated in Figure 3.7.

A scale is a function, and its inverse, along with a set of parameters. For example, the colour gradient scale maps a segment of the real line to a path through a colour space. The parameters of the function define whether the path is linear or curved, which colour space to use (eg. LUV or RGB), and the start and end colours.

The inverse function is used to draw a guide so that you can read values from the graph. Guides are either axes (for position scales) or legends (for everything else). It may seem like some mappings don't have an inverse (e.g. map treatments A and B to red, and C, D and E to blue), but this can be thought of as collapsing some of the data values prior to plotting.

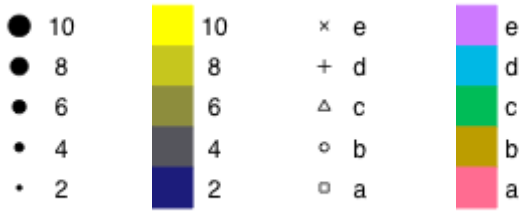


Figure 3.7: Examples of four scales from ggplot2. From left to right: continuous variable mapped to size and colour, discrete variable mapped to shape and colour. The ordering of scales seems upside-down, but this matches the labelling of the y-axis: small values occur at the bottom.

Scales typically map from a single variable to a single aesthetic, but there are exceptions. For example, we can map one variable to hue and another to saturation, to create a single aesthetic, colour. We can also create redundant mappings, mapping the same variable to multiple aesthetics. This is particularly useful when producing a graphic that works in both colour and black and white.

The scale of the layered grammar is equivalent to the SCALE and GUIDE of Wilkinson's grammar. There are two types of guides: scale guides and annotation guides. In the layered grammar, the guides (axes and legends) are largely drawn automatically based on options supplied to the relative scales. Annotation guides are not necessary as they can be created with creative use geoms if data dependent, or the underlying drawing system can be accessed directly. Scales are also computed somewhat differently as it is possible to map a variable produced by a statistic to an aesthetic. This requires two passes of scaling, before and after the statistical transformation.

3.3.3 Coordinate system

A coordinate system, **coord** for short, maps the position of objects onto the plane of the plot. Position is often specified by two coordinates (x, y), but could be any number of coordinates. The Cartesian coordinate system is the most common coordinate system for two dimensions, while polar coordinates and various map projections are used less frequently. For higher dimensions, we have parallel coordinates (a projective geometry), mosaic plots (a hierarchical coordinate system) and linear projections onto the plane.

Coordinate systems affect all position variables simultaneously and differ from scales in that they also change the appearance of the geometric objects. For example, in polar coordinates, bar geoms look like segments of a circle. Additionally, scaling is performed before statistical transformation, while coordinate transformations occur afterward. The consequences of this are shown in Section 3.6.3.

Coordinate systems control how the axes and grid lines are drawn. Figure 3.8 illustrates three different types of coordinate systems. Very little advice is available for drawing these for non-Cartesian coordinate systems, so a lot of work needs to be done to produce polished output.

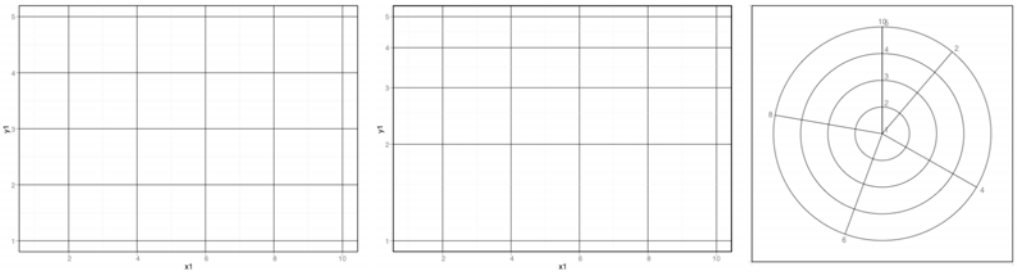


Figure 3.8: Examples of axes and grid lines for three coordinate systems: Cartesian, semi-log and polar. The polar coordinate system illustrates the difficulties associated with non-Cartesian coordinates: it is hard to draw the axes correctly!

3.3.4 Faceting

There is also another thing that turns out to be sufficiently useful that we should include it in our general framework: faceting (also known as conditioned or trellis plots). This makes it easy to create small multiples of different subsets of an entire dataset. This is a powerful tool when investigating whether patterns hold across all conditions. The faceting specification describes which variables should be used to split up the data, and how they should be arranged in a grid.

In Wilkinson's grammar, faceting is an aspect of the coordinate system, with a somewhat complicated parameterisation: the faceting variable is specified within the ELEMENT and a separate COORD specifies that the coordinate system should be faceted by this variable. This is simplified in the layered grammar as the faceting is independent of the layer and within-facet coordinate system. This is less flexible, as the layout of the facets always occurs in a Cartesian coordinate system, but in practice is not limiting. Figure 3.9 illustrates the specification.

```
COORD: rect(dim(3), dim(1,2))
ELEMENT: point(position(x * y * z))

geom_point(aes(x, y)) + facet_grid(. ~ z)
```

Figure 3.9: Difference between GPL (top) and ggplot2 (bottom) parameterisations. Note that z is included in the position specification for the GPL element.

3.4 A hierarchy of defaults

The five major components of the layered grammar allow us to completely and explicitly describe a wide range of graphics. However, having to describe every component, every time, quickly becomes tiresome. This section describes the hierarchy of defaults that simplify the work of making a plot. There are defaults present in GPL, but they are not described in the Grammar of Graphics (Wilkinson, 2005). This section will also serve to demonstrate the syntax of the `ggplot2` package.

To illustrate how the defaults work, I will show how to create the two graphics of Figure 3.10. These plots show the relationship between the price and weight (in carats) of

1000 diamonds.

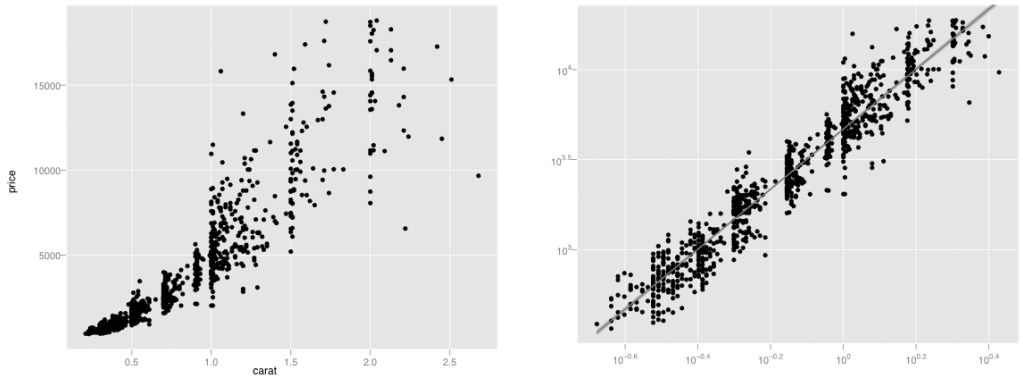


Figure 3.10: (Left) Scatterplot of price vs carat. (Right) scatterplot of price vs carat, with log-transformed scales, and a linear smooth layered on top.

The full specification of the scatterplot of price vs carat is:

```
ggplot() +
layer(
    data = diamonds, mapping = aes(x = carat, y = price),
    geom = "point", stat = "identity", position = "identity"
) +
scale_y_continuous() +
scale_x_continuous() +
coord_cartesian()
```

We start with `ggplot()` which creates a new plot object, and then add the other components: a single layer, specifying the data, mapping, geom and stat, the two continuous position scales and a Cartesian coordinate system. The layer is the most complicated and specifies that we want to:

- use the diamonds dataset,
- map carat to horizontal (x) position, and price to vertical (y) position, and
- display the raw data (the identity transformation) with points.

Intelligent defaults allow us to simplify this specification in a number of ways. First, we only need specify one of geom and stat, as each geom has a default stat (and vice versa). Second, the Cartesian coordinate system is used by default, so does not need to be specified. Third, default scales will be added according to the aesthetic and type of variable. For position, continuous values are transformed with a linear scaling, and categorical values are mapped to the integers; for colour, continuous variables are mapped to a smooth path in the HCL colour space, and discrete variables to evenly spaced hues with equal luminance and chroma. The choice of a good default scale is difficult, and is touched on in Section 3.7.

This leads us to the following specification:

```
ggplot() +
layer(
  data = diamonds, mapping = aes(x = carat, y = price),
  geom = "point"
)
```

Typically, we will specify a default dataset and mapping in the `ggplot` call, and use a shorthand for the layer:

```
ggplot(diamonds, aes(x = carat, y = price)) +
geom_point()
```

Any aesthetics specified in the layer will override the defaults. Similarly, if a dataset is specified in the layer, it will override the plot default. To get to the second plot of Figure 3.10 we need to add another layer, and override the default linear-transforms with log-transformations:

```
ggplot(diamonds, aes(x = carat, y = price)) +
geom_point() +
stat_smooth(method = lm) +
scale_x_log10() +
scale_y_log10()
```

which is short hand for:

```
ggplot() +
layer(
 data = diamonds, mapping = aes(x = carat, y = price),
 geom = "point", stat = "identity", position = "identity"
) +
layer(
 data = diamonds, mapping = aes(x = carat, y = price),
 geom = "smooth", position = "identity",
 stat = "smooth", method = lm
) +
scale_y_log10() +
scale_x_log10() +
coord_cartesian
```

Even with these many defaults, the explicit grammar syntax is rather verbose, and usually spans multiple lines. This makes it difficult to rapidly experiment with different plots, very important when searching for revealing graphics. For this reason the verbose grammar is supplemented with `qplot`, short for quick plot, which makes strong assumptions to reduce the amount of typing needed. It also mimics the syntax of the `plot` function, making ggplot2 easier to use for people already familiar with base R graphics.

The `qplot` function assumes that multiple layers will use the same data and aesthetic mappings, and defaults to creating a scatterplot. This allows us to recreate the first plot with this concise code:

```
qplot(carat, price, data = diamonds)
```

The second plot is not much longer:

```
qplot(carat, price, data = diamonds,
  geom = c("point", "smooth"),
  method = "lm", log = "xy"
)
```

Note the use of the `log` argument, which mimics the `plot` function in base and specifies which axes should be log-transformed. The geom argument can take a vector of geoms, which are added sequentially to the plot. Everything else is taken to be a layer parameter. The limitations to this approach are obvious: which layer does the method argument apply to?

3.5 An embedded grammar

The previous section hints at an important point: the specification of the layered grammar is tightly embedded within an existing programming language, R. This section discusses some of advantages and disadvantages of such an approach. This discussion is centred around R, but many of the issues apply regardless of language. To make this section concrete Table 3.8 shows the syntax of GPL and ggplot2 used to create Figure 3.11, which has been adapted from Wilkinson (2005, Figure 1.5, page 13).

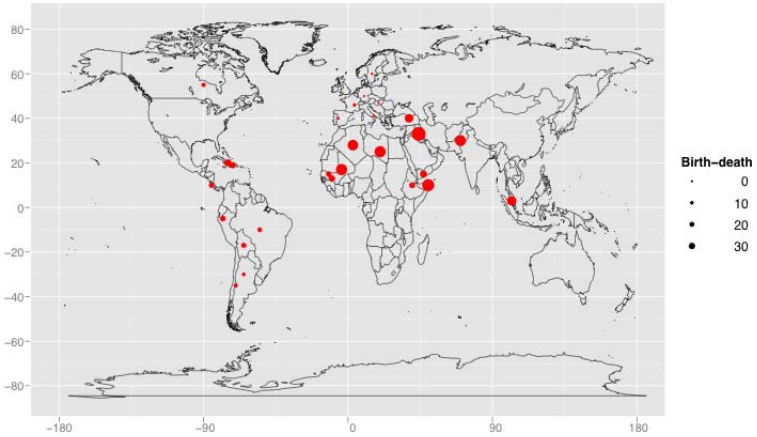


Figure 3.11: Plot of birth rate minus death rate in selected countries.

The advantages of embedding a graphical grammar into another programming language are obvious: one immediately gains all of the existing capabilities of that language. We can use all the facilities of a full programming language to automate repetitive tasks: loops to iterate over variables or subsets, variables to store commonly used components, and functions to encapsulate common tasks. For example, ggplot2 provides a set of plot templates, which are functions that generate plot specifications for more complicated graphics, like the parallel coordinates plot and scatterplot matrix.

```
DATA: source("demographics")|
DATA: long, lati = map(source("World"))
TRANS: bd = max(birth - death, 0)
ELEMENT: point(position(lon * lat), size(bd), color(color.red))
ELEMENT: polygon(position(long * lati))
COORD: project.mercator()
```

```
demographics <- transform(demographics,
  bd = max(birth - death, 0))

ggplot(demographics, aes(x = lon, y = lat))
+ geom_point(aes(size = bd), colour="red")
+ geom_polygon(data = world)
+ coord_map(projection = "mercator")
```

Table 3.8: Specification of Figure 3.11 in GPL (top) and ggplot2 (bottom) syntax.

With R in particular, we gain a wide range of pre-packaged statistics useful for graphics: loess smoothing, quantile regression, density estimation, etc. Additionally, we can also use the host's facilities for data import and manipulation. In order to have data to plot, any graphics tool must have at least rudimentary tools for importing, restructuring, transforming and aggregating data. By relying on other tools in R, ggplot2 does not need three elements of Wilkinson's grammar: the DATA component, the TRANS component and the algebra.

The DATA component is no longer needed as data is stored as R data frames; it does not need to be described as part of the graphic. The TRANS stage can be dropped because variable transformations are already so easy in R; they do not need to be part of the grammar. The example uses the `transform` function, but there are a number of other approaches. The algebra describes how to reshape data for display and, in R, is replaced by the `reshape` package (Wickham, 2005). Separating data manipulation from visualisation allows you to check the data, and the same restructuring can be used in multiple plots. Additionally, the algebra can not easily perform aggregation or subsetting, but `reshape` can.

The disadvantages of embedding the grammar are somewhat more subtle, and centre around the grammatical restrictions applied by the host language. One of the most important features of the grammar is its declarative nature. To preserve this nature in R, ggplot2 uses $+$ to create a plot by adding pieces together. The `ggplot` function creates a base object, to which everything else is added. This base object is not necessary in a stand-alone grammar.

3.6 Implications of the layered grammar

What are some of the implications of defining a graphic as described above? What is now easy that was hard? This section discusses three interesting aspects of the grammar:

- The histogram, which maps bar height to a variable not in the original dataset, and raises questions of parameterisation and defaults.
- Polar coordinates, which generate pie charts from bar charts.
- Variable transformations, and the three places in which they can occur.

3.6.1 Histograms

One of the most useful plots for looking at 1D distributions is the histogram, as shown in Figure 3.12. The histogram is rather special as it maps an aesthetic (bar height) to a variable created by the statistic (bin count), and raises some issues regarding parameterisation and choice of defaults.

The histogram is a combination of a binning stat and a bar geom, and could be written as:

```
ggplot(data = diamonds, mapping = aes(x = price))  +
layer(geom = "bar", stat = "bin", mapping = aes(y = ..count..))
```

One interesting thing about this definition is that it does not contain the word histogram: a histogram is not a primitive, but is a combination of bars and binning. However, we do want to create histograms, without having to specify their composition, and so we add the histogram geom as an alias of the bar geom, with a default bin stat, and a default mapping of bar-height to bin count. This lets us produce the same plot in either of the following two more succinct ways:

```
ggplot(diamonds, aes(x = price)) + geom_histogram()
qplot(price, data=diamonds, geom="histogram")
```

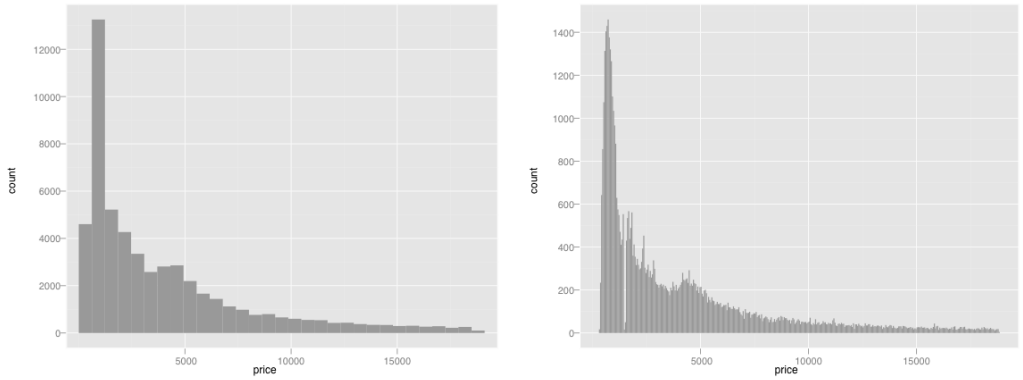


Figure 3.12: Two histograms of diamond price produced by the histogram geom. (Left) Default bin width, 30 bins. (Right) Custom $50 bin width reveals missing data.

The choice of bins is very important for histograms. Figure 3.12 illustrates the difference changing bin widths can make. In ggplot2, a very simple heuristic is used for the default number of bins: it uses 30, regardless of the data. This is perverse, and ignores all of the

research on selecting good bin sizes automatically, but sends a clear message to the user that they need to think about and experiment with the bin width. The histogram geom facilitates this experimentation by being parameterised by bin width (as opposed to number of bins, as is common elsewhere in R). This is preferable as it is directly interpretable on the graph. If you need more control you can use an alternative specification: the breaks argument lists exactly where bin breaks should occur.

The separation of statistic and geom enforced by the grammar allows us to produce variations on the histogram, as shown in Figure 3.13. Using a ribbon instead of bars produces a frequency polygon, and using points produces a graphic that doesn't have its own name.

