Conditional Visualisation for Statistical Models

Mark O'Connell

A thesis presented for the degree of
Doctor of Philosophy

Department of Mathematics and Statistics
National University of Ireland, Maynooth

Head of Department: Professor Stephen Buckley
Supervisors: Dr. Catherine Hurley & Dr. Katarina Domijan

February 2017

Abstract

It is difficult to understand data and statistical models in high-dimensional space. One way to approach the problem is conditional visualisation, but methods in this area have lagged behind the considerable advances in statistical modelling in recent decades. This thesis presents a new approach to conditional visualisation which uses interactive computer graphics, and supports the exploration of a broad range of statistical models.

The new approach to conditional visualisation consists of visualising a single low-dimensional section at a time, showing fitted models on the section, and enhancing the section by displaying observed data which are near the section according to a similarity measure. Two ways of choosing sections are given —choosing sections interactively using data summary graphics, and choosing sections programmatically according to some criteria.

The visualisations in this thesis necessitate interactive graphics, which are implemented in the condvis package in R.

Acknowledgements

First, and foremost, I must thank my parents, Séamus and Josephine, for their endless support and patience throughout my many years of full-time education.

I would like to thank my supervisors, Catherine and Katarina, for having the courage to take me on as a student, and the endurance to put up with me for four years! I have seen great development in myself over these years —as a person, a student, a statistician, a programmer— no small part of which is due to you and your guidance.

I would like to acknowledge the John and Pat Hume Scholarship, without the support of which, this thesis would have been impossible.

It would be remiss of me if I neglected to thank the wonderful support staff in the Department of Mathematics & Statistics, as well as the wider community of staff and students in the department, who create an enjoyable and productive learning environment.

Contents

List of Figures

List of Code Snippets

Chapter 1

Introduction

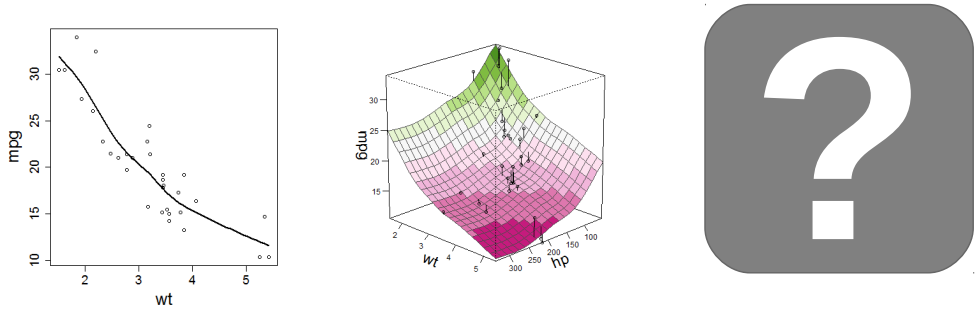
This thesis concerns visualisation for statistical models, and is chiefly motivated by the development of ever more complicated models over the last 50 years, and the apparent lag in the development of visualisation techniques to support the use of these models. With increasing emphasis being placed on prediction performance as opposed to inference, it has become common practice to trade interpretability for prediction performance when modelling data. In many cases, the only real contact an analyst has with a given model is through evaluation metrics, validation and testing results. It would seem a shame to develop a successful predictive model, and make no effort to interpret what it says about predictor effects. This work attempts to make the process of interpreting complicated models with high-dimensional inputs slightly less painful.

This thesis is accompanied by the condvis package in R; available from CRAN, with the most up-to-date version on Github. Most of the examples in this thesis will work with package version 0.3-x, but some require version 0.4-x.

1.1 Conditional visualisation

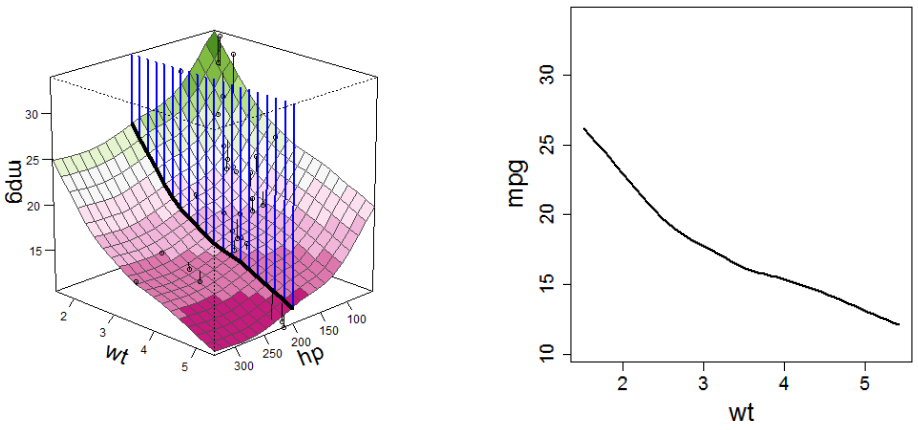
When a model consists of a single continuous predictor and a single response, the fitted model is simply visualised as a curve in two dimensions (Figure 1.1a). When a model involves two predictors, it may be visualised as a surface in three dimensions; either as a contour plot or a perspective mesh (Figure 1.1b). When a model involves more than two predictors, there is no direct way to visualise the model behaviour (Figure 1.1c). Clearly, there is a need for producing low-dimensional visualisations of models in high-dimensional space. One approach is conditional visualisation.