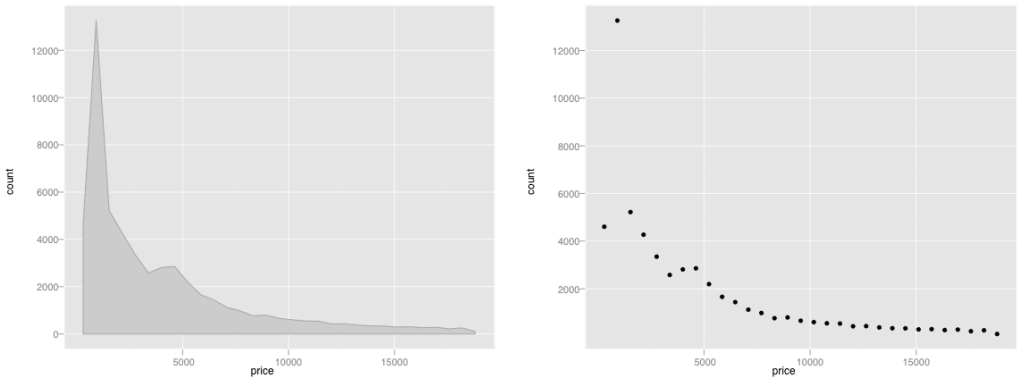


Figure 3.13: Variations on the histogram. Using a ribbon (left) to produce a frequency polygon, or points (right) to produce an unnamed graphic.

The histogram is interesting for another reason: one of the aesthetics, the y-position, is not present in the original data, but is the count computed from the binning statistic. In the ggplot2 this is specified as `aes(y = ..count..)`. The two dots are a visual indicator highlighting that variable is not present in the original data, but has been computed by the statistic. There are other variables produced by the binning statistic that we might want to use instead, e.g. `aes(y = ..density..)` or `aes(y = ..density../sum(..density..))`. A less common mapping is `aes(y = ..count.. / max(..count..))`, which creates a histogram with a vertical range of $[0, 1]$, useful for faceted and interactive graphics.

The histogram also knows how to use the `weight` aesthetic. This aesthetic does not directly affect the display of the data, but will affect stat: instead of counting the number of observations in each bin, it will sum the weights in each bar.

3.6.2 Polar coordinates

In ggplot2, the user is offered the choice of a number of coordinate systems. One coordinate system that is used very commonly in business graphics is the polar coordinate system, used to produce pie charts and radar plots. The polar coordinate system parameterises the two-dimensional plane in terms of angle, θ, and distance from the origin, or radius, r. We can convert to regular Cartesian coordinates using the following equations:

$$
\begin{aligned}
x &= r \cdot cos(\theta) \\
y &= r \cdot sin(\theta)
\end{aligned}
$$

As with regular Cartesian coordinates, we can choose which variable is mapped to angle and which to radius. It can also be useful to tweak the range of the radius: having a minimum radius that is greater than zero can be useful because it avoids some of the compression effects around the origin.

The angle component is particularly useful for cyclic data because the starting and ending points of a single cycle are adjacent. Common cyclical variables are components of dates, for example, days of the year, or hours of the day, and angles, e.g. of wind direction. When not plotted on polar coordinates, particular care needs to be taken with cyclic data to ensure that we don't miss patterns occurring across the arbitrary start and end points.

In the grammar, a pie chart is a stacked bar geom drawn in a polar coordinate system. Figure 3.14 shows this, as well as a bullseye plot, which arises when we map the height to radius instead of angle. A regular bar chart converted into polar coordinates produces another type of graphic: the coxcomb plot, popularised by Florence Nightingale (Nightingale, 1857). A comparison between a bar chart and Coxcomb chart is shown in Figure 3.15.

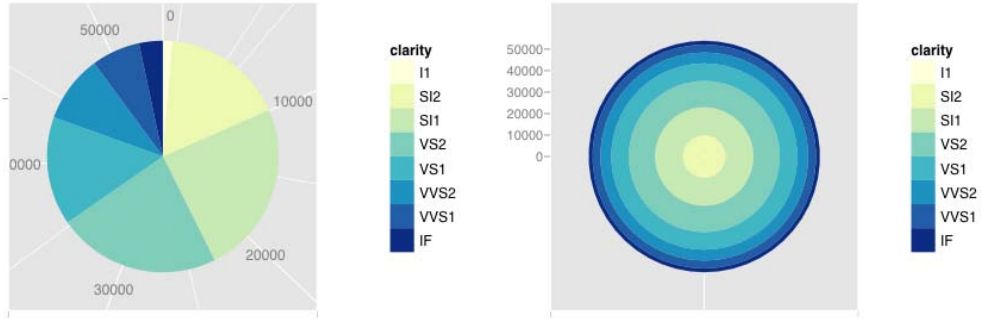


Figure 3.14: Pie chart (left) and bullseye chart (right) showing the distribution of diamonds across clarity (I1 is worst, IF is best). A radial chart is the polar equivalent of the spineplot: in the pie chart, categories have equal radius and variable angle; in the radial chart, categories have equal angle and variable radius.

3.6.3 Transformations

There are three ways to transform values in ggplot2: by transforming the data, by transforming the scales and by transforming the coordinate system. Transforming the data or the scales produces graphs that look very similar, but the axes (and grid lines) are different: everything else remains the same. This is because statistics operate on data that has been transformed by the scale. Figure 3.16 shows an example of this using a scatterplot with a smoothed fit.

Transforming the coordinate system does something quite different, as shown in Figure 3.17. The coordinate transformation occurs at the very end of the plotting process,

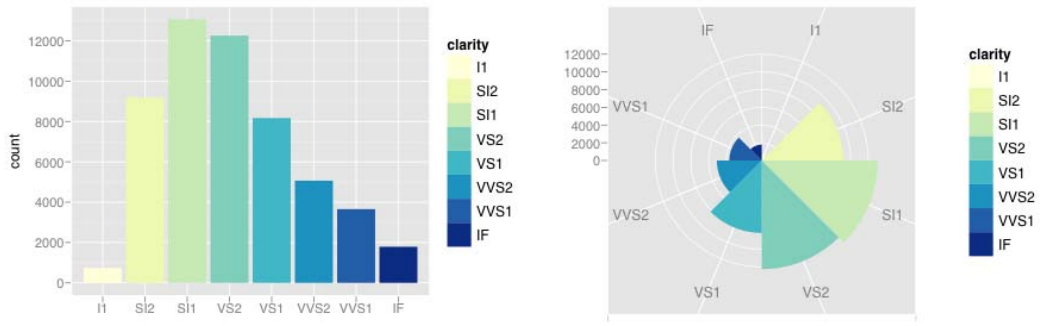


Figure 3.15: Bar chart (left) and equivalent Coxcomb plot (right) of clarity distribution. The Coxcomb plot is a bar chart in polar coordinates. Note that the categories abut in the Coxcomb, but are separated in the bar chart: this is an example of a graphical convention that differs in different coordinate systems.

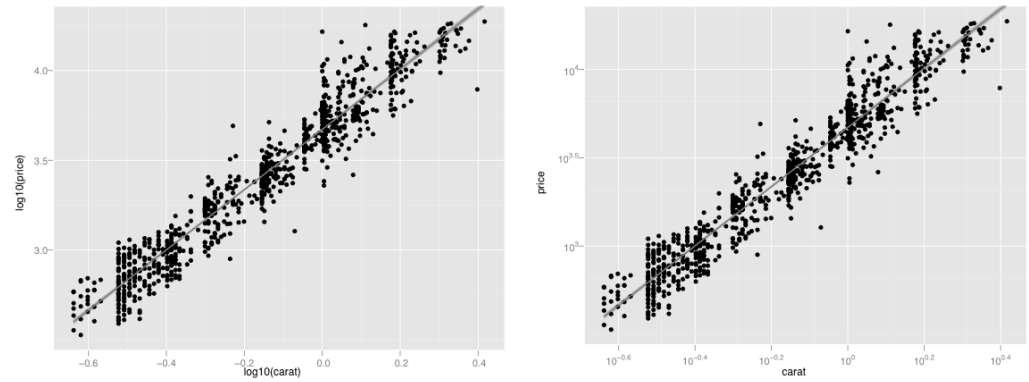


Figure 3.16: Transforming the data (left) vs transforming the scale (right). From a distance the plots look identical. Close inspection is required to see that the scales and minor grid lines are different. Transforming the scale is to be preferred as the axes are labelled according to the original data, and so are easier to interpret. The presentation of the labels still requires work.

and alters the appearance of geoms. Here, the smooth is performed on the raw data, producing a straight line, but is then stretched into the new logarithmic coordinate system.

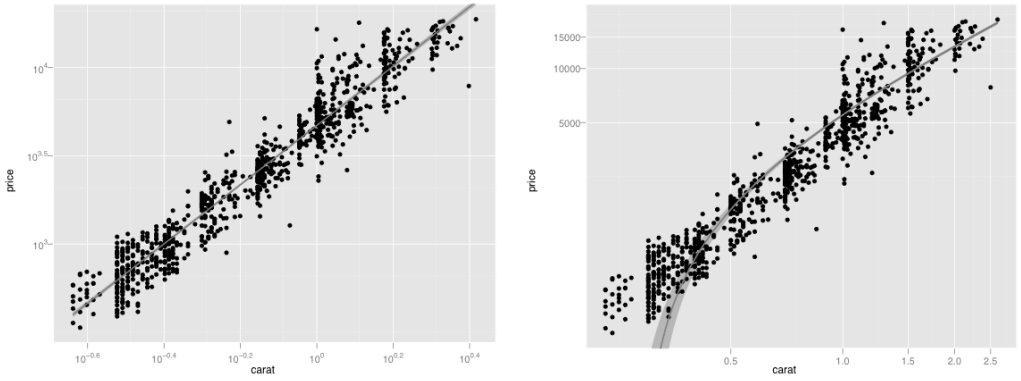


Figure 3.17: Transforming the scales (left) vs transforming the coordinate system (right). Coordinate system transformations are the final step in drawing the graphic, and affect the shape of geometric objects. Here a straight line becomes a curve.

We can also use the scales and coordinate system together to display a smooth calculated on the logged data and then back-transform it to the original scale. This is shown in Figure 3.18. This is equivalent to fitting the model $log(y) \sim log(x)$ and then back-transforming predicted values.

Currently the placement of the tick marks are sub-optimal, as raw vs transformation and back-transformation generate different tick positions when arguably they should be the same. This would mean that the scales in Figure 3.17 should be the same. This has the drawback of concealing whether the transformation was applied to the scale or the coordinate system. The nature of the transformation would need to be indicated in some other way, either in the caption of the graphic or elsewhere on the plot. This raises an interesting philosophical question: should we be able to uniquely identify the specification of a graphic from its rendering?

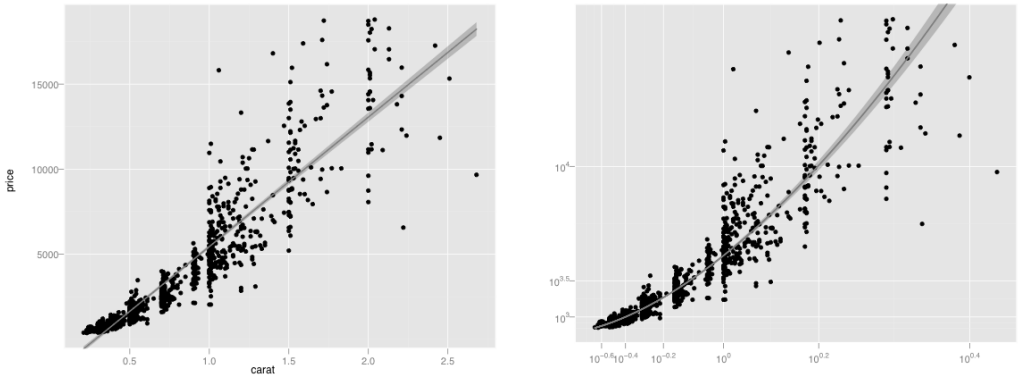


Figure 3.18: Linear model fit to raw data (left) vs linear model fit to logged data, then back-transformed to original scale (right).

3.7 Perceptual issues

The grammar describes the components of the plot, but not their appearance. To create appealing, useful graphics it is necessary to devote considerable effort to these "minor" construction details, bearing in mind the cognitive psychology of perception. Difficulty arises because many of these issues are subjective: they may look nice to me, but look horrid to you, and there is little research available to guide these design choices. The points below illustrate some of the decisions which we have made, and some of the issues that we have struggled with.

- Grey background and white grid lines. This follows from the advice of Tufte (1990, 1997, 2001, 2006) and Brewer (1994a); Carr (1994, 2002); Carr and Sun (1999). We can still see the gridlines to aid in the judgement of position (Cleveland, 1993), but they have little visual impact and we can easily "tune" them out. The grey background gives the plot a similar colour (in a typographical sense) to the remainder of the text, ensuring that the graphics fit in with the flow of a text without jumping out with a bright white background. Finally, the grey background creates a continuous field of colour which ensures that the plot is perceived as a single visual entity.

- Data points are not plotted next to the margins of the plot. This is an old, but important, guideline from Cleveland and McGill (1987). Data points need to be placed some distance from the axis so that the components of the axis and the data are not confused. In ggplot2 there are different rules depending on whether the axis is continuous or discrete. For continuous data, 5% extra space is added, and for discrete data half a bar is added. These can be fine tuned by the user if necessary.

- Placing tick marks on axes is difficult, and currently ggplot2 uses the `pretty` function in base R. This is original work by Martin Maechler (personal comm.), which has not been published. I also experimented with the heuristics discussed in Wilkinson (2005), but to my eye these produced poor results. Other methods are discussed in Murdoch (2000); Nelder (1976); Sparks (1971); Stirling (1981); Thayer and Storer (1969). There has been little recent work on "optimal" tick mark placement, especially for non-linear scales. Minor grid lines are also displayed, by default, one between each major tick mark. This helps judge if scale is non-linear.

  A wide range of scales are implemented in ggplot2 (any function with an inverse), so a rule was needed for placing minor tick marks. Following the typical display of a log-log plot, major tick marks are evenly spaced on the plot, while minor tick marks are evenly spaced in the original data space.

- The direct representation of the data, the geom, is the most important. While scales and coordinate systems are important in that we need to be able to trust them, and may sometimes want to read off values, typically we will use other tools to get exact values (eg. interactive identification, or reading off table of values), and we are largely interested in the gestalt. In ggplot2, we spent most ink on the geom, not on the guides.

- The colour of default colour scales is difficult. Initially ColorBrewer (Brewer, 1994a,b) palettes were used. Unfortunately, they have a major drawback in that they are designed for maps, where the coloured areas are large and contiguous. For scatterplots, where the colour regions are small and separate, the colours are not very distinguishable. In general, smaller areas require brighter, more saturated colours. There is a perceptual interaction between the colour and the size of the region that is coloured (Ihaka, 2003) that is difficult to model and adjust for.

  Currently, different scales are used for categorical or continuous variables. For continuous data, a colour scale generated by linearly interpolating between two colours in HCL space (eg. blue and yellow). Categorical variables are mapped to evenly spaced hues, with equal luminance and chroma. These choices are good, but not great: the categorical colours can look muddy and unattractive, and print to black and white as indistinguishable shades of grey; and much care needs to be taken when choosing starting colours for the continuous scale.

3.8 A poetry of graphics?

The grammar tells us what words make up our graphical "sentences", but offers no advice on how to write well. How can we build on top of the grammar to help data analysts build compelling, revealing graphics? What would a poetry of graphics look like? The ideas in this section draw inspiration from the tools that word processors provide to try and help us write better.

With ggplot2, we already have the equivalent of a spelling checker: the plot simply won't work unless the components have been correctly specified. How about a grammar checker? This tool would identify common mistakes and warn the user. This is a helpful first step: a grammar checker ensures that you haven't made any obvious mistakes, but will not significantly improve your prose, and sometimes the warnings may be incorrect. For graphics, some readily detected problems and possible solutions are:

- Too many variables. It is hard to see the relationship between more than three variables in a single panel: two position and one other. We should warn the user when too many aesthetics are used, and suggest alternatives, such as faceting.

- Overplotting. It is easy to draw incorrect conclusions about the distribution of points in the presence of overplotting. There are many possible solutions. We could supplement the plot with the contours of a 2d density estimate, or colour each point by the local density (Unwin et al., 1996). Other solutions are suggested in Carr et al. (1987); Cleveland and McGill (1984); Huang et al. (1997).

- Alphabetical ordering. Categorical variables are often displayed in alphabetical ordering. Wainer (2000, 2004) calls this mistake "Alabama first!" due to the propensity for states to be listed in alphabetical order. Ordering by some property of the data (e.g. mean or median) often makes the plot more useful (Becker et al., 1996).

- Polar coordinates. We know that humans are better at judging length than angle or area (Cleveland and McGill, 1987), and polar coordinates have a singularity at the

origin which makes it difficult to judge angle for objects with small radius. Should we always warn users when they choose to use this coordinate system?

We can also provide tools to help the analyst get started; instead of a blank page, we could start with a template, containing some potentially useful views. One approach is to calculate a measure of interest for a wide number of possible plots and then show the most interesting. This is the basic idea behind scagnostics (Wilkinson et al., 2005) and projection pursuit (Cook et al., 2008b; Friedman, 1987), which explore the space of scatterplots generated by projection. We can also explore other parameter spaces, as the aspect ratio of the plot, as in Heer and Agrawala (2006), which create multiple plots each banking a different component to 45° (as recommended by Cleveland (1993)). Similarly, instead of displaying the histogram with the optimal bin width, we could display the five or ten most revealing.

Beyond that, it seems difficult to see how to do much more algorithmically, and we need to turn to education and training. The grammar provides a good foundation for teaching, as it helps students to see the deeper themes underlying different graphics, as well as providing a tool for describing and drawing graphics.

3.9 Conclusions

The aim of the grammar is to "bring together in a coherent way things that previously appeared unrelated and which also will provide a basis for dealing systematically with new situations" (Cox, 1978). How well have we succeeded?