In a geometric sense, conditional visualisation means taking a section. Consider a simple model with two predictors, relating fuel efficiency (`mpg`) to car weight (`wt`) and horsepower (`hp`) in the `mtcars` data in R (R Core Team, 2015). The fitted model may be visualised as a surface as in Figure 1.2a. If we want to visualise the modelled effect of `wt` conditional on `hp`, we take a section. The intersection of the fitted model and the section is then a curve in two dimensions as in Figure 1.2b. In this sense,



(a) 1 predictor. (b) 2 predictors. (c) . . . more predictors.

Figure 1.1: Visualising fitted models. With one predictor, the model may be visualised as a curve. With two predictors, the model may be visualised as a surface in three dimensions. What can we do for more predictors?



(a) (b)

Figure 1.2: Visualising a section. (a) Visualising a 2 predictor model, and taking a section at `hp` = 200. (b) Visualising the section through model at `hp` = 200.

conditional visualisation offers a way to produce low-dimensional visualisations of models in high-dimensional space. It is important to note that such sections typically have no observed data lying on them (Figure 1.2b), and so it is difficult to understand how the observed data support the fitted model. In this work, we choose to display observed data points which are deemed to be *close* to the section (discussed further in Chapter 3).

1.2 Example: Forced Expiratory Volume data

The Forced Expiratory Volume (FEV) dataset in Kahn (2005) (originally in an earlier edition of Rosner (2010)) provides some useful discussion material for conditional relationships in statistical models. We use it here to launch into an example of conditional visualisation using the condvis package (O'Connell, 2016) in R, which

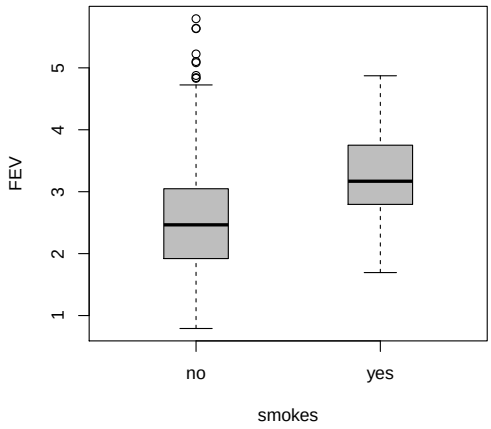


Figure 1.3: Boxplot of FEV versus smoking status. Suggests smoking is associated with higher FEV values.

forms the software implementation part of this thesis. The dataset concerns the relationship between lung health and smoking in children. The response is `fev` (the amount of air an individual can exhale in the first second of a forceful breath, used as a proxy for lung health), and the predictors are `gender`, `age`, `height` and `smokes` (binary smoking status).

In making a plot of FEV versus smoking status, we get our first surprise (see Figure 1.3). In the marginal view, it seems as though smoking is associated with better lung health! To illustrate the use of `condvis`, we fit a support vector machine (Smola and Vapnik, 1997). We then produce an interactive conditional expectation plot with `ceplot` and start looking at sections through the fitted model, focusing on the smoking status predictor. See Code Snippet 1.1 for the code to reproduce this example, and Chapter 4 for more on interactive conditional expectation plots.

Taking a section around `age = 14`, `height = 67`, with either gender, shows a more sensible result (see Figure 1.4. The user can choose this section by clicking on the condition selector plots on the right. Here, `gender = male`). In these parts of the predictor space, the fitted model suggests that smoking is associated with slightly lower FEV values. This is an example of Simpson's paradox, where the modelled conditional association is of opposite sign to the apparent marginal association. The observed data near these sections also seem to support the fitted model, although it is worth noting that, for each section, there are consistently more observations in the non-smoking group compared to the smoking group.