With the histogram, we saw how a familiar graphic can be broken into its component parts, and then those parts changed independently to get variations on an old friend. In the polar coordinates examples, we saw the deep similarities that underlie the bar chart, pie chart and Coxcomb chart. We also accidentally created an unusual chart, the bullseye plot, and saw how it is like the polar equivalent of the spineplot. The transformation examples showed the usefulness of multiple transformation stages, and how we can mimic the common statistical practice of back-transformation using graphics alone.

One area where this grammar is not so strong is area plots: bar charts, spineplots, histograms, mosaic plots, etc. (Hartigan and Kleiner, 1981; Hofmann, 2003). While they can be constructed with the grammar, it gives no great insight into their underlying structure, or how we can create new, useful, variations. This suggests that it may be fruitful to describe "sub-grammars" for particular families of plots.

The layered grammar does not include user interaction with plot; all plots are static and separate. Clearly there is huge scope for adding interaction to this grammar. Some types of interaction will be easier to add than others. For example connecting a slider to the bin width of a histogram. Here we connect some user interface element to some specification of the plot. This type of interaction includes zooming and panning (changing scale limits), but not linked brushing. Linked brushing is an example of interaction with the underlying data, and incorporating this type of interaction into the grammar will require deeper thought. Speed is also a challenge; to be seamlessly perceived an interactive graphic must be updated multiple times a second, whereas many ggplot2 graphics take over a second to draw.

The grammar is powerful and useful, but not all encompassing. As well as specialising the grammar for particular families of graphics, we also need to support the creation of attractive, useful plots. The ideas outlined in the last section suggest ways to build on top of the grammar to support graphical data analysis and exploration.

ggplot2 is available from http://had.co.nz/ggplot2.

3.10 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0706949.

Chapter 4

Visualising statistical models Removing the blindfold

Abstract

This paper discusses strategies for visualising statistical models. It is organised around three strategies: display the model in the data space; look all members of a collection; and explore the process of model fitting, not just the end result. Case studies develop visualisations for MANOVA, classification algorithms, hierarchical clustering, ensembles of linear models, projection pursuit, self organising maps and neural networks.

4.1 Introduction

There are very many well-known techniques for visualising statistical data, but far fewer for statistical models. This paper pulls together some general techniques for visualising statistical models, stressing the importance of displaying the model in the context of the data. Why do we need to be able to visualise statistical models? What should a good visualisation of a model help us to do? There are three questions that graphics are particularly well suited to answering:

- What is the model? While we always know the general form of the model, inspection of the numerical parameters may give no insight into the details of its structure. For example, looking at the estimates for natural spline terms in a linear model will not reveal the shape of the curve. A graphical summary of the model will often be more interpretable, particularly to non-statisticians.

- How well does the model fit the data? Graphics allow us to compare the shape of the model to the data, giving us diagnostics that not only indicate if something is wrong, but also suggest ways to fix the problem.

- How can we do better? Graphics also provide us with inspiration. The form of a model is fixed and it is unlikely to surprise us. A linear model will never reveal a non-linear relationship, or that there are clusters in the data. If we know exactly what we are looking for, then numerical methods are provably most efficient. But how often does that occur?

Developing good visualisations for models requires some general strategies. This paper is organised around three of them:

- Display the model in data space (not data in the model space). We should strive to display the essence of model in the high-d data space, rather low-d summaries of the data produced by the model.
- Look at all members of a collection, not just one. Multiple models will often give more insight than a single model; looking at many local optima may give more insight than look at a single global optimum; looking at multiple summaries statistics is more informative than looking at one.
- Explore the process of fitting, not just the end result. Understanding how the algorithm works allows us to better understand how the data contributes to the final model.

Each strategy is described in detail in the following sections, and is illustrated by one or more case studies. The paper concludes with a larger case study that develops informative new visualisations for neural networks, using all of the strategies discussed in the paper.

Many of these techniques have grown out of our work with `rggobi` (Temple Lang et al., 2007), an R package which connects the statistical programming environment of R (**?**) with the interactive and dynamic graphics of GGobi (Swayne et al., 2003). This paper ties together the ideas underlying three R packages that use `rggobi`: `classifly` (Wickham, 2007a), `clusterfly` (Wickham, 2007b), and `meifly` (Wickham, 2006). The importance of a tight connection between statistical and graphical environments has long been recognised, particularly in Lisp-Stat (Tierney, 1990), the ancestors of `rggobi` (Swayne et al., 1991), Datadesk (Velleman, 1992), ViSta (Young et al., 1993), ARC (Cook, 1998), Mondrian (Theus, 2003) and iplots (Urbanek and Wichtrey, 2007). A recent trend has been to use R as the statistical computation engine and develop interactive graphics in another language. This is the approach of Mondrian, iplots, rggobi, Klimt (Urbanek, 2002b) and Gauguin (Gribov, 2007).

The techniques described in this paper make extensive use of dynamic and interactive graphics. To supplement the static snapshots of these tools, we have provided a number of movies. Each movie is hyperlinked from the section it is used, and a complete listing is available at `http://www.vimeo.com/album/8875`. These movies should work on any modern browser, but a broadband internet connection is recommended. As well as interactive graphics, good static graphics play an important role, especially when communicating results in papers and presentations. This paper uses the `ggplot2` package (Wickham, 2008) extensively to rapidly go from graphics in our heads to graphics on the page. The majority of graphics in this paper were produced with a few lines of code. This is a considerable help when exploring a wide range of possible representations: the less time it takes to produce a single representation, the more time you have to try other options.

One dataset (Asuncion and Newman, 2007) is used throughout this paper. It contains information on 178 wines made with grapes from 3 varieties, and with 13 variables describing physical composition (including chemical constituents and colour). As the variables are not directly comparable, we have standardised them to range $[0, 1]$. In addition to this

dataset, there several simpler simulated datasets are used to focus attention on specific points.

4.2 What is a model? Terminology and definitions

Before we continue, it's worth discussing in more detail what exactly a model is, since "model" is such an overloaded term. We use it to refer to different levels of specificity: model family, model form and fitted model, as described below.

- The model **family** describes, at the broadest possible level, the connection between the variables of interest. For example, the linear model predicts a single continuous response to a linear combination of predictor variables, and assumes normally distributed error, $Y|X \sim \text{Normal}(\mu = AX, \sigma)$. The model family is almost always specified by the statistician (i.e. you can't use a test to determine whether you should use a linear model or model-based clustering).

- The model **form** specifies exactly how the variables of interest are connected within the framework of the model family. For example, the model form of a linear model specifies which variable is the response, and which variables are the predictors, e.g. $height|age, sex \sim \text{Normal}(\mu = b + a_1 \cdot age + a_2 \cdot sex, \sigma)$. The model form might be specified a priori by the statistician, or chosen by a model-selection procedure.

- The **fitted** model is a concrete instance of the model form where all parameters have been estimated from data, and the model can be used to generate predictions. For example, $height|age, sex \sim \text{Normal}(\mu = 10 + 5 \cdot age + 3 \cdot sex, \sigma = 4)$. Typically, the fitted model optimises some criterion for the given data, e.g. the linear model minimises the total squared deviation between responses and predictions.

Bayesian models can be described in a similar manner, although there we tend to focus on the the distributions of the parameters given the data, rather than a single estimated value.

Not all techniques used by statisticians have these explicit probabilistic models, but the basic breakdown remains the same. For example, neural networks are a model family, the model form specifies the variables and number of nodes, and the fitted model will have estimates for the parameters of each node. Neural networks are a good example of where going from the model form to the fitted model is difficult. We must use numerical methods, and convergence to a global optimum is not guaranteed. This means to find the "best" fitted model (i.e. the model with the highest criterion value) we need multiple random starts. We will come back to this in Section 4.6.

Knowing the model family and form does not guarantee we know what the fitted model looks like. For example, the parameters of the linear model can be interpreted as the change in the response when the predictor is changed by one unit, if all other variables are held constant. This seems easy to interpret, but most models have interdependencies between the predictors which means variables do not change independently, as with interactions or polynomial terms. For more complex models, there may never be any direct interpretation of any of the model parameters.

4.3 Display the model in data-space

A major theme of this paper is the importance of visualising the model in the context of the data, displaying the model in the high-dimensional data space, or for short, showing the **m-in-ds**. This is important because it allows us see how well (or how poorly) the model responds to the data. The opposite of visualising the model in data space is visualising the data in the model space (**d-in-ms**), where the data is displayed in some low-dimensional space generated by the model. Both methods have their uses, but m-in-ds displays are particularly helpful for understanding and critiquing the model.

To compare these approaches, think of a linear model. One of the most common diagnostics is a plot of fitted values versus residuals. This is a d-in-ms display because we are plotting summary statistics from the model in a low-d view. An m-in-ds approach would be to generate the full regression surface, and visualise it in the context of the data; this is hard, but often revealing. For an m-in-ds approach, we can still use the residuals, but we would want to see them in the context of the entire predictor space. For categorical predictors, another approach would be to use population marginal means (Searle et al., 1980) to summarise the categorical coefficients in such way that they are interpretable in the data space.

More generally, what will a model embedded in the data space look like? We define the data space to be the region over which we can reliably make inferences, usually a hypercube containing the data, unless the variables are highly dependent, in which case the data space might be a small subspace of the full cube. We can say a few things about the form of models in general because almost all models share two important characteristics: they summarise the data and interpolate the data space. This means that the model will have many fewer dimensions than the data, and it will take values over the entire data space, even where there are no data. This implies that models are manifolds, or high-d surfaces.

It will be challenging to visualise the model in the data space: we have high-d data points and a high-d model manifold. The following subsections describe some general tools for high-d visualisation, and develop some ideas for representing the model in the data space. Finally, three case studies illustrate the m-in-ds strategy for MANOVA, classification algorithms and hierarchical clustering.

4.3.1 Tools for visualising high-d data and models

To visualise data and models in many dimensions, we need some good tools. While it is possible to investigate high-d objects with only static graphics, it is much easier to do so interactively. We will use two basic tools extensively: the grand tour, to look at many projections of the data; and linked brushing, to interactively conditioning the data and look at many subsections of the data. These tools are described briefly below, and in more detail in Cook and Swayne (2007). Other tools, such as the parallel coordinate plot (Inselberg, 1985; Wegman, 1990), are also useful, but we prefer the tour because for low to moderate-d it gives a more immediately interpretable view of the data.

When we have a 3d object, how do we figure out what shape it is? We pick it up and look at it from multiple sides, joining a sequence of 2d views into a 3d model. The grand tour generalises this idea to more than n-d, generating a sequence of 2d projections of an n-d

object. It chooses new projections randomly and smoothly rotates between them. The new projections are selected at random so that if we watch the tour long enough we'll see every possible view, but we'll never get stuck in the same uninteresting region for long. Among other things, the grand tour can be used to identify multivariate outliers and clusters in the data, as the snapshots in Figure 4.1 illustrate. A variant of the grand tour is the guided tour, where new projections are selected according to an index of interestingness, and is described in Section 4.5.1.

The limitation of the grand tour is that it is only useful for continuous variables. Conditioning is useful for exploring a few categorical variables, but if the data contains many categorical variables, it will be necessary to use more specialised tools such as the mosiac plot (Hartigan and Kleiner, 1981; Hofmann, 2000).

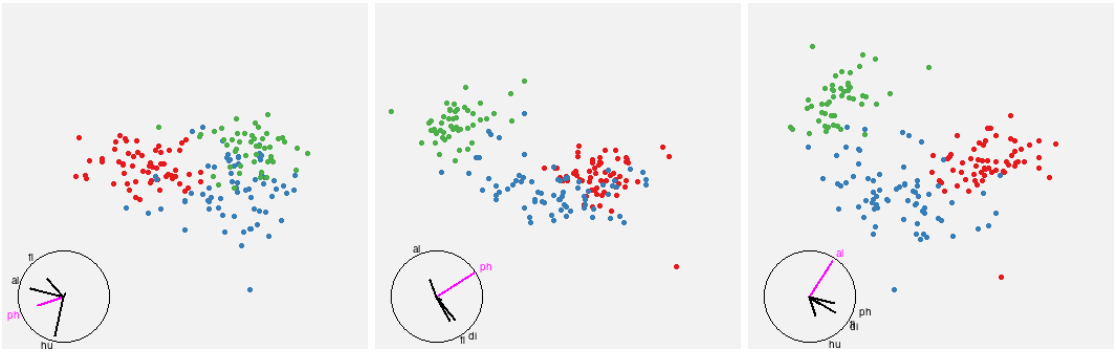


Figure 4.1: Three interesting projections from the grand tour of the wine data, illustrating (from left to right) an outlying blue point, an outlying red point, and that the groups are fairly distinct. Animated version available at http://vimeo.com/823263.

Brushing is an interactive conditioning tool, analogous to trellis plots, but much more flexible. Brushing links multiple views together by styling observations consistently across views. We use a brush to colour observations in one plot and see the same observations change colour in other views. Figure 4.2 shows one use of the brush, as a 1d conditioning tool. We sweep a long and narrow brush across one plot and observe where that slice falls in the context of another plot. As in the figure, we will often need to use a combination of tours and linked brushing as the combination of projections and sections can be more revealing than either one alone (Furnas and Buja, 1994).

4.3.2 Representing models as data

Representing the model as data is challenging and requires some creativity and imagination. You can draw inspiration from existing d-in-ms plots, but otherwise coming up with a good representation requires some thinking and experimentation. Here we describe some techniques that we have found useful.

For predictive models with a continuous response we want to be able to show the prediction surface, $z = f(a, b, c, ...)$. For complex models, it's often difficult to describe this surface parametrically, and we have few tools to display high-d surfaces, so we suggest using an approximation. The simplest method is to generate a dense sample of points lying on the surface. One way to do this is to sample from the predictor space and compute a

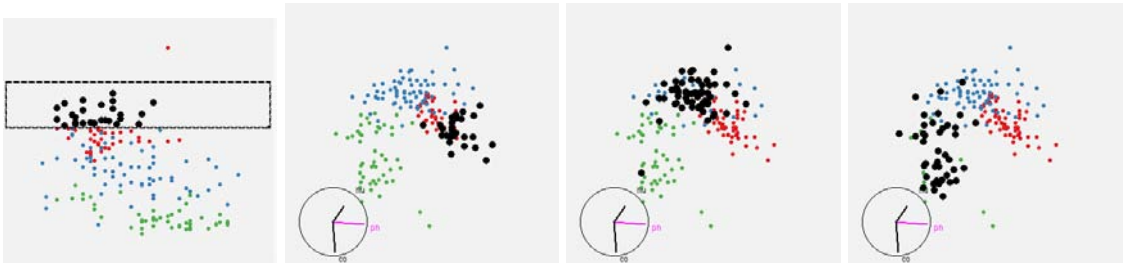


Figure 4.2: (Far left) The active window containing the brush, and (right) the corresponding points brushed in another view, with three snapshots as we move the brush down. Selected points are enlarged and coloured black. Here brushing shows us that the data lies in a spiral in 3d: as we move around the 2d curve from mid-right to bottom-left we move from high to low values of the conditioned variable. Animated version available at http://vimeo.com/823262.

prediction for each sample. Borrowing from the terminology of spatial statistics, we can sample the predictor space in a stratified, random or non-aligned manner. Non-aligned sampling tends to work best: stratified sampling creates a grid that can produce distracting visual artefacts, and uniform sampling generally needs many more points to create the illusion of a continuous surface.

If we want to display more information about the predictive distribution than its expected value, what can we do? One approach is to also display other quantities from the distribution. For example, with a linear model, we might display the mean and the 5% and 95% quantiles. This is straightforward when the predictor only enters the distribution through a single parameter (e.g. the mean), but maybe complex if it enters through multiple parameters. In that case, an alternative is to draw random samples from the predictive distribution (Gelman and Hill, 2006).

For models that incorporate connections between components it can be useful to display these connections explicitly with lines. This technique is used for hierarchical clustering in Section 4.3.5 and for self organising maps in Section 4.5.2. This approach is useful for any model with a neighbourhood structure.

The case studies highlight some of these different techniques: for MANOVA we display a quantile from the predictive distribution; for classification algorithms we sample from the data space and display prediction regions; and for hierarchical clustering we draw inspiration from 2d plots and make explicit the implicit connections.

4.3.3 Case study: MANOVA

MANOVA is a multivariate extension of ANOVA where there are multiple continuous responses instead of just one. Correspondingly, the error distribution of MANOVA is the multivariate normal, with mean 0 and variance-covariance matrix Σ. Because MANOVA looks at response variables simultaneously it can identify differences that individual ANOVAs cannot, as shown in Figure 4.3.

This case study explores adding MANOVA model information to the raw data. We will investigate a simple MANOVA with a single categorical explanatory variable, and discuss how we might extend the ideas the more general multivariate linear model. In this simple case, the model is that each group is multivariate normal with the same variance-covariance

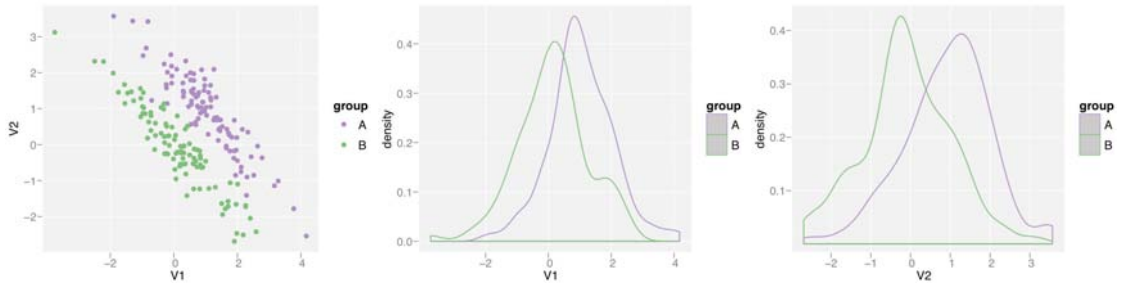


Figure 4.3: Example where MANOVA would detect a difference between the groups, but two ANOVAs would not. Groups are distinct in 2d (left), but overlap on both margins (right).

matrix. To summarise this model we'll display a 84% confidence region around the mean. 84% corresponds to 1.4 standard errors of the mean, used because the standard error of the difference is 1.4× ($\sqrt{2}$) that of standard error of the mean. If two 84% confidence regions don't overlap, then the means must be at least 2 standard errors apart and thus significantly different at the 5% level.