Taking a section around `age = 6`, `height = 55`, `gender = female`, the model is suggesting that smoking is related to higher FEV values (see Figure 1.5), as in the marginal view before! Why is this? On examining the section, we see there are no observed data points in the smoking group in this part of the data space. It is not surprising that there are no 6 year old smokers! Such a prediction for 6 year

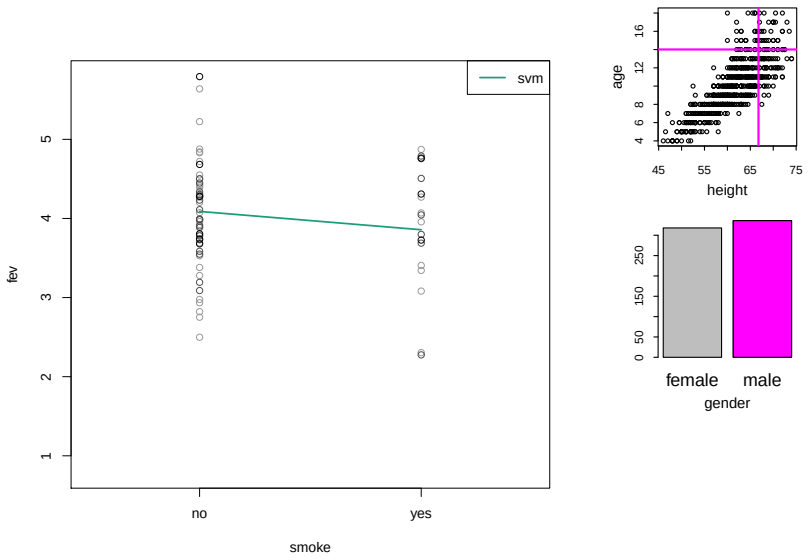


Figure 1.4: Section showing the modelled effect of smoking on FEV conditional on `height = 67`, `age = 14`, `gender = male`, shown in pink. Suggests that smoking is associated with lower FEV values, and observed data near this section seem to support this. From Code Snippet 1.1.

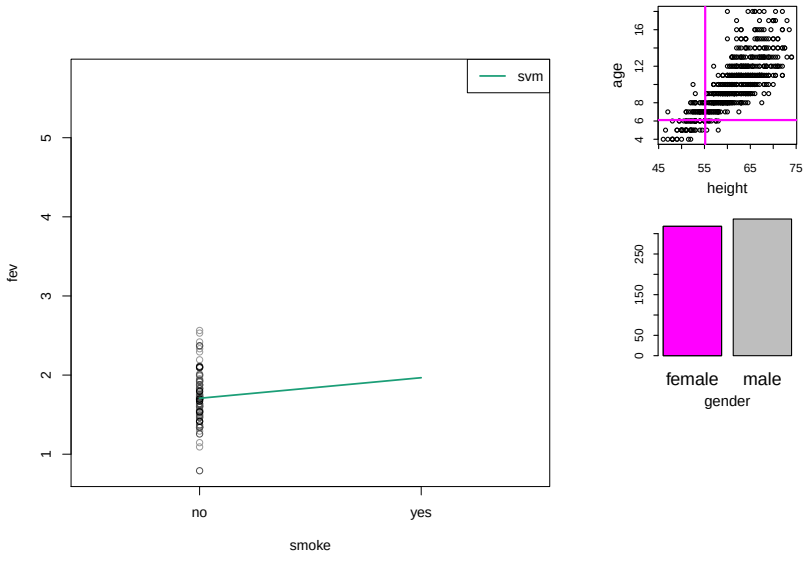


Figure 1.5: Section showing the modelled effect of smoking on FEV conditional on `height = 55`, `age = 6`, `gender = female`, shown in pink. Model suggests that smoking is associated with higher FEV values, but there are no observations in the smoking group near this section. From Code Snippet 1.1.

old smokers should clearly be considered an extrapolation and held in suitable suspicion accordingly. This example demonstrates how 'black-box' models can mislead analysts just as easily as more rigid linear models. A good defence against this is to take sections through the model and see how nearby observed data support the model.

```
library("condvis")
library("e1071")  # to fit model
library("covreg") # for FEV data
data(fev)

## Tidy up the data.

fev$male <- as.factor(c("female", "male")[fev$male + 1])
names(fev)[4] <- "gender"
fev$smoke <- as.factor(c("no", "yes")[fev$smoke + 1])

## Visualise marginal relationship of FEV with smoking.

plot(fev$smoke, fev$fev, xlab = "smokes", ylab = "FEV",
    col = "gray", boxwex = 0.35)

## Fit a support vector machine.

m <- list(svm = svm(fev ~ gender + smoke + age + height,
  data = fev))

## Create interactive conditional visualisation.

ceplot(data = fev, model = m, sectionvars = "smoke", type = "separate")
```

Code Snippet 1.1: FEV example. Resulting visualisations in Figures 1.4 and 1.5. Video of this example at youtu.be/rKFq7xwgdX0?t=270.

1.3 Contribution

The main research contribution of this thesis is a new, flexible approach to conditional visualisation for statistical models. This approach may be implemented using either an interactive or automated methodology. A large portion of this work has been implemented in the condvis package (O'Connell, 2016) in R, available from CRAN (Comprehensive R Archive Network). An introduction to condvis, and some content of Chapters 3 and 4 have been published in O'Connell et al. (2016).

1.4 Thesis outline

An outline of the proposed approach to conditional visualisation is presented in a flowchart in Figure 1.6. The thesis proceeds as follows:

- Chapter 2 discusses conditional visualisation in general and reviews some literature on the topic.
- Chapter 3 presents a new approach to conditional visualisation. This new approach involves visualising sections one at a time, showing where fitted models intersect the section, as well as nearby observed data.
- Chapter 4 introduces interactive conditional expectation plots. This is an implementation of the conditional visualisation described in Chapter 3, where sections are chosen interactively.
- Chapter 5 introduces the conditional tour, a framework for automatic exploration of sections in data space. This is an implementation of the conditional visualisation described in Chapter 3, where sections are chosen in advance.
- Chapter 6 presents real data applications of the proposed techniques for conditional visualisation.
- Chapter 7 gives a brief discussion on the potential of using conditional visualisation for model ensembles.
- Chapter 8 describes some of the software options available for programming interactive graphics, with a focus on the production of interactive graphics in R.
- Chapter 9 concludes the thesis and provides an outlook on further research possibilities.

A list of acronyms is provided at the end for the convenience of the reader. Appendix A describes a computer simulation which provides the data for one of the applications in Chapter 6. Appendix B contains R code to support the examples throughout the thesis. The workspace of each script in the appendix is saved to a .rda file named *'xxxx-workspace.rda'*, which is then loaded by the code snippets throughout the main text of the thesis. The scripts and previously saved workspace files are available for download at http://tinyurl.com/condvisthesis. Certain packages are required to exactly replicate the graphics produced in this thesis, particularly RColorBrewer (Neuwirth, 2014) for colour palettes, and gplots (Warnes et al., 2015) for two-dimensional histograms. Appendix C provides an excerpt from the documention for the condvis package.