Generating the points on the surface of the confidence region is straightforward. First, we draw points from a d-dimensional standard multivariate normal, then project these onto the surface of a sphere by normalising each row to distance 1. Next, we skew this sphere to match the desired variance-covariance matrix, blow it up to give the appropriate cl-level confidence ellipsoid, and translate it by the group mean. This generates a p-dimensional ellipsoid, which in 2d projections will look like a filled ellipse. Figure 4.4 shows the result of this approach applied to the wine data.

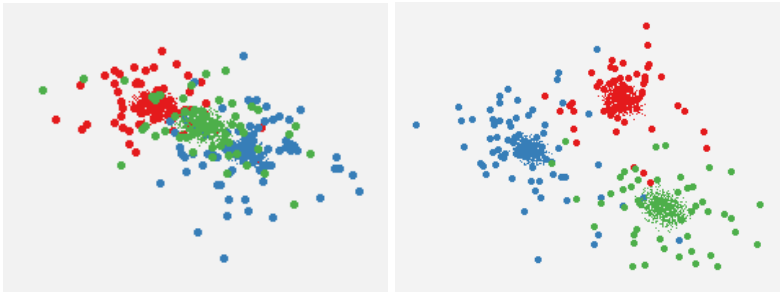


Figure 4.4: Two projections of the wine data with 84% confidence regions around the group means. Large points are data, small points are a sample on the surface of the confidence region. While the confidence ellipsoids appear to overlap in a few projections (left), in most views we see that the means are distant (right). The groups may overlap, but their means are significantly different.

Extending this idea to deal with multivariate linear models is straightforward, although visualising the resulting surfaces will be difficult. If the model contains continuous variables, then we will no longer have point predictors of the mean, but instead continuous functions each surrounded by a confidence region. We can generate these confidence regions by randomly sampling the predictor space and then generating the confidence region for each sampled point as for MANOVA. Visualisation is complicated when we have multiple response and predictor variables as we want to keep them separate, e.g. a linear

combination of a predictor and a response variable probably doesn't make much sense. A potentially useful tool here is the correlation tour (Buja et al., 1996), a version of the tour which uses separate sets of variables for the x and y axes.

4.3.4 Case study: Classification models

A classifier is a model with a categorical response. Typically, the classifier is treated as a black box and the focus is on finding classifiers with high predictive accuracy. For many problems the ability to predict new observations accurately is sufficient, but it is interesting to learn about how the algorithms operate by looking at the boundaries between groups. Understanding the boundaries is important for the underlying real problem because it tells us where the groups differ. Simpler, but equally accurate, classifiers may be built with knowledge gained from visualising the boundaries, and problems of overfitting may be intercepted.

Much of what we know of how a classifier works is based on the knowledge of the algorithm. Each family of classifiers partitions the data space in a different way: LDA divides up the data space with hyperplanes, while trees recursively split the space into boxes. Most classifiers produce connected areas for each group, but there are exceptions, e.g. k-nearest neighbours. To visualise the model in the data space, we can either display the complete prediction regions or just their boundaries. These two approaches are shown in Figure 4.5.

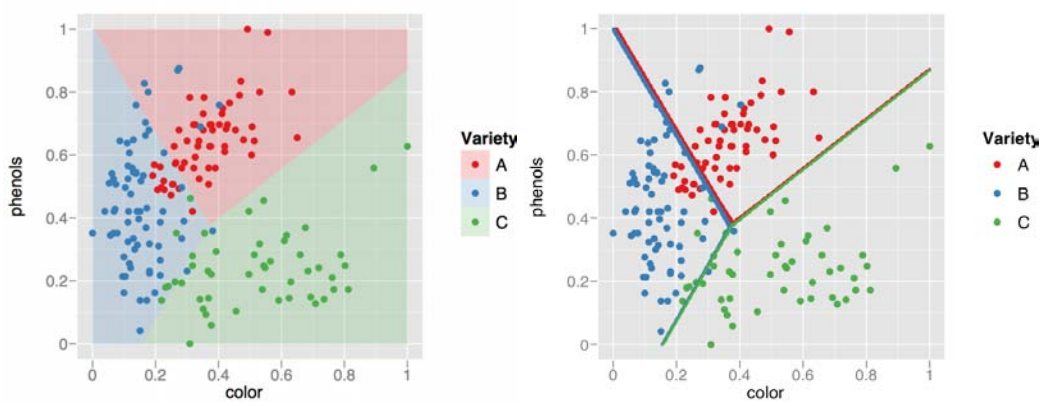


Figure 4.5: A 2d LDA classifier on the wine dataset. (Left) entire classification region shown and (right) only boundary between regions. Data points are shown as large circles.

Generating the complete prediction region is straightforward: sample many points from data space and classify each one. Generating only the boundary is more difficult. We may be able to use a closed form expression if available, but otherwise we can start with the full region and then remove non-boundary points. To determine if a point is near a boundary, we look at the predicted class probabilities. If the highest and second highest probabilities are nearly equal, then the classifier is uncertain about the class, so the point must be near a boundary. We call this difference in probabilities the *advantage*, and to find the boundary we discard points with values above a tolerance. The thickness of the resulting boundary is controlled by this cut off, and it should be small enough to give a sharp boundary, but

large enough that the set of boundary points is dense. This is a tricky trade-off. It may be possible to use adaptive sampling, sampling more densely in regions closer to the boundary, to do better.

If the classification function does not generate posterior probabilities, a k-nearest neighbours approach on the grid of predictions can be used. If the neighbours of a point are all the same class, then the point is not on a boundary. This method can be applied to any classification function, but is slow, $O(n^2)$, because it computes all pairwise distances to find the nearest neighbours. In practice, this imposes a limit of around 20,000 points.

Figure 4.6 illustrates the results of a support vector machine (Cortes and Vapnik, 1995) with radial kernel fitted to three variables from the wine data. We see the red region is contained almost completely inside the blue region, except where it abuts the green region. The boundaries and regions are straightforward to understand because our brains are adept at 3d modelling. http://www.vimeo.com/821284 explores the boundaries in another way, by brushing along the advantage variable. This shows that the boundaries are equal sharp between the different groups.

As we move beyond 3d it gets harder to see what is going on, but if we persevere we can find informative views. Figure 4.7 shows three informative views of a polynomial kernel with five variables. It looks like the boundary is largely flat (linear), apart from a region that bulges out of the blue into red. This is easier to see interactively, as at http://vimeo.com/823271.

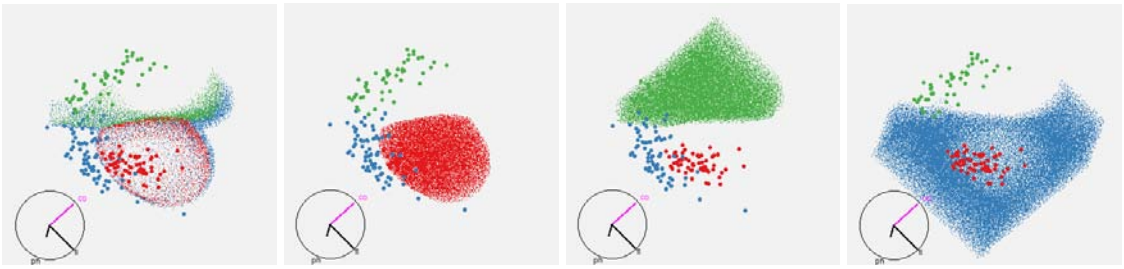


Figure 4.6: Views of a 3d radial SVM classifier. From left to right: boundary, red, green and blue regions. Variables used: color, phenols, and flavanoids. Animated version available at http://vimeo.com/821284.

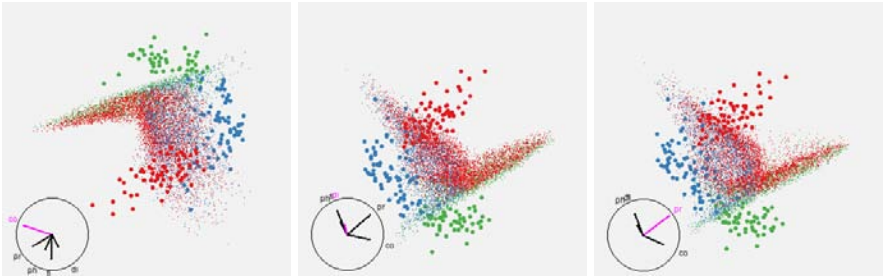


Figure 4.7: Informative views of a 5d SVM with polynomial kernel. It's possible to see that the boundaries are largely linear, with a "bubble" of blue pushing into the red. A video presentation of this tour is available at . Variables used: color, phenols, flavanoids, proline and dilution. Animated version available from http://vimeo.com/823271.

These ideas are implemented in the R package `classifly` (Wickham, 2007a), which can visualise classifiers generated by LDA and QDA (Venables and Ripley, 2002), SVM (Dimitriadou et al., 2006), neural networks (Venables and Ripley, 2002), trees (Therneau and Atkinson, 2008), random forests (Liaw and Wiener, 2002) and logistic regression; and it can easily to be extended to deal with other classifiers. More specific visualisations may also be useful for each of these families of models: Section 4.6 discusses neural networks; Cook et al. (2008a), support vector machines; and KLIMT provides many tools for visualising trees (Urbanek, 2002a,b, 2003).

4.3.5 Case study: Hierarchical clustering

Agglomerative hierarchical clustering methods build up clusters point by point, iteratively joining the two closest points or clusters. This requires a distance metric, and a method for calculating the distance between two clusters (linkage). There are a number of common methods: use the closest distance (simple linkage), the largest distance (complete linkage), the average distance (UPGMA), or the distance between cluster centroids (Ward's). Each of these methods finds clusters of somewhat different shapes: single linkage forms long skinny clusters, average linkage forms more spherical clusters.

The most common visualisation of a hierarchical clustering is a dendrogram, a d-inms display. There are also a number of other methods of this type: icicles (Kruskal and Landwehr, 1983), silhouettes (Trauwaert et al., 1989) and clustergrams (Schonlau, 2002). Figure 4.8 shows two dendrograms produced by clustering the wine dataset with Wards linkage and single linkage. These dendrograms aren't very informative. For the Wards clustering we can see that there are three major clusters, and for single linkage the clusters seem to grow mainly by adding a single point to an existing cluster. We can't see what the clusters have in common, or what variables are important for the clustering.

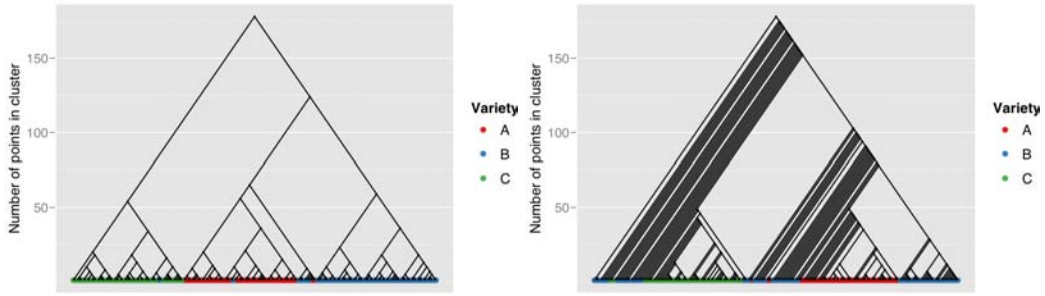


Figure 4.8: Dendrograms from a hierarchical clustering performed on wine dataset. (Left) Wards linkage and (right) single linkage. Points coloured by wine variety. Wards linkage finds three clusters of roughly equal size, which correspond fairly closely to three varieties of wine. Single linkage creates many clusters by adding a single point, producing the dark diagonal stripes.

To do better we need to display the model in data space. Buja et al. (1996) draws inspiration from the dendrogram and suggests an interesting idea: connect the observations in the data space according to their hierarchy in the clustering. For intermediate nodes in the hierarchy, representing clusters containing more than one point, we supplement the

data with extra points located at the cluster centroid. An extension would be to connect clusters in the same way as inter-cluster distances are calculated, i.e. connect closest points for single linkage, most distant for complete, and centroids for Wards. A related technique is supplementing a low-dimensional ordination with a the minimum spanning tree of the original data (Gower and Ross, 1969; Kim et al., 2000).

Figure 4.9 displays some views of the wine dataset supplemented with a hierarchical clustering with Wards linkage. It is easy to see the three groups and the hierarchical links in the tour, as shown at http://www.vimeo.com/768329.

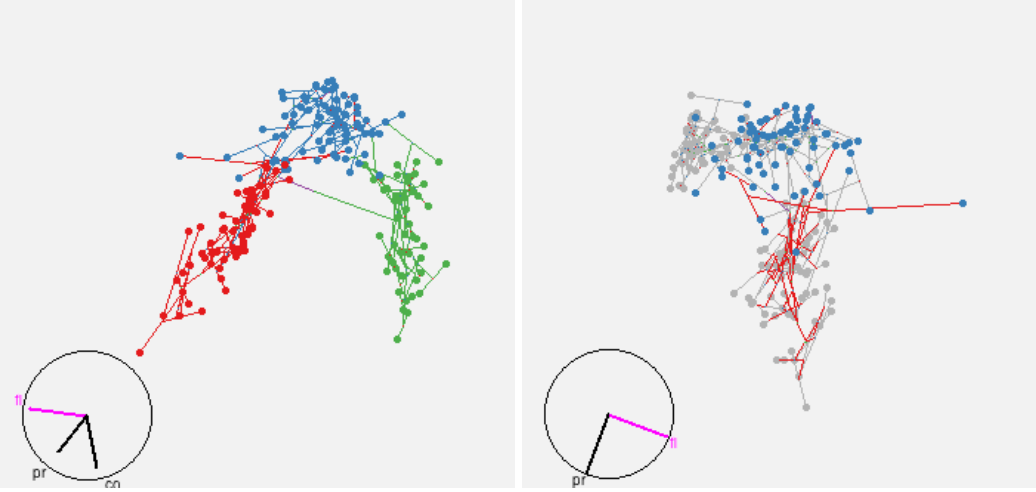


Figure 4.9: (Right) A informative projection of the flavanoid, colour, and proline variables for a hierarchical clustering of the wine data, with Wards linkage. Edges are coloured by cluster, to match the majority variety. The varieties are arranged in a rough U shape, with a little overlap, and some of the outlying blue points have been clustered with the red points. (Left) Only red links and blue points shown to focus in on the points that clustered with the wrong group. Animated version available at http://vimeo.com/768329.

An implementation of these ideas is available in the `clusterfly` package (Wickham, 2007b).

4.4 Collections are more informative than singletons

So far we have looked at visualising a single model in the data space, and many modern methods use collections of models. What can we do if we want to explore such a collection? One approach is to visualise all of the model simultaneously. This is the strategy of trace plots Urbanek (2005), which show hundreds of tree models simultaneously. This makes it possible to see the space of models, showing both the common and the unusual. However, in most cases we will not be able to view more than a couple of models simultaneously because they will overlap and obscure one another. To deal with this we need tools to help us to select smaller subsets of interesting models. We can do this by calculating descriptive statistics on multiple levels, then using linked brushing to connect these statistics to one another and to displays of the model. These ideas grow out of exploratory model analysis as described by Unwin et al. (2003); Urbanek (2004) and draw on the notions of

descriptive statistics (Bickel and Lehmann, 1975a,b).

Collections of models arise in many ways, and in most cases most of the models are ignored and we only look at the best model. This is perilous as we may miss alternative models that explain the data almost as well as the best model, and suggest substantially different explanations for the phenomenon of interest. Here are some collections that may arise in practice:

- From exploring the space of all possible models. For a given model family, we can generate all possible model forms. e.g. for a linear model, we can generate all possible linear models with main effects. This will often produce too many models to possibly fit or compare, so typically we will use some model selection algorithm to explore the space of "best" models. Typically, only the best, or maybe the two or three best models are examined.

- During the the process of data analysis. When analysing a data set, we may create and discard many models in an attempt to reveal the salient features of the data. It may be useful see all of these models simultaneously.

- While trying to find a global optima. When model fitting is not guaranteed to converge to a global optimum, we may have a collection generated from multiple random starts. This is useful for multidimensional scaling, neural networks, k-means clustering, and self organising maps. Typically, only the model with the highest criterion value is examined.

- By varying model settings. If a model has some tuning parameters, we can systematically alter them and observe the result, e.g. the penalty parameter in lasso, or the degrees of freedom in a smoothing term. Often, cross-validation is used to select a single optimal model.

- By fitting the same model to different datasets. These datasets might be groups within a larger dataset (perhaps a prequel to fitting mixed effects model), or might have been generated by leaving-one-out, cross validation, bootstrapping, permuting, simulating, as part of a sensitivity analysis or by adding random noise to the response (Luo et al., 2006). Typically, these multiple models are collapsed into a small set of summary statistics.

- As an intrinsic part of the model. Certain model families use ensembles of simple submodels: neural networks (ensembles of logistic models), random forests (ensembles of trees), bagging, and boosting. Typically, we just look at the overall model, and ignore how each part contributes to the whole.

- By treating each iteration as a model. If we are interested in exploring the process of model fitting, and the algorithm is iterative, we might treat each iteration as a separate model. This gives a unique ordering to the models and is described in depth in Section 4.5. Typically, we only look at the last iteration.

In this section, we focus on displaying the space of models, with the assumption that this space will be linked to additional m-in-ds displays. These extra displays will help us

tune our descriptive statistics to best match features of the model that we are interested in. Just as we would never look at a plot of a single data point, here we will never look at just the single "best" model. Best implies that there is some complete ordering of the models, which is often constructed with some ready made tradeoff between model complexity and fit built, like AIC or BIC.

This suggests a set of descriptive statistics to start with: model-level summaries of complexity and performance. Model complexity is often summarised by degrees of freedom, but there may be more informative measures for specific model families, e.g. the number of hidden nodes in a neural network. Model fit can be summarised with the likelihood at the parameters for models fit with ML, or by some measure of predictive ability. These statistics help us explore overall model quality and the tradeoff between quality and complexity, but do not give any information about how the models differ, how the fit of a given observation varies between models, or what exactly the parameters estimates are.

To explore these aspects of the model, we need descriptive statistics at other levels. A good example of a statistical procedure with built in summary statistics at additional levels is the random forest (Breiman, 2001), an ensemble method which uses many simple trees fit to random subsets of the variables and observation. As well as the model-level summaries described above, random forests provide tree-level, variable-level and observation-level descriptive statistics:

- Tree-level. Each tree in the ensemble has its own test and training data sets and so can computes an unbiased estimate of classification error. Inspecting these errors allows us to find trees that perform particularly well or poorly. Looking at many good trees in the data space, allows us to see commonalities (suggesting important variables) and differences (suggesting correlations between variables).

- Variable-level. Each variable is ranked by how by the drop in model performance when that variable is randomly permuted. This is a summary of a tree-variable statistic, which computes variable importance for each tree.

- Observation-level. For each observation we have the distribution of predictions across all trees. This can show which observations are easy (or hard) to classify, and which classes are most often confused.

For other models, we will need to develop and calculate our own descriptive statistics. Often there will be a large existing literature which can be probed for ideas. Another approach is to calculate a familiar summary statistic over an unfamiliar population. For example, we could calculate an observation-level summary by computing the average residual over the models. If models in the collection have common parameters, we might create parameter-level summaries of the distribution of the estimates over the models. The important thing is to generate descriptive statistics at multiple levels, in order to gain maximum insight into the models.

Once we have computed the summary statistics, we need to explore them. Static plots are helpful, but make it difficult to link between summaries at different levels. For example, we might want to see how the model-estimate summaries differ between the best and second best models. Linked brushing is particularly useful here. We can have one plot of

model-level statistics and one of model-estimate-level statistics and use brushing to link between them. This idea is described in more detail in the case study.

The RAFT tool (Breiman and Cutler, 2008) for visualising random forests provides a good set of static graphics, but provides no way to link between the plots, and no m-in-ds visualisations. This is a problem: if we found a single tree that did a particularly good job, we would like to see how it divides up the data space. Without linked graphics it is difficult to see exactly what is going on within a random forest, and to see how they compare to other models.

4.4.1 Case study: Linear models

In this case study we'll explore a collection of linear models containing all possible main effects models for a given dataset (or a large subset of these models). We will assume we have m models describing a data set with n observations and p variables. If all possible main effects models are fit, there will be $2^p - 1$ models in the collection. We will explore summary statistics on five levels:

- Model level: model fit statistics. m observations.
- Model-estimate level: coefficient estimates on various scales. $m \times p$ observations.
- Estimate level: summary of estimates over the models. p observations.
- Model-observation level: residuals and influence measures. $m \times n$ observations.
- Observation level: the original data, plus summaries of residual behaviour. n observations.

The relationship between these different levels is shown schematically in Figure 4.10. In this case study, we will focus on the the model and model-estimate level summaries. The remainder are discussed more fully in Wickham (2007d).

We will use a data set on fertility in French-speaking Swiss provinces in the late 1800's (Mosteller and Tukey, 1977). We are interested in predicting fertility based on the proportional of agricultural workers, average performance on an army examination, amount of higher education, proportion of Catholics and infant mortality. There are 47 observations and six predictor variables, giving a collection containing 31 ($2^5 - 1$) models. The data itself is rather irrelevant but it is a well-worn dataset which presents many interesting features.

Figure 4.11 shows the model-level summary statistics: a measure of model complexity, degrees of freedom; and five measurements of model fit, log-likelihood, AIC, BIC, R^2 and adjusted R^2. The pattern is very similar across measures, with improvement in model quality decelerating as model complexity increases. The improvement plateaus after six degrees of freedom, suggesting that a six df/four variable model is the 'best' (one degree of freedom is used by the estimate of intercept, and one by the estimate of the variance).

Each model contains between 1 and p variables, and for each variable in each model we calculate:

- The raw estimate. Useful when all covariates are on the same scale.

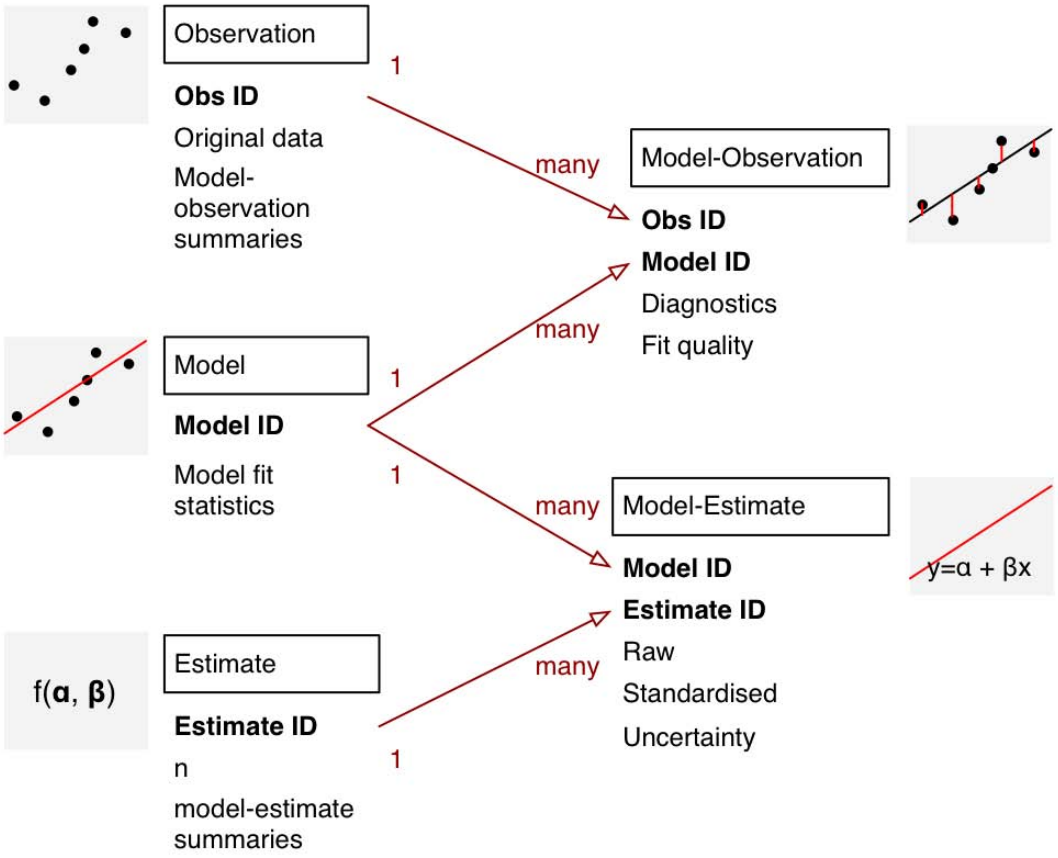


Figure 4.10: Relationship between five levels of summary statistics for a collection of linear models. Arrows indicate one-to-many relationships, e.g. for each model-level summary, there are many model-observation and model-estimate statistics.

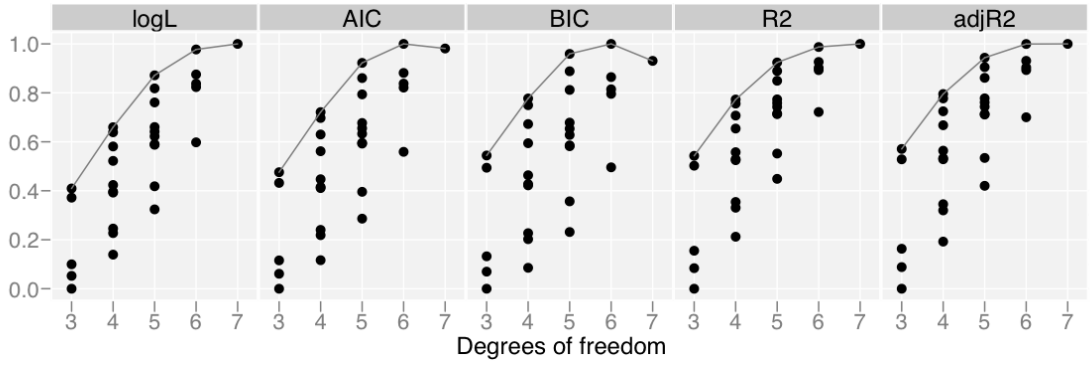


Figure 4.11: Model summary statistics, each scaled to $[0, 1]$ to aid comparison. A grey line connects the best models. The intercept only model is not displayed. Degrees of freedom include calculation of intercept and variance. These summary statistics suggest that a 6 df/4 variable model has the best tradeoff between complexity and performance.

- The standardised estimate, from model fit to data standardised to mean 0, standard deviation 1. Useful as measure of relationship strength. Can be interpreted as the change in predictor when response changes by one standard deviation, if all other variables are held constant.

- The t-value and absolute t-value. Allow us to assess the significance and direction of the relationship between the response and predictor.

We can use this graphic to explore the variance-covariance matrix of the predictors. There are two interesting examples of this in Figure 4.12. First, the standardised coefficients of infant mortality are very similar across all models. This indicates that this variable is largely independent of the other explanatory variables. Another interesting phenomenon is the behaviour of the agriculture variable. For four of the models the relationship between fertility and agriculture is positive, while for all others it is negative. Figure 4.13 highlights these models and reveals that they all fit the data poorly, and do not include examination or education covariates.

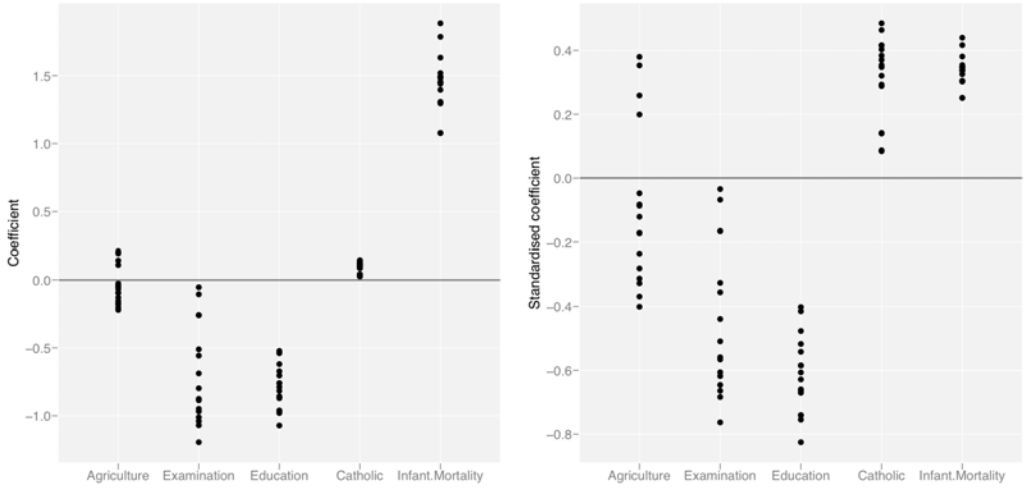


Figure 4.12: (Left) Raw coefficients are useful when variables are measured on a common scale. (Right) In this case, as with most data sets, looking at the standardised coefficients is more informative, as we can judge relative strength on a common scale. The education variable has the most negative coefficients. Only catholic and infant mortality variables are consistently related to increased fertility.

Figure 4.14 shows another way to use these lined plots, by highlighting the two best models and then inspecting their standardised coefficients. We can see that the best and second best models differ on inclusion of the examination variable.

These ideas are implemented in the `meifly` package (**m**odels **e**xplored **i**nteractively) which uses R to fit the models and calculate the summary statistics, and GGobi to display them.

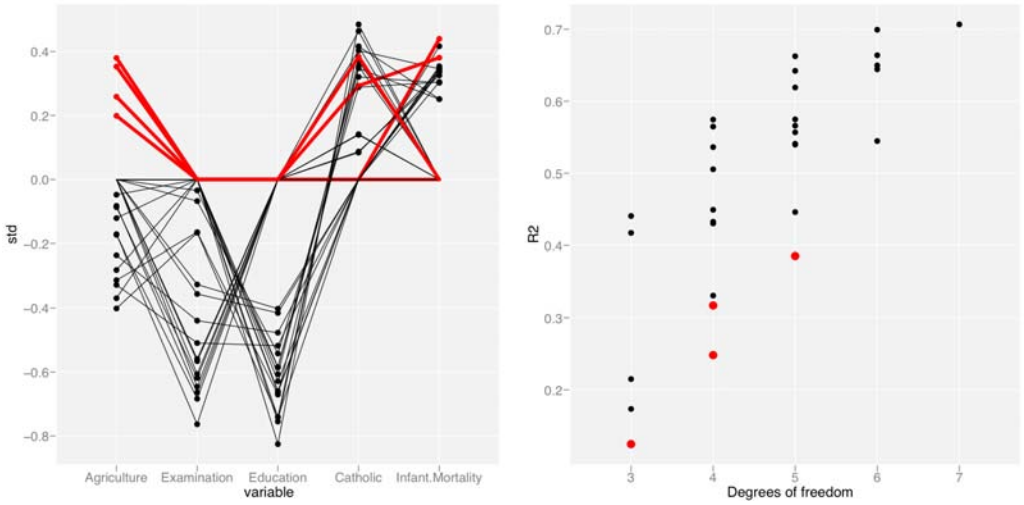


Figure 4.13: (Left) Parallel coordinates plot of standardised coefficient vs variable. (Right) Scatterplot of R^2 vs degrees of freedom. The four models with positive values of the agriculture coefficient have been highlighted in both plots. These models all fit the data poorly, and include neither examination nor education variables.

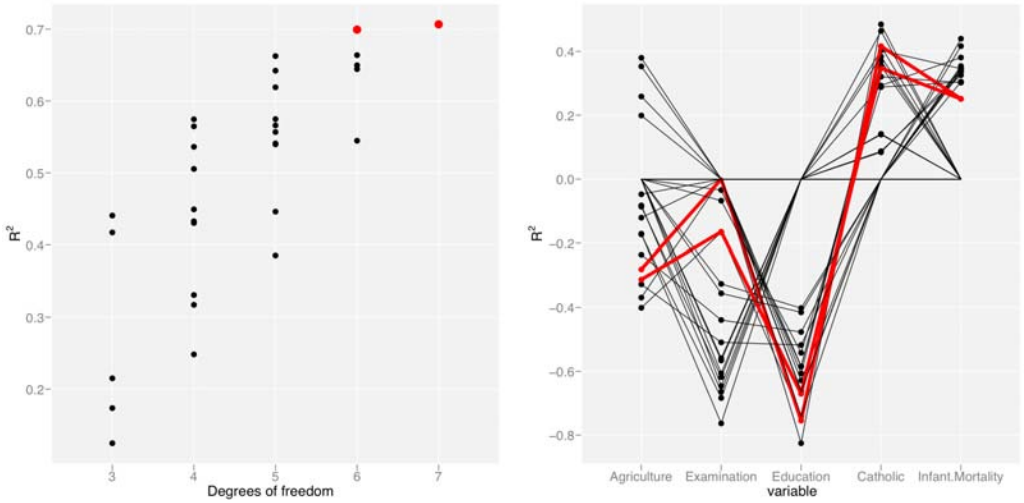


Figure 4.14: (Left) Scatterplot of R^2 vs degrees of freedom. (Right) Parallel coordinates plot of standardised coefficient vs variable. The two best models have been highlighted in red in both plots.

4.5 Don't just look at the final result; explore how the algorithm works

Whenever we can gain insight into the process of model fitting, we should. Observing iterations helps us understand how the algorithm works and can reveal potential pitfalls. Developing suitable visualisations forces you to think deeply about exactly what the algorithm does and can suggest possible avenues for improvement. For some algorithms, it may be possible to intervene, contributing a more global perspective to help the algorithm escape local maxima.

Many statistical algorithms are inherently iterative: IRLS in the generalised linear model, the Newton-Raphson method in maximum likelihood, the EM algorithm, multidimensional scaling, and k-means clustering. Some these methods are guaranteed to converge to global optimum (e.g. IRLS, EM), but most are not. We can think of each iteration as its own model, and we want to explore the changes over time. Many of the messages of Section 4.4 also apply here: we should look at each step, not just the last one; we should try and display all models on a single plot where possible; and if not, we should develop good summary statistics. The specific feature of collections generated by iteration is that there is a unique ordering of the models.

This ordering suggests two approaches to visualisation: ordering in space, to make time series plots, and ordering in time, to make movies. The time series plot displays time on the x-axis and some summary statistic on the y-axis, and shows at a glance the progress of a single parameter. This only works for a small number of numeric summaries, so for more complicated summaries we can make a movie. A movie strings together many static plots, each of which captures the state of the model at a single point in time. The `animation` package (Xie, 2008) is a rich source of animations in R, but only provides one animation that displays the progress of an algorithm, k-means clustering.

To capture the data we need for these plots, we can either stream data updates, replacing the values from the previous iteration; or store all iterations for later exploration. Streaming updates allows us to intervene in the algorithm and observe the effect of our changes, while storing everything allows us to later run time forwards and backwards. A particularly nice example of intervening in the progress of an algorithm is `ggvis` (Buja et al., To appear), an interactive tool for multidimensional scaling that allows you to manually drag points out of local optima.