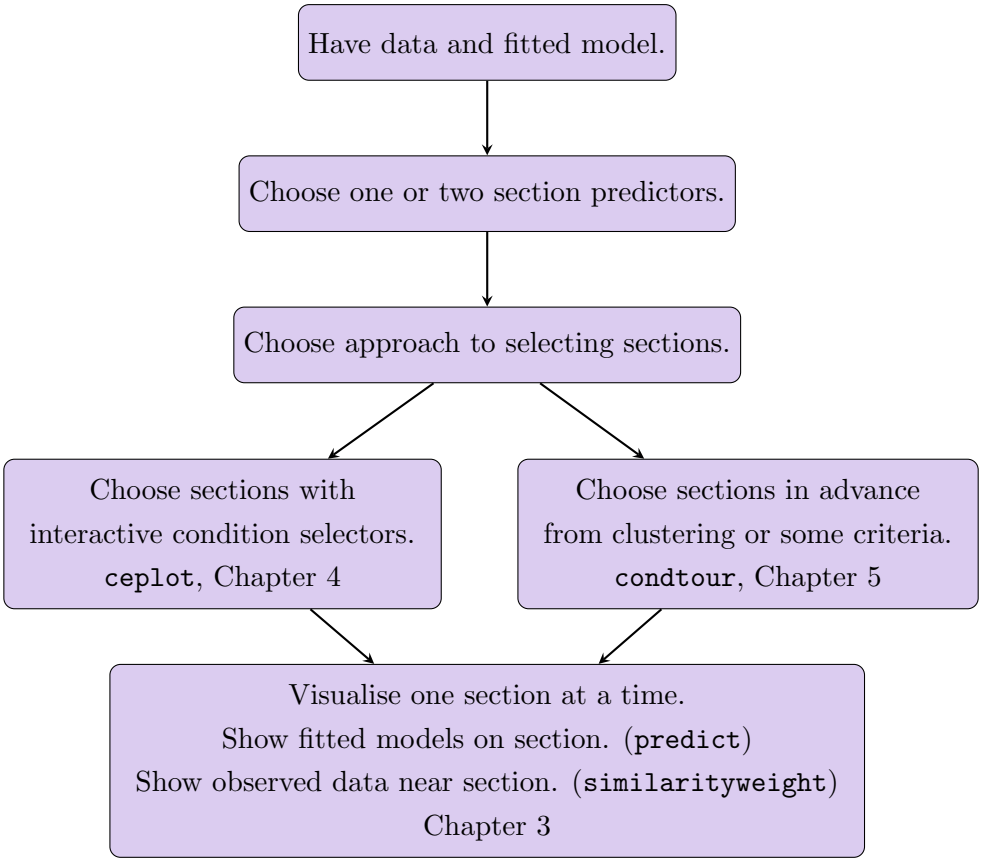


Figure 1.6: Overview of workflow in condvis. This flowchart summarises the new approach to conditional visualisation proposed in this thesis. Also shown is where the main functions of condvis: `ceplot`, `condtour`, and `similarityweight` fit into the process, as well as the chapter references for the relevant detailed discussions.

Chapter 2

Conditional visualisation

2.1 Introduction

This chapter explores the role of conditional visualisation in understanding and assessing statistical models. Some notation is introduced, and a brief review of relevant literature is made.

Chapter goal

The goal of this chapter is to give a brief summary of previous efforts in the area of conditional visualisation for data and models. Both research publications and software implementations are discussed. Basic notation is also introduced, which will be used throughout the remainder of the thesis.

Chapter outline

The remainder of this section describes the notation used throughout this thesis, and discusses the main difficulty with conditional visualisation. Section 2.2 discusses trellis graphics and related methods. Section 2.3 discusses partial residual plots and their long history in the literature. Section 2.4 discusses effect displays, which combine elements of trellis and partial residual plots. Section 2.5 discusses Individual Conditional Expectation plots and the Partial Dependence Plot. Section 2.6 concludes the chapter with a summary.

2.1.1 Notation

In this work, notation is taken from the Individual Conditional Expectation (ICE) plots of Goldstein et al. (2015), which is similar to that used by Friedman (2001) to describe the Partial Dependence Plot (PDP). Consider training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^N$, where $\boldsymbol{x}_i = (x_{i,1}, ..., x_{i,p})$ is a vector of predictors and y_i is the response. We construct a fitted model $\hat{f}$ that maps the predictors $\boldsymbol{x}$ to fitted responses $\hat{f}(\boldsymbol{x})$. Let $S \subset \{1, ..., p\}$ and let $C = \{1, ..., p\} \setminus S$, the complement set of S, and partition $\boldsymbol{x}$ into $\boldsymbol{x}_S$ and $\boldsymbol{x}_C$ accordingly. We may refer to $\boldsymbol{x}_S$ as *section predictor(s)* and $\boldsymbol{x}_C$ as

conditioning predictor(s). We have interest in observing the relationship between y (or $\hat{y}$, or $\hat{f}$) and $\boldsymbol{x}_S$, conditional on $\boldsymbol{x}_C$. In other words, we want to visualise the dependence of the observed response or fitted response on a subset of the predictors, conditional on the remainder of the predictors.

2.1.2 Difficulty with conditional plots

The crux of the problem in producing conditional plots lies in what is meant by conditioning. We take conditioning on $\boldsymbol{x}_C$ to mean setting each element of $\boldsymbol{x}_C$ to a certain value, defining a set of points in data space, which we call a section. In the case of $|S| = 1$, this section is a two-dimensional plane. If we want to visualise any data given this condition, we restrict ourselves to data points lying on this plane section. With the exceptions of well designed experiments or $\boldsymbol{x}_C$ being categorical data, most such sections contain no data points.

Some ways around this problem have been developed, most of which relax the conditioning requirement to one of approximate conditioning; conditioning on a neighbourhood, or subregion. Linked brushing (Stuetzle, 1987, 1991; Becker and Cleveland, 1987; Buja et al., 1991) allows the user to colour points in one view of the data, clearly defining an approximate conditioning in that view, which can then be seen in all other views where the conditioning would not otherwise be visible. Producing separate plots for different subregions of $\boldsymbol{x}_C$ results in the trellis graphics of Becker et al. (1996). Partial residual plots (Larsen and McCleary, 1972) visualise a section along a predictor in an additive model, and project all the data onto this single section. Effect plots (Fox, 1987) combine elements of trellis and partial residual plots. ICE plots (Goldstein et al., 2015) show sections through $\hat{f}$ at each value of $\boldsymbol{x}_C$ plotted against x_S. While not directly addressing conditional visualisation or statistical models, Furnas and Buja (1994) describe the use of sections and projections in visualising multivariate data. With the exception of linked brushing, all of these visualisation methods are static.