Being able to extract this data is dependent on the implementation of the algorithm. If you are an algorithm developer, you need to provide either a hook into the algorithm which calls a function at every iterations; or the ability to run the algorithm for a fixed number of steps, and to be able to start the fitting process from the results of a previous run. This is not always straightforward as many algorithms also have time-varying parameters that must be captured and recreated. Typically, extracting and saving this data will slow down the algorithm considerably, often by an order of magnitude.

The following two case studies illustrate some of these ideas. The first case study, on projection pursuit, visualises the use of simulated annealing to find interesting projections of the data. The second case study explores the fitting process of self organising maps, as implemented by the `kohonen` package.

4.5.1 Case study: Projection pursuit

The guided tour (Cook et al., 1995) is a combination of the grand tour and projection pursuit. Instead of randomly picking new projections, as in the grand tour, the guided tour only picks new projections that are more interesting, by optimising an index of interestingness with simulated annealing (Kirkpatrick et al., 1983). This case study investigates how we can gain insight into the path that the simulated annealing takes. The indices we use in this example come from the graphic theoretic scagnostics of Wilkinson et al. (2005) and are implemented in the `scagnostics` package (Hofmann et al., 2006).

Figure 4.15 shows a time series of the scagnostic index being optimised, clumpiness, which is takes high values when the data is grouped into distinct clumps. We can see that the paths are not monotone: there is a stochastic component to simulated annealing which helps it avoid local optima. Each of the points on the plot corresponds to a projection matrix, and it is interesting to see if all the peaks correspond to the same or different projections. Figure 4.16 shows the four projections with highest clumpiness. Visualising the projection matrices themselves is difficult (each column corresponds to a point on a p-dimensional hypersphere) but can be revealing (Cook et al., 2008b).

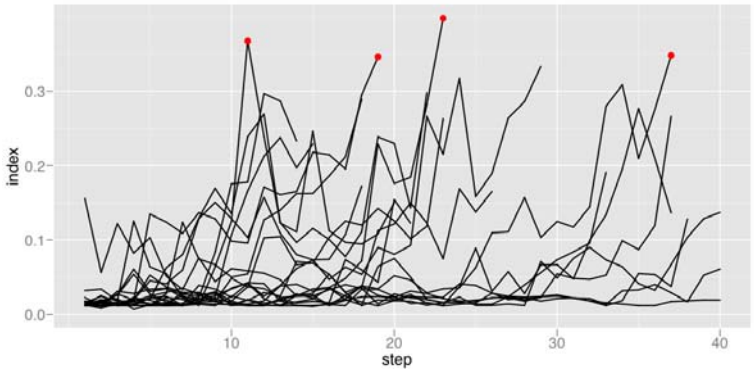


Figure 4.15: Variation of the clumpy index over time. Simulated annealing with 20 random starts, run until 40 steps or 400 tries. Red points indicate the four highest values.

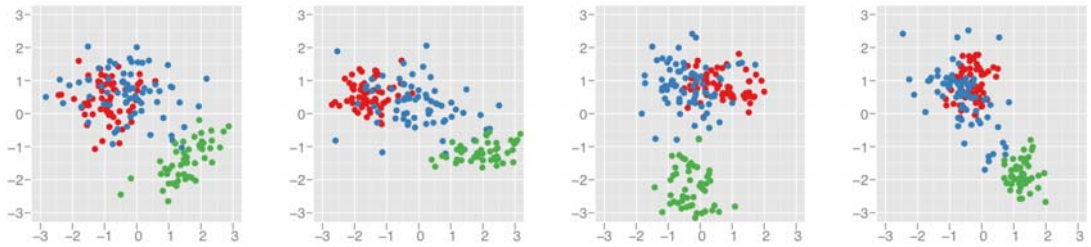


Figure 4.16: Four projections of the data with highest values of clumpiness, found by the simulated annealing shown in Figure 4.15. Plots are ordered left to right from highest to lowest. All these projections have good separation between the green group at the others, but not between the red and blue groups. Looking at the top 15 maxima does not find any projections that separate these two groups.

We can also use interaction with this plot to do something rather interesting: restart the simulated annealing from one of the local optima and see if it can do better in that same location. Figure 4.17 restarts from a local optima and shows that it is difficult to do better.

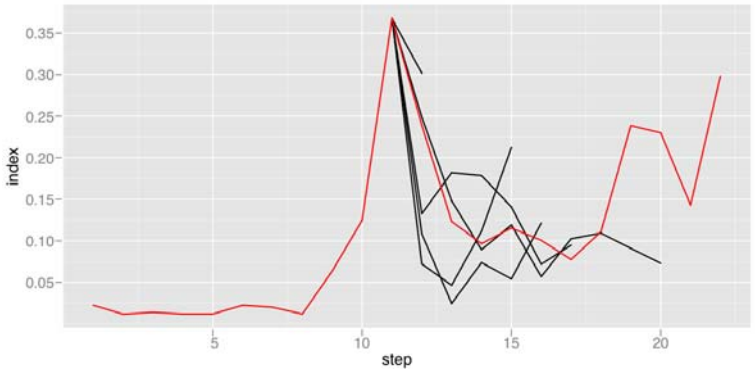


Figure 4.17: One of the guided tour runs of Figure 4.15 with new guided tours launched from particularly good projections. Red line shows path of original simulated annealing.

4.5.2 Case study: self organising maps

A self-organising map (SOM) is a machine learning algorithm which simultaneously performs clustering and dimension reduction (Kohonen, 2001). SOMs are often described as a type of neural network, but they are more easily understood as a type of constrained k-means. They are intuitively simple: we are wrapping a net of points into the data cloud. The knots in the net are called nodes, and are constrained by their neighbours. Each node has a position in the data space (like a cluster centroid), and the model space (a grid coordinate on the net). In this case study we will look at the self organising maps implemented in the `kohonen` package (Wehrens and Buydens, 2007).

We first need a method for visualising the model in the data space. The majority of SOM displays show the data in the model space, and give little insight into the quality of the model. An idea for a visualisation comes immediately from the net metaphor: we'll display the data, the nodes, and the connections between neighbouring nodes; a net wrapped through the data. Figure 4.18 contrasts a typical d-in-ms view with a m-in-ds view for 10 $\times$ 3 node svm fit to the wine data.

The algorithm for fitting a SOM is very similar to k-means, but instead of updating a single node at a time, we also update that node's neighbours:

1. Randomly initialise nodes to positions in data space.

2. Iterate through each data point, and assign it to the closest node
   a) Update the location of node: $(1 - \alpha) \cdot \text{node} + \alpha \cdot \text{point}$
   b) Also update the location of all neighbouring nodes within distance r on grid

3. Decrease α and r, and repeat step 2 until stopping condition is met

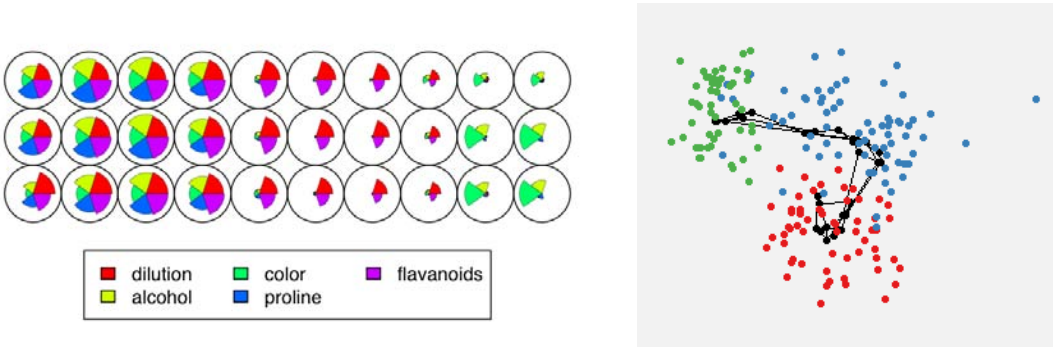


Figure 4.18: (Left) A visualisation of the model space. Each node is represented by a by a Coxcomb plot which represents the position of that node in the data space. (Right) A projection of the model embedded in the data space.

For the wine dataset, we'll fit a 10×3 net: we know the data seems to fall along a 1d path, but maybe there's something interesting along another dimension. To record the progress of the model fit, we'll save the following information at each of 100 iterations: the positions of the nodes, the values of r and α, and the distance from each point to its corresponding node.

Figure 4.19 summarises the quality of the model fit over time, by plotting the distance between each point and its corresponding node. Looking at the mean alone is misleading: there is a dramatic drop in mean distance when switching from a radius of three to two. However, when we look at the complete distribution, we can see that this drop is small compared to the total variation. We also see a number of outlying points that never get close to a node. The variation in those lines might also make us wonder if we have stopped the algorithm too soon.

To see how the model changes over, Figure 4.20 displays the model in the data space for four selected iterations. A movie showing more iterations, more views of the model, and interaction between the model and model summaries is available at `http://vimeo.com/823541`. It's interesting to note that the green and blue groups are close in the data space, but distant in the model space. We could improve our SOM model by joining the two ends. We also might have expected that the middle of the net would go through the blue group.

When we first used this package, it was not possible to break up the iterations into single steps, because there wasn't sufficient control over the changes to r and α. After asking the package author to give access to the iterations, this visualisation revealed some problems with the implementation of the SOM. The m-in-ds visualisation also revealed that the nets were twisted and tangled up. The author has since fixed these bugs.

4.6 Pulling it all together: visualising neural networks

This case study pulls together all three themes of the paper to develop visualisations that investigate the fit of single-hidden-layer neural networks, as implemented by the `nnet` package (Venables and Ripley, 2002). Much like self organising maps, neural networks are often treated as black boxes, and are used purely for prediction and not to give insight into the data. We want to dig deeper, and explore how they work, how well they work,

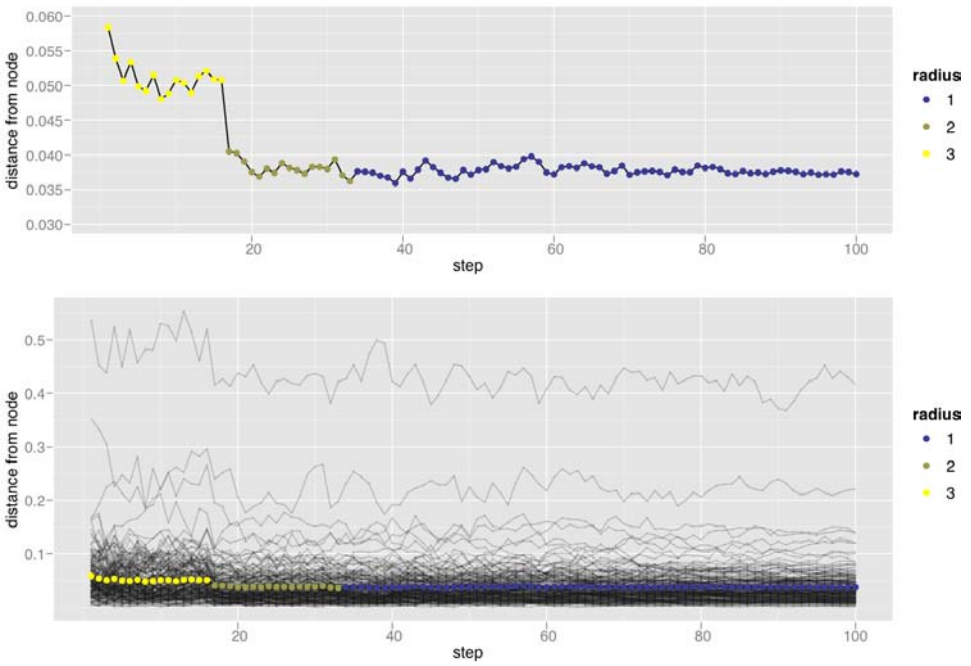


Figure 4.19: (Top) Time series of mean distance from point to corresponding node, over time. (Bottom) Distance to corresponding node for each individual point. Note the difference in scale! Mean points are coloured by radius. Alpha is not displayed, but deceases linearly from 0.05 to 0.

and what we need to do to get the best performance. This longer case study is broken up into three parts:

- In Section 4.6.1 we will develop a visualisation of the output of the neural network in the data space, based on ideas from the classification section.
- The definition of neural networks reveals that they are an ensemble model, composed of multiple logistic regressions. In Section 4.6.2 this insight will lead us to develop a visualisation that shows what the internal sub-models are doing.
- The parameters of a neural network are estimated with a numerical method and often get stuck in local optima. Section 4.6.3 explores why this happens and how to ensure we get a good fit, by visualising many random starts and the process of iteration.

We will illustrate this section with a simple two class, two dimension, classification problem, shown in Figure 4.21. The two classes can be completely separated by a non-linear boundary, easily drawn in by hand: the human neural network can easily solve this problem! Even a simple linear classifier does well on this dataset, correctly classifying 92% of the points. This is a simple problem but it will reveal many interesting features of neural networks.

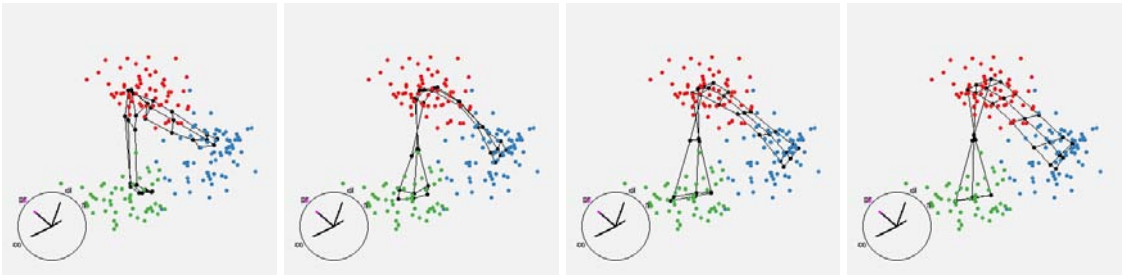


Figure 4.20: Iterations 1, 5, 25 and 100. After only one iteration the model seems to run through the centre of groups fairly well, and changes little over the next 99 iterations. The net performs as we expect, extracting the 1d path between the 3 groups.

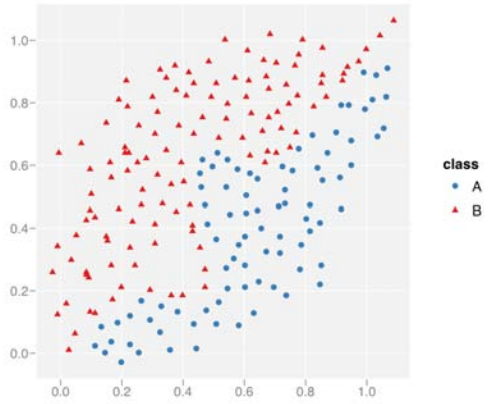


Figure 4.21: Simple two class, two dimension, classification problem. The classes can be completely separated by an easily seen non-linear boundary.

4.6.1 Model in data space

Assume we have been given a fitted neural network model. How can we visualise it? The basic output from the model is a prediction surface for each class, giving the probability that a new observation at (x, y) belongs to that class. In the two class case, we only need to look at one of these surfaces, as the two probabilities must sum up to 1. To visualise this surface, we have two basic options, shown in Figure 4.22. We could draw an image plot showing the probability at each location, or summarise the shape with a few contour lines, here at 10%, 25%, 50%, 75% and 90%. The 50% boundary determines whether a new observation will be classified as class A or B. Both displays show that the boundary does what we intuitively expect, and that the classification boundary is very sharp; this classifier is quite certain which class a new observation should be given.

4.6.2 Looking at multiple models: ensemble

A single-hidden-layer neural network is composed of a set of submodels, logistic regressions, whose outputs are combined using another logistic regression. In the terminology of neural networks, the sub-models are hidden nodes, and the number of these nodes is the

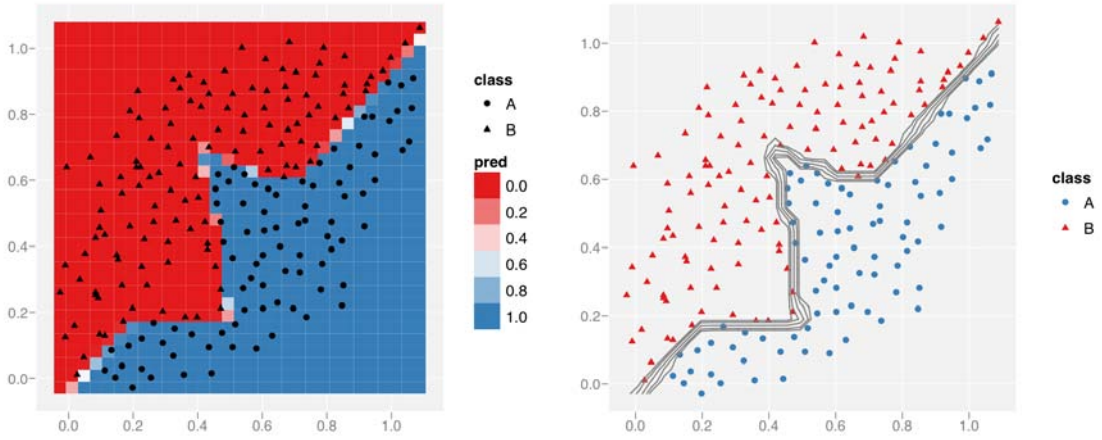


Figure 4.22: Visualising the classification surface. (Left) An image plot, showing probability that a new observation belongs to class B, and (right) a contour plot showing five iso-probability contour lines (10%, 25%, 50%, 75%, 90%). This classifier perfectly distinguishes the two classes, and is very sharp.

single most important tuning parameter. The hope is that these sub-models will identify important lower-level features which can then be combined to reveal important high-level features, much in the same way that biological neural networks (brains) work.

To be more precise, the probability that unit y with values $x_1, ..., x_p$ belongs to class j is

$$P(y \in \text{class}_j | x_1, ..., x_p) = k \cdot \phi(\alpha + \sum_{h=1}^{s} w_{hj} \phi(\alpha_h + \sum_{i=1}^{p} w_{ih} x_i))$$

where s is the number of hidden nodes, p is the number of explanatory variables, and k is a normalisation constant which ensures that the probabilities for all classes add up to one. The function ϕ can be anything but is usually taken to be the logit for classification problems.