2.2 Trellis graphics

Trellis graphics (Becker et al., 1996) are a framework for the conditional visualisation of data and models. The framework comprises (in their own words) an "overall visual design, reminiscent of a garden trelliswork, in which panels are laid out into rows, columns, and pages. On each panel of the trellis, a subset of the data is graphed by a display method such as a scatterplot, curve plot, boxplot, 3-D wireframe, normal quantile plot, or dot plot. Each panel shows the relationship of certain variables conditional on the values of other variables." In the specific context of this work, we take this to mean separate plots of $(\boldsymbol{x}_S, y)$ made conditional (or approximately conditional) on $\boldsymbol{x}_C$ by displaying subsets of the data, where y (observed responses) may be substituted with $\hat{f}$ (a fitted model) or $\hat{y}$ (fitted responses, that is $\hat{f}$ evaluated

at observed predictor values).

When $\boldsymbol{x}_C$ are all categorical predictors, a panel is produced for each combination of the levels of $\boldsymbol{x}_C$. For continuous predictors in $\boldsymbol{x}_C$, the data are binned according to intervals which are allowed to overlap, and so the special term 'shingles' is used to describe the structure. The process continues as though the shingles were levels of a categorical predictor. The basic difficulty when using trellis for continuous variables is the choice of these shingles. An even grid of shingles can produce many panels containing little or no data. An alternative is to allow shingles to vary in size according to the data density across the predictor space, but this results in panels which may not be directly comparable because they represent subregions of different sizes. Becker et al. (1996) gives the 'equal count' algorithm to try obtaining shingle arrangements with as close to equal counts of observations in each shingle as possible. Anand and Talbot (2016) propose a method for the automatic choice of shingles for a trellis display using *scagnostics* (Wilkinson and Wills, 2008).

Trellis graphics are implemented in the lattice package (Sarkar, 2008) in R. The `coplot` function in graphics produces a trellis graphic. Trellis graphics are also used by higher level plotting functions found in ggplot2 (Wickham, 2009) and effects (Fox et al., 2016).

2.2.1 Example of trellis

Consider an exploratory analysis of the `mtcars` data in R, where we are interested in the effect of car weight ($x_S = \{\texttt{wt}\}$) on fuel efficiency ($y = \texttt{mpg}$) while taking account of the number of cylinders ($x_C = \{\texttt{cyl}\}$). We produce a trellis plot (Figure 2.1) using the `coplot` (Cleveland, 1993) function in the graphics package in R. The plot involves three panels, each being a scatterplot of `mpg` versus `wt`, conditional on a level of `cyl`. The trellis plot allows us to see the conditional distribution of `wt` and `mpg`, and in each panel the variability in `mpg` may be attributed to either `wt` or some other unobserved covariate, but not to `cyl`.

Suppose we are interested in the effects of `wt` and `qsec` (time to cover a quarter mile from a standstill) on `mpg`. We produce a trellis plot (Figure 2.2) as before, with the only difference being that `qsec` is continuous. As a result, each panel of the trellis is conditional on a range of `qsec`, or approximately conditional on `qsec`. As with the previous example, the trellis plot gives an impression of the conditional distribution of `wt` and `mpg`. In this instance, however, there may be variability in `mpg` attributable to `qsec` present in each panel due to the approximate conditioning. To eliminate this variability completely requires exact conditioning. As discussed in Section 2.1.2 however, this would necessitate potentially as many panels as observations, most of which would contain no more than a single data point. The approximate conditioning using shingles is a useful compromise. In this work, we intend to make use of a similar compromise with more flexibility.

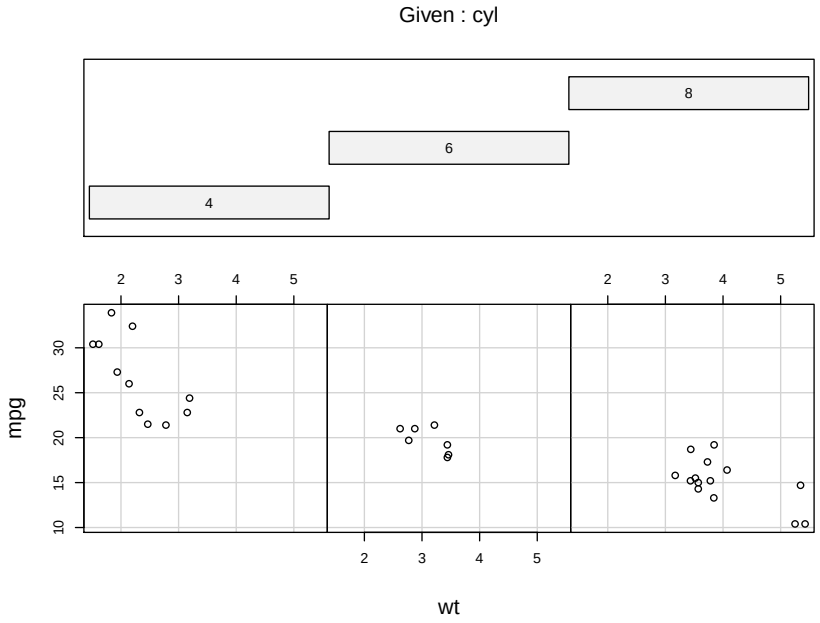


Figure 2.1: Trellis plot of `mpg` versus `wt`, conditional on `cyl`. The conditioning is exact because `cyl` is categorical.

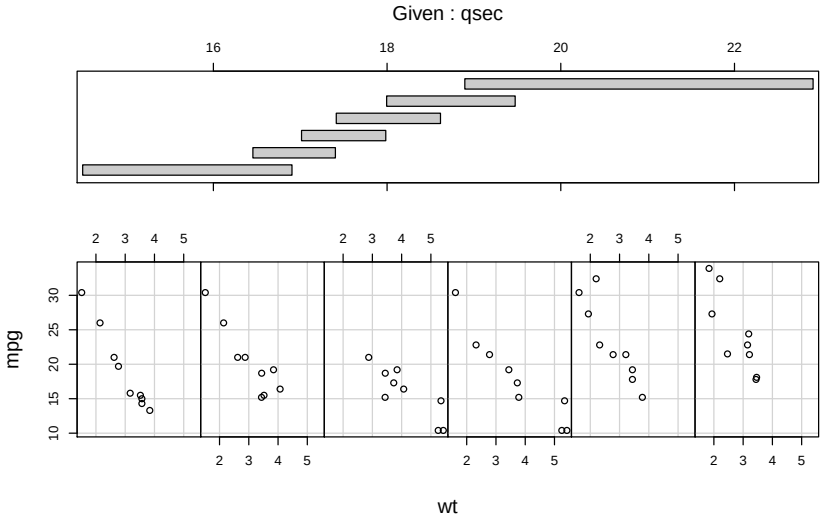


Figure 2.2: Trellis plot of `mpg` versus `wt`, conditional on `qsec`. The conditioning is approximate because `qsec` is continuous. Note the overlapping shingles represented in the upper half of the graphic, particularly the shingle for the far-right plot, encompassing almost half the range of `qsec`.

2.2.2 An interactive approach to trellis

There is a clear difficulty when conditioning plots on continuous predictors with trellis graphics. It seems that even with careful thought, any single choice of shingles may be unsatisfactory for some reason or another; either a lack of data to display in each panel, or a single panel representing a disproportionately large sub region of $\boldsymbol{x}_C$. The problem is that there are infinitely many choices of shingles, their size, their overlaps, and the number of them. Yet, we attempt to select just one set of shingles and produce a single plot, as in Figure 2.2.

This thesis proposes a more flexible approach to conditional visualisation, producing a single plot which we interactively (or programmatically) condition on $\boldsymbol{x}_C$. This allows us to visualise many more of the potential conditionings mentioned above in a very practical way. For example, the trellis plot in Figure 2.2 shows six static panels each approximately conditioned on a continuous variable. Using an interactive or animated approach, we can visualise several such conditional plots in a matter of seconds. Interactively visualising sections one at a time sacrifices direct comparison of different sections, but if done smoothly, we can observe how the section changes as we change the conditioning. See Chapter 3 for a more complete discussion.

2.2.3 CARTscans

CARTscans (Nason et al., 2004) are a trellis of coloured contour plots of the fitted regression $\hat{f}$ conditioned on $\boldsymbol{x}_C$. A CARTscan is used in a situation where we want to take a section on two predictors ($|S| = 2$). The ideal CARTscan has $|C| = 2$, where $\boldsymbol{x}_S$ are two continuous predictors (termed inner variables), and $\boldsymbol{x}_C$ are two categorical predictors (termed outer variables), where ideal means that the conditioning is exact and the entire graphic can be reasonably produced on one page or computer screen. A contour plot of $(\boldsymbol{x}_S, \hat{f})$ is then produced for each unique combination of levels of $\boldsymbol{x}_C$ in a rectangular grid. On each contour plot, the $\boldsymbol{x}_S$ coordinates of the observed data may be plotted. CARTscans are named for their similarity to images from computerised axial tomography, and their application to the Classification and Regression Trees (CART) of Breiman et al. (1984).