So far we have just looked at the overall output of the model, but we can also dig in and see what the hidden nodes are doing. Each internal node is a logistic regression, so we can visualise it the same way as the overall output, except that now we need to display the output for each node. Since there only a few models, we can display each side-by-side, or superimpose the 50% contour line on a single plot. Note that output from the internal nodes does not map directly to a prediction class, but is used by the final logistic regression to generate classification probabilities. For our model, these displays are shown in Figure 4.23. Each internal node has a sharp boundary which identifies part of the piece-wise linear path that separates the groups.

This plot hints at some of the difficulties faced when estimating the model parameters. The parameters are over-specified, as we can swap any two hidden nodes, or flip the direction of logistic regressions along the boundary lines, and still get the same overall model.

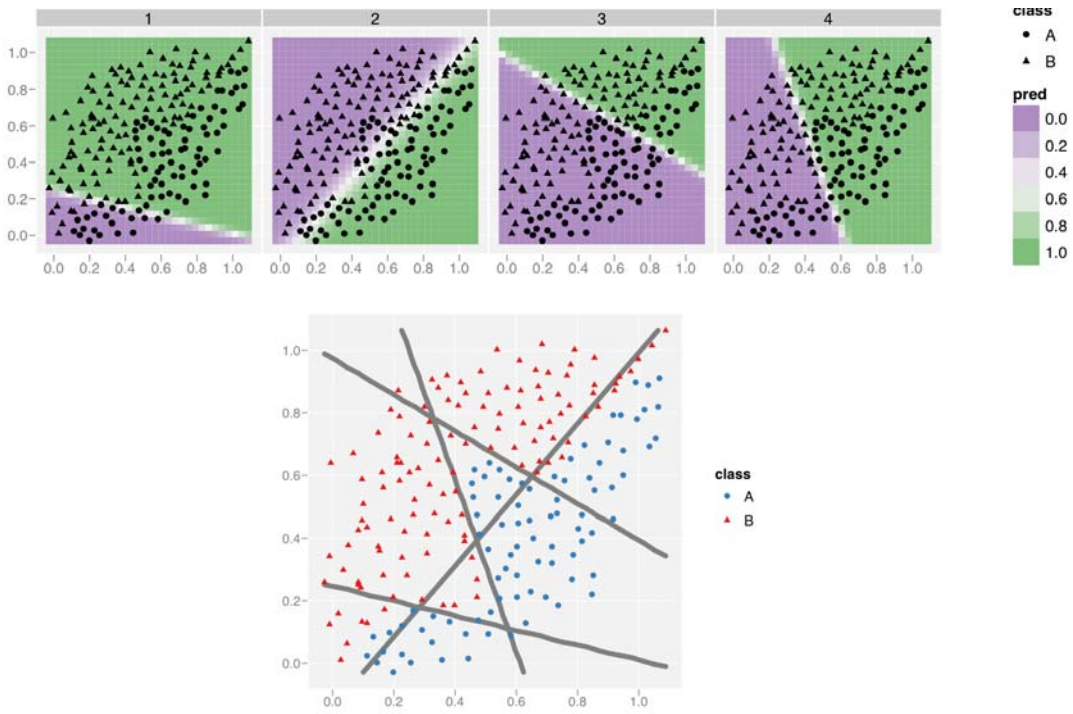


Figure 4.23: Visualisation of the hidden nodes of a neural network. (Top) The probability surfaces for the hidden nodes are displayed side-by-side. (Bottom) The 50% probability contours from the nodes are overlaid on a single plot. We see how each node identifies one linear break producing the final classification boundary shown in Figure 4.22.

4.6.3 Looking at multiple models: random starts

The parameters of the neural network are estimated by numerical optimisation. Neural networks often converge to local optima and it is customary to try multiple random starts to find the global optima. For our dataset, we will try 200 random starts each for neural networks with 2, 3 and 4 hidden nodes. This gives a total of 600 neural networks fit to the data. Figure 4.24 summarises the results. There is a striking variation in the ability of the neural network to correctly separate the two classes, with prediction accuracy ranging from 80% to 100%. Most of the networks correctly classify 92% of the points, no better than a linear boundary. Only one network classifies the training data perfectly! Interestingly, increasing the number of nodes doesn't affect the predictive ability of the majority of the networks, but, surprisingly, does increase the number of very good and very bad models.

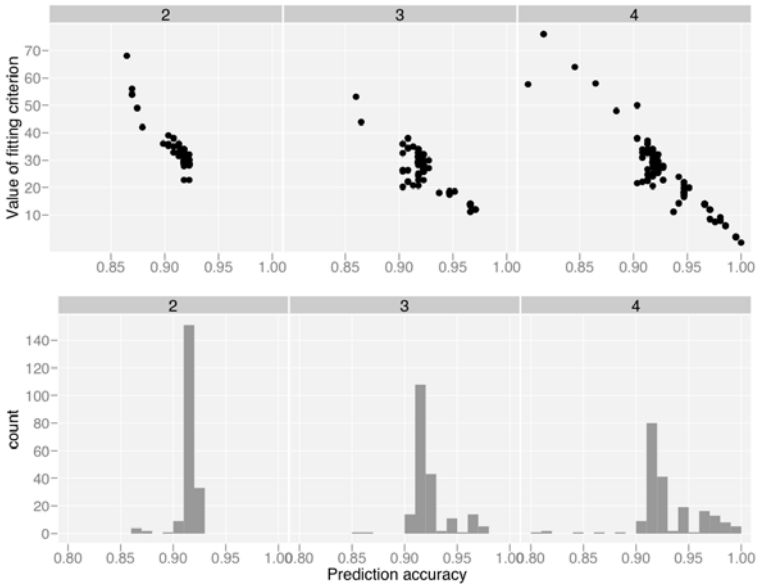


Figure 4.24: Summary of 600 neural networks with two, three and four hidden nodes. (Top) Scatterplot of prediction accuracy vs internal criterion. (Bottom) Histograms of prediction accuracy. Most networks achieve an accuracy of 92%, with just a few that do much better or much worse.

Figure 4.25 shows the classification boundaries of all 600 neural networks roughly broken down by model quality. There is a lot of overplotting, but we can see that many of the networks have a linear classification boundary, some seem to get one of the gaps, but not both, and a number of the four-node models capture the essence of the boundary, even if they are not completely correct.

We can also use this same idea to explore the process of model fitting, treating each iteration as a model in its own right. The video at `http://www.vimeo.com/767832` shows the changes in the hidden nodes as the algorithm proceeds. An alternative approach to display the iterations simultaneously by only showing the 50% boundary, as in Figure 4.26. We can see the boundaries move rapidly at first, and then appear to stabilise.

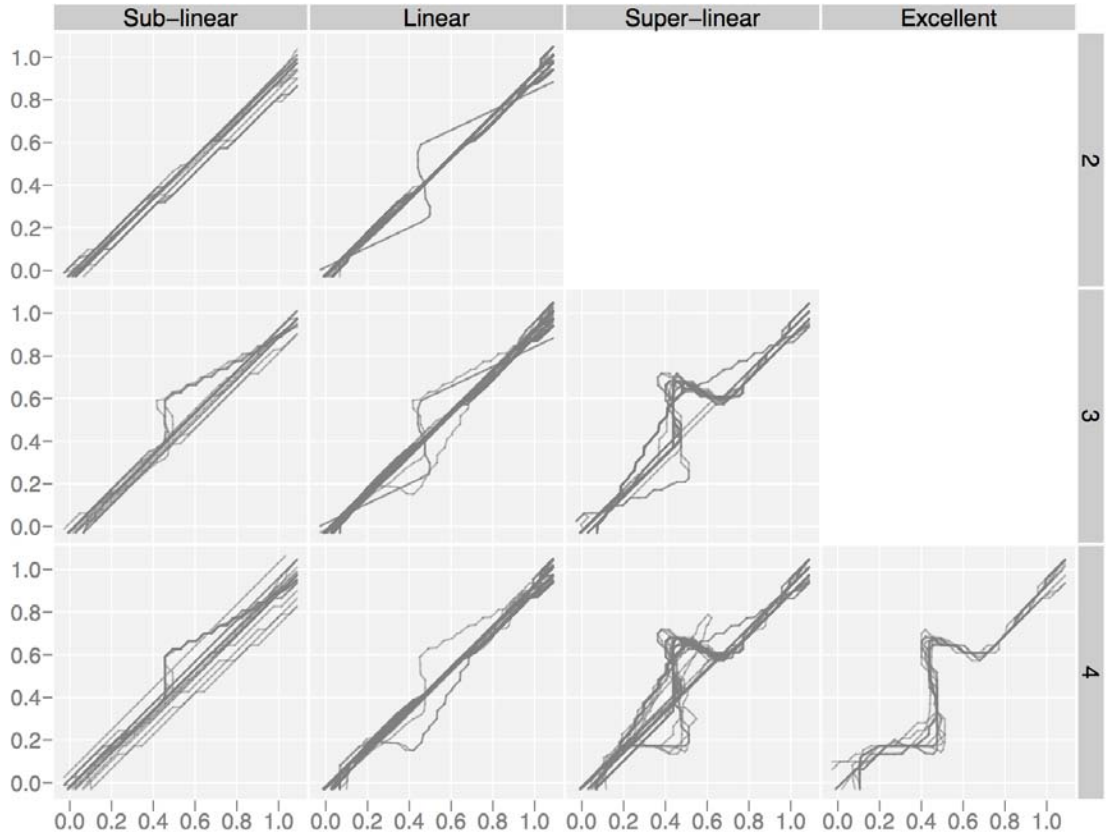


Figure 4.25: Classification boundaries from all 600 neural networks, broken down by number of numbers and groups based on accuracy. Sub-linear models have accuracy in $[0\%, 91.5\%]$, linear in $[91.5\%, 92.5\%]$, super-linear in $[92.5\%, 98\%]$, and excellent in $[98\%, 100\%]$. Most of the boundaries are linear, a few get one of the bumps, and quite a few of the four node networks get the overall form right, even if they are not perfectly accurate.

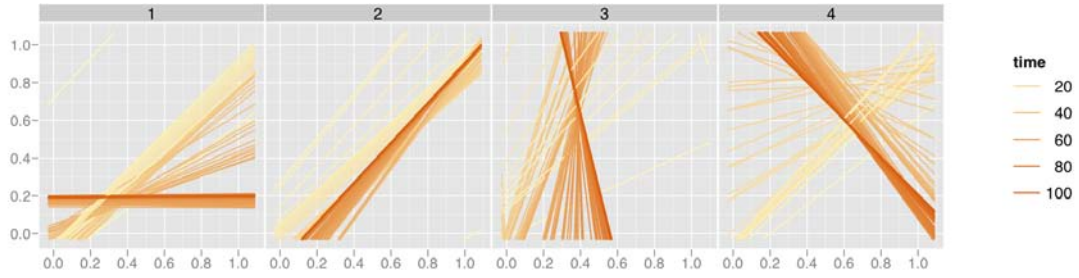


Figure 4.26: How the 50% probability line for each node changes with time. 100 runs with 5 iterations between each. Nodes start similarly, move rapidly in the first few iterations, then slow down, converging to surfaces that are quite different. Animated version available at http://www.vimeo.com/767832.

4.6.4 Real data

Even though we have used a very simple example we have still discovered many interesting features of the neural network algorithm. In real life the neural network will be considerably more complicated: more data points, more internal nodes, and more possible classes. However, we can continue to use the same ideas, with a few tweaks:

- If there are more than two classes, we can longer summarise the model with a single probability surface, but we will need one for each class. When computing classification boundaries we will need to use the advantage statistic developed in Section 4.3.4.

- If there are more than two input variables, we will need to use high-dimensional tools to visualise the classification surface. These were also described in Section 4.3.4.

- More data points shouldn't affect our choice of visualisations, but it will make the neural networks slower to fit. This may affect our ability to try the many possible random starts necessary to find a good fit.

4.7 Conclusion

We have presented three important strategies for visualising models: visualise the model in data space, collections are more informative than singletons and explore the process of model fitting. These were illustrated with many case studies that developed informative visualisations using the strategies outline in the paper. The tight connection between R and GGobi facilitated the development of all of the examples, and has resulted in the development of R packages that one can easily download and try out.

To make these tools useful in practical model building we need to improve our ability to visualise high-d data and surfaces. We have the basic tools, but as the number of dimensions increases, the number of potential views becomes overwhelming. How can we provide support for finding revealing views, while maintaining the ability of the analyst to look for the unexpected? We have touched on projection pursuit for finding interesting views of the data, and we expect developing descriptive statistics that summarise both the model and data to be a fruitful approach. The challenge is, as always, developing good summaries, but our graphical tools can help us develop these for low-d data for application to high-d data.

We also need better tools to visualise high-d surfaces. In this paper we have gone to considerable lengths to approximate high-dimensional surfaces with a dense set of points. Development of better sampling methods would be useful, but the development of higher dimensional drawing primitives is more of a priority. GGobi currently provides 0d (points) and 1d (lines) primitives, can we supplement these with 2d primitives (triangles?) as well? The development of adaptive contouring algorithms for high-d would also be useful.

The case studies in this paper are just a beginning and we hope that others pick up these ideas to develop visualisations for more models in a wider array of statistical and graphical environments.

4.8 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0706949.

Chapter 5

Conclusion and future plans

The three papers in this thesis described three practical tools for exploring data and models. Chapter 2, "Reshaping data with the `reshape` package", described a framework for reshaping data. Chapter 3, "A layered grammar of graphics", described a framework for static statistical graphics. Chapter 4, "Visualising statistical models: removing the blindfold", described strategies for visualising statistical models. These papers are united by their practical nature and their philosophy of data analysis. In this conclusion I will expand on these unifying themes, investigate the impact of my work and discuss future directions.

5.1 Practical tools

A particular emphasis of my work is the development of practical tools. The ideas presented in this thesis are implemented in the open-source R packages `reshape`, `ggplot2`, `classifly`, `clusterfly`, and `meifly`, all of which are available from CRAN.

This emphasis on practical tools grows out of my experiences in consulting and teaching. The variety of data formats provided by my consulting clients and the difficulty restructuring them with existing tools led to the `reshape` package, while `ggplot2` grew out of my frustration with existing graphics tools in R, particularly the plots that were easy to imagine but took hours of work to create. For me, `reshape` and `ggplot2` substantially reduce the time between the conception and realisation of a data structuring or graphic, giving me more time to dig into the data. This has also increased the number of investigations that I can perform on the fly, while in front of a client, which is particularly useful for shortening the communication loop.

Teaching R forces you to re-see R through the eyes of a beginner. You are forced to reconsider all of the idiosyncrasies that you once struggled with, but have now become so internalised that you don't realise they are there. R has some delights. The modelling framework is particularly strong: once you have mastered the linear model, you can easily use extensions like the generalised linear or the mixed effects model, and learning new model classes is straightforward. But data analysis is not just about fitting models, and other aspects of R are not so pleasant. For example, the built-in graphics and data manipulation tools seem to consist only of special cases, all with different parameters and different outputs. This requires much rote memorisation to master, and makes it hard for students to see the underlying themes. By building consistent frameworks for thought and action, `reshape` and `ggplot2` make it much easier to teach data analysis because you can focus on

the big picture and not worry so much about the minor details.

The tools for models visualisation are also pedagogically useful, as they give students a way to see how the methods work. They fit naturally in to the process of data analysis and force students to consider their models in the light of the data.

5.2 Data analysis

The process of data analysis is not a straight line from point A to B, but involves much circling back, and looking at the map to see we've been and where we should go next. Often we will need to return to previous findings to reinterpret them in terms of what we have since learned. Not only will our models of the data change, but also our visualisations and even the form of the data itself. Velleman (1997) captures this spirit well with his aphorism "*The process of data analysis is one of parallel evolution. Interrelated aspects of the analysis evolve together, each affecting the others*." While presented separately, in practice the three tools of this thesis work together to build our understanding of the data. This process can be seen in more depth in Hobbs et al. (To appear), which involved multiple re-expressions of the data, and many plots and models that never saw the light of day.

While powerful, these tools are lacking in one respect. They fail to combat what Francis Bacon describes as "*The human understanding, on account of its own nature, readily supposes a greater order and uniformity in things than it finds. And . . . it devises parallels and correspondences and relations which are not there*." We need tools to combat the innate human ability to see patterns in random data, and the innate desire to tell stories about data. To balance our curiosity we need methods to encourage scepticism and remind us to question what we see. Of course classical statistics provides a great number of these tools, but how can we apply them in an exploratory, graphical situation? Buja et al. (1988); Swayne et al. (1998) provide some initial ideas from which I hope to build from in the future.

5.3 Impact

It is difficult to judge the impact of your work. Unfortunately, because of the way that CRAN is set up, it is not possible to know exactly how many times my packages have been downloaded. However, last month around 1,600 people viewed package related pages on my website and 18 people emailed me a question about one of my packages. An additional indicator of impact is that for graphics and data manipulation problems posed on R-help, others will offer a solution using `reshape` or `ggplot`. For my reshape paper, the Journal of Statistical Software tracks the number of downloads and as of March 28 2008, the paper has been download 982 times, approximately five times per day on average since it was published.

Professionally, my work has resulted in an invitation to teach a 3-day course at the University of Zurich in July 2007, and to speak at the World Congress for Probability and Statistics in Singapore in July 2008.

5.4 Future work

I see much of my future work evolving around two obvious deficiencies of the layered grammar: it is purely static, with no interaction; and it gives no insight into area plots for categorical data. I am also interested in developing other frameworks that facilitate common tasks in data analysis, and in building a toolkit for graphical inference.

Interactive graphics are an important family of tools because they speed up the process of data analysis. With a well designed interactive graphics package, the time between steps is further decreased because we can modify the previous representation, rather than having to start from scratch. This makes it important to extend the layered grammar to also describe interaction. Currently, the layered grammar is purely static; it does not describe how we can interact with the plot to change what it displays.

Extending the grammar to include interaction will be challenging. In preparation for this work, I have begun studying the types of interaction present in GGobi (Swayne et al., 2003), Mondrian (Theus, 2003), MANET (Unwin et al., 1996) and Excel. A particularly important component of interactive graphics is linked brushing, and am currently working on a paper about this with Graham Wills. One of the biggest challenges in adapting `ggplot2` to incorporate interaction will be ensuring that plotting is fast enough so that the sequence of static plots is seamless. To this end, I have been working on the data pipeline which underlies GGobi (Wickham et al., Accepted), particularly how the pipeline coordinates events across different views of the same data.

The grammar poorly describes area plots for categorical data: spine charts, mosaic plots, equal-bin-size plots, fluctuation diagrams (Hartigan and Kleiner, 1984; Hofmann, 2001; Unwin et al., 1996), and the closely related treemaps (Shneiderman, 1992) and dimensional stacking (LeBlanc et al., 1990). These plots represent each combination of categorical levels by a rectangle with area proportional to the number of observations, hence the name area plot. The plots differ primarily in whether they focus on conditional or joint distributions, and whether or not marginal distributions are visible. With Heike Hofmann, I am working on an extension to the grammar that describes all of these plots in a consistent framework, helping us to see the underlying commonalities and to suggest new plot types. There is also an interesting connection with log-linear models, particularly as a tool for removing marginal effects once they have been recognised.

I am also interested in developing better tools for other data analysis tasks. For example, a common problem-solving strategy is to break a big problem into small chunks, operate on each chunk individually and then join them all back together. In R, there are a number of functions that support parts of this strategy, but they are scattered and inconsistent.

5.5 Final words

This thesis has presented three practical tools that support data analysis, improving our ability to explore data and models. The tools are not just computational, but also provide mental tools that support a cohesive philosophy of data analysis that stresses iteration and progressively building our understanding of the data.

Bibliography

A. Asuncion and D. Newman. UCI machine learning repository, 2007. URL http://www.ics.uci.edu/~mlearn/MLRepository.html.

R. A. Becker, W. S. Cleveland, and M.-J. Shyu. The visual design and control of trellis display. *Journal of Computational and Graphical Statistics*, 5(2):123–155, 1996.

J. Bertin. *Semiology of Graphics*. University of Wisconsin Press, Madison, WI, 1983.

P. J. Bickel and E. L. Lehmann. Descriptive statistics for nonparametric models. I: Introduction. *The Annals of Statistics*, 3:1038–1044, 1975a.

P. J. Bickel and E. L. Lehmann. Descriptive statistics for nonparametric models. II: Location. *The Annals of Statistics*, 3:1045–1069, 1975b.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

L. Breiman and A. Cutler. *RAFT Documentation*, 2008. URL http://www.webcitation.org/5WCgv1oW9. Accessed from http://www.stat.berkeley.edu/~breiman/RandomForests/cc_graphicsdoc.htm on 2008-03-09.

C. A. Brewer. Color use guidelines for mapping and visualization. In A. MacEachren and D. Taylor, editors, *Visualization in Modern Cartography*, chapter 7, pages 123–147. Elsevier Science, Tarrytown, NY, 1994a.

C. A. Brewer. Guidelines for use of the perceptual dimensions of color for mapping and visualization. In *Color Hard Copy and Graphic Arts III, Proceedings of the International Society for Optical Engineering (SPIE), San Jose*, volume 2171, pages 54–63, 1994b.

A. Buja, D. Asimov, C. Hurley, and J. A. McDonald. Elements of a viewing pipeline for data analysis. In *Dynamic Graphics for Statistics*. Wadsworth, Inc., 1988.

A. Buja, D. Cook, D. Asimov, , and C. Hurley. Theory and computational methods for dynamic projections in high-dimensional data visualization. Technical report, ATT, 1996. URL http://www.research.att.com/~andreas/papers/dynamic-projections.ps.gz.

A. Buja, D. F. Swayne, M. L. Littman, N. Dean, and H. Hofmann. Interactive data visualization with multidimensional scaling. *Journal of Computational and Graphical Statistics*, To appear. URL http://www-stat.wharton.upenn.edu/~buja/PAPERS/paper-mds-jcgs.pdf.

D. Carr. Using gray in plots. *ASA Statistical Computing and Graphics Newsletter*, 2(5):11–14, 1994. URL http://www.galaxy.gmu.edu/~dcarr/lib/v5n2.pdf.

D. Carr. Graphical displays. In A. H. El-Shaarawi and W. W. Piegorsch, editors, *Encyclopedia of Environmetrics*, volume 2, pages 933–960. John Wiley & Sons, Ltd, Chichester, 2002. URL http://www.galaxy.gmu.edu/%7Edcarr/lib/EnvironmentalGraphics.pdf.

D. Carr and R. Sun. Using layering and perceptual grouping in statistical graphics. *ASA Statistical Computing and Graphics Newsletter*, 10(1):25–31, 1999.

D. B. Carr, R. J. Littlefield, W. L. Nicholson, and J. S. Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey. *Graphical methods for data analysis*. Wadsworth, 1983.

C. Chatfield. *Problem Solving : A Statistician's Guide*. Chapman & Hall, 1995.

W. Cleveland. A model for studying display methods of statistical graphics. *Journal of Computational and Graphical Statistics*, 2:323–364, 1993. URL http://stat.bell-labs.com/doc/93.4.ps.

W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

W. S. Cleveland and R. McGill. Graphical perception: The visual decoding of quantitative information on graphical displays of data. *Journal of the Royal Statistical Society. Series A (General)*, 150(3):192–229, 1987.

D. Cook and D. F. Swayne. *Interactive and Dynamic Graphics for Data Analysis: With Examples Using R and GGobi*. Springer, 2007.

D. Cook, A. Buja, J. Cabrera, and C. Hurley. Grand tour and projection pursuit. *Journal of Computational and Graphical Statistics*, 4(3):155–172, 1995.

D. Cook, D. Caragea, and H. Wickham. Visual methods for examining SVM classifiers. In S. Simoff, M. H. Böhlen, and A. Mazeika, editors, *Data Mining: Theory, Techniques and Tools for Visual Analytics*, LNCS State of the Art Surveys. 2008a.

D. Cook, E.-K. Lee, A. Buja, and H. Wickham. Grand tours, projection pursuit guided tours and manual controls. In C.-h. Chen, W. Härdle, and A. Unwin, editors, *Handbook of Data Visualization*, Springer Handbooks of Computational Statistics, chapter III.2, pages 295–314. Springer, 2008b.

R. D. Cook. *Regression Graphics: Ideas for Studying Regressions through Graphics*. John Wiley Sons, 1998.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

D. R. Cox. Some remarks on the role in statistics of graphical methods. *Applied Statistics*, 27(1):4–9, 1978.

N. J. Cox. The grammar of graphics. *Journal of Statistical Software*, 17, 2007. URL http://www.jstatsoft.org/v17/b03/v17b03.pdf.

E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, , and A. Weingessel. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2006. R package version 1.5-16.

J. H. Friedman. Exploratory projection pursuit. *Journal of American Statistical Association*, 82:249–266, 1987.

G. W. Furnas and A. Buja. Prosection views: Dimensional inference through sections and projections. *Journal of Computational and Graphical Statistics*, 3(4):323–353, 1994.

A. Gelman and J. Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, 2006.

J. C. Gower and G. J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 18:54–64, 1969.

J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, and M. Venkatrao. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–53, 1997.

A. Gribov. Gauguin (grouping and using glyphs uncovering individual nuances), 2007. URL http://rosuda.org/software/Gauguin/gauguin.html.

J. A. Hartigan and B. Kleiner. Mosaics for contingency tables. In *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface*, pages 268–273, Fairfax Station, VA, 1981. Interface Foundation of North America, Inc.

J. A. Hartigan and B. Kleiner. A mosaic of television ratings. *The American Statistician*, 38 (1):32–35, 1984.

J. Heer and M. Agrawala. Multi-scale banking to 45 degrees. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 2006.

J. Hobbs, H. Wickham, H. Hofmann, and D. Cook. Glaciers melt as mountains warm: A graphical case study. *Computational Statistics*, To appear. Special issue for ASA Statistical Computing and Graphics Data Expo 2007.

H. Hofmann. Exploring categorical data: Interactive mosaic plots. *Metrika*, 51(1):11–26, 2000.

H. Hofmann. *Graphical Tools for the Exploration of Multivariate Categorical Data*. BOD, 2001.

H. Hofmann. Constructing and reading mosaicplots. *Computational Statistics and Data Analysis*, 43(4):565–580, 2003.

H. Hofmann, L. Wilkinson, H. Wickham, D. T. Lang, and A. Anand. *scagnostics: Compute scagnostics.*, 2006. R package version 0.1.0.

C. Huang, J. A. McDonald, and W. Stuetzle. Variable resolution bivariate plots. *Journal of Computational and Graphical Statistics*, 6:383–396, 1997.

R. Ihaka. Colour for presentation graphics. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, 2003. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/Ihaka.pdf.

A. Inselberg. The Plane with Parallel Coordinates. *The Visual Computer*, 1:69–91, 1985.

S.-S. Kim, S. Kwon, and D. Cook. Interactive visualization of hierarchical clusters using MDS and MST. *Metrika*, 51(1):39–51, 2000.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Science*. Springer, 2001.

J. B. Kruskal and J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37:162–168, 1983.

J. LeBlanc, M. Ward, and N. Wittels. Exploring n-dimensional databases. In *Proceedings of Visualization '90*, pages 230–237, 1990.

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL http://CRAN.R-project.org/doc/Rnews/.

X. Luo, L. A. Stefanski, and D. D. Boos. Tuning variable selection procedures by adding noise. *Technometrics*, 48(2):165–175, 2006.

F. Mosteller and J. W. Tukey. *Data Analysis and Regression: A Second Course in Statistics*. Addison-Wesley, Reading Mass., 1977.

D. Murdoch. Drawing a scatterplot. *Chance*, 13(3):53–55, 2000.

J. A. Nelder. Algorithm AS 96: A simple algorithm for scaling graphs. *Applied Statistics*, 25 (1):94–96, 1976.

F. Nightingale. *Notes on matters affecting the health, efficiency and hospital administration of the British Army*. Private publication, London, 1857.

OED Online. *The Oxford English Dictionary*. Oxford University Press, 2nd ed edition, 1989. URL http://dictionary.oed.com/cgi/entry/50097652.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. URL http://www.R-project.org. ISBN 3-900051-07-0.

D. Sarkar. *lattice: Lattice Graphics*, 2006. R package version 0.14-16.

M. Schonlau. The clustergram: A graph for visualizing hierarchical and non-hierarchical cluster analyses. *The Stata Journal*, 3:316–327, 2002. URL http://www.schonlau.net/clustergram.html.

S. R. Searle, F. M. Speed, and G. A. Milliken. Population marginal means in the linear model: An alternative to least squares means. *The American Statistician*, 34(0):216–221, 1980.

B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, 1992. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/102377.115768.

A. Shoshani. Olap and statistical databases: Similarities and differences. In *Proc. ACM PODS '97*, pages 185–196, 1997.

D. N. Sparks. Algorithm AS 44: Scatter diagram plotting. *Applied Statistics*, 20(3):327–331, 1971.

W. D. Stirling. Algorithm AS 168: Scale selection and formatting. *Applied Statistics*, 30(3): 339–344, 1981.

D. Swayne, A. Buja, and N. Hubbell. XGobi meets S: integrating software for data analysis. *Computing Science and Statistics*, 23:430–434, 1991.

D. F. Swayne, D. Cook, and A. Buja. Xgobi: Interactive dynamic data visualization in the x window system. *Journal of Computational and Graphical Statistics*, 7(1):113–130, 1998.

D. F. Swayne, D. Temple Lang, A. Buja, and D. Cook. GGobi: Evolving from XGobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 43:423–444, 2003.

D. Temple Lang, D. Swayne, H. Wickham, and M. Lawrence. *rggobi: Interface between R and GGobi*, 2007. URL http://www.ggobi.org/rggobi. R package version 2.1.7.

R. P. Thayer and R. F. Storer. Algorithm AS 21: Scale selection for computer plot. *Applied Statistics*, 18(2):206–208, 1969.

T. M. Therneau and B. Atkinson. *rpart: Recursive Partitioning*, 2008. R port by Brian Ripley. R package version 3.1-39.

M. Theus. Interactive data visualiating using Mondrian. *Journal of Statistical Software*, 7 (11):1–9, 2003.

L. Tierney. *Lisp-Stat: an object-oriented environment for statistical computing and dynamic graphics*. John Wiley & Sons, New York; Chichester, 1990.

E. Trauwaert, P. Rousseeuw, and L. Kaufman. Some silhouette-based graphics for clustering interpretation. *Belgian Journal of Operations Research, Statistics and Computer Science*, 29(3):35–55, 1989.

E. R. Tufte. *Envisioning information*. Graphics Press, Chesire, Connecticut, 1990.

E. R. Tufte. *Visual explanations*. Graphics Press, Chesire, Connecticut, 1997.

E. R. Tufte. *The visual display of quantitative information*. Graphics Press, Chesire, Connecticut, 2001.

E. R. Tufte. *Beautiful evidence*. Graphics Press, Chesire, Connecticut, 2006.

J. W. Tukey. *Exploratory data analysis*. Addison Wesley, 1977.

A. Unwin, C. Volinsky, and S. Winkler. Parallel coordinates for exploratory modelling analysis. *Computational Statistics & Data Analysis*, 43(4):553–564, 2003. URL http://rosuda.org/~unwin/AntonyArts/uvwCSDA.pdf.

A. R. Unwin, G. Hawkins, H. Hofmann, and B. Siegl. Interactive graphics for data sets with missing values - MANET. *Journal of Computational and Graphical Statistics*, 5(2): 113–122, 1996.

S. Urbanek. Exploring statistical forests. In *Proceedings of the Joint Statistical Meetings*, 2002a. URL http://simon.urbanek.info/research/pub/urbanek-jsm02.pdf.

S. Urbanek. Different ways to see a tree - klimt. In *Proceedings of the 14th Conference on Computational Statistics, Compstat 2002*, 2002b. URL http://simon.urbanek.info/research/pub/urbanek-cs02.pdf.

S. Urbanek. Interactive construction and analysis of trees. In *Proceedings of the Joint Statistical Meetings*, 2003. URL http://simon.urbanek.info/research/pub/urbanek-jsm03.pdf.

S. Urbanek. *Exploratory Model Analysis. An Interactive Graphical Framework for Model Comparison and Selection*. PhD thesis, Universität Augsburg, 2004.

S. Urbanek. Following traces of lost models. In *Proceedings of the Joint Statistical Meetings*, 2005. URL http://simon.urbanek.info/research/pub/urbanek-jsm05.pdf.

S. Urbanek and T. Wichtrey. *iplots: iPlots - interactive graphics for R*, 2007. URL http://www.iPlots.org/. R package version 1.1-1.

P. F. Velleman. *Data desk. The New Power of Statistical Vision*. Data Description Inc, 1992. URL http://www.datadesk.com/products/data_analysis/datadesk/.

P. F. Velleman. The philosophical past and the digital future of data analysis: 375 years of philosophical guidance for software design on the occassion of john w. tukey's 80th birthday. In *The practice of data analysis: Essays in honor of John W. Tukey*. Princeton University Press, 1997.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4. ISBN 0-387-95457-0.

H. Wainer. *Visual revelations: graphical tales of fate and deception from Napoleon Bonaparte to Ross Perot*. Lawrence Erlbaum, 2000.

H. Wainer. *Graphic discovery: A trout in the milk and other visual adventures*. Princeton University Press, 2004.

E. J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 85(411):664–675, 1990.

R. Wehrens and L. Buydens. Self- and super-organising maps in R: the kohonen package. *Journal of Statistical Software*, 21(5), 2007. URL http://www.jstatsoft.org/v21/i05.

H. Wickham. *classifly: Explore classification models in high dimensions*, 2007a. URL http://had.co.nz/classifly. R package version 0.2.3.

H. Wickham. *clusterfly: Explore clustering interactively using R and GGobi*, 2007b. URL http://had.co.nz/clusterfly. R package version 0.2.3.

H. Wickham. *ggplot2: An implementation of the Grammar of Graphics*, 2008. URL http://had.co.nz/ggplot2. R package version 0.6.

H. Wickham. *meifly: Interactive model exploration using GGobi*, 2006. URL http://had.co.nz/meifly. R package version 0.1.1.

H. Wickham. *reshape: Flexibly reshape data.*, 2005. URL http://had.co.nz/reshape/. R package version 0.7.1.

H. Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21 (12), 2007c. URL http://www.jstatsoft.org/v21/i12/paper.

H. Wickham. Meifly: Models explored interactively. Technical Report 4, Department of Statistics, Iowa State University, 2007d. URL http://www.stat.iastate.edu/preprint/articles/2007-04.pdf.

H. Wickham, M. Lawrence, D. Cook, A. Buja, H. Hofmann, and D. F. Swayne. The plumbing of interactive graphics. *Computational Statistics*, Accepted. Special issue for Proceedings of the 5th International Workshop on Directions in Statistical Computing.

L. Wilkinson. *The Grammar of graphics*. Statistics and Computing. Springer, 2nd edition, 2005.

L. Wilkinson, A. Anand, and R. Grossman. Graph-theoretic scagnostics. In *IEEE Symposium on Information Visualization*, pages 157–164, 2005.

Y. Xie. *animation: Demonstrate Animations in Statistics*, 2008. URL http://R.yihui.name. R package version 0.1-7.

W. Youden. Graphical diagnosis of interlaboratory test results. *Industrial Quality Control*, 15:24–28, 1959.

F. W. Young, R. A. Faldowski, and M. M. McFarlane. Multivariate statistical visualization. In C. Rao, editor, *Handbook of statistics*, volume 9, pages 959–998. Elsevier, 1993.