As with trellis, if $\boldsymbol{x}_C$ contains continuous predictors, Nason et al. (2004) resort to binning and producing an average $\hat{f}$ over the relevant subspaces. If $|C| > 2$, they apply the trellis technique recursively, resulting in multiple CARTscans of the basic type with $|C| = 2$, each of which are made conditional on further predictors. Nason et al. (2004) acknowledge that this solution works well with binary predictors in $\boldsymbol{x}_C$, but is not very practical otherwise. Another approach given in their paper is to order the predictors according to some importance measure, take the top four as $\boldsymbol{x}_S$ and $\boldsymbol{x}_C$, and take the $\hat{f}$ averaged over the remaining predictors.

To my knowledge, there is no software implementation of CARTscans available at the time of writing.

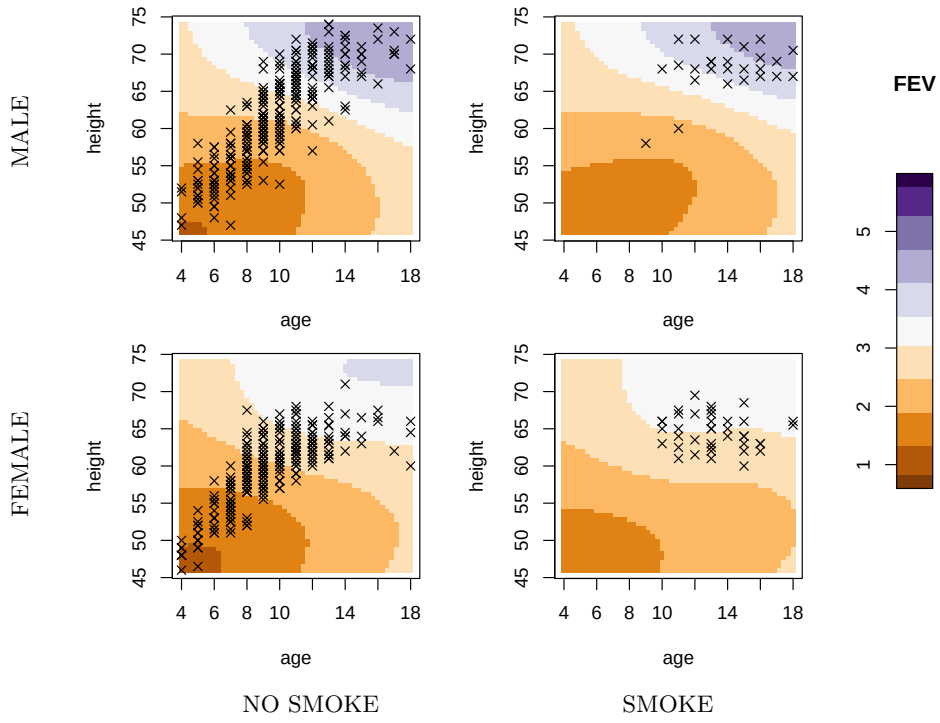


Figure 2.3: CARTscan of support vector machine fitted to the FEV data. Outer variables are smoking status and gender. Inner variables are height and age. The contours show the different modelled effects for each combination of conditioning predictors, and we also get an impression of the conditional distribution of observed data.

2.2.4 Example of CARTscans

Consider again the FEV data referred to in Section 1.2. This is the same data used by Nason et al. (2004) to demonstrate CARTscans, and we use it here to visualise a support vector machine fit to the data (Figure 2.3). Each panel of the plot shows the expected response as a coloured contour map, conditional on smoking status and gender. The plotted points show the coordinates of observed data. We can see that there are less observations in the smoking group than the non-smoking group, and that height and age are correlated.

2.3 Partial residual plots

In general, a partial residual plot (Larsen and McCleary, 1972) displays the pairs

$$(x_S,\ \hat{f}_S(x_S) + \mathrm{e}) \tag{2.1}$$

for a fitted additive regression model $\hat{f}$, where $\hat{f}(\boldsymbol{x}) = \sum_j \hat{f}_j(\boldsymbol{x}_j)$, $\hat{f}_S$ denotes the component(s) of $\hat{f}$ involving x_S, and e denotes the residuals from the model. The main purpose of these plots is

1. to assess the appropriateness of the curvature of $\hat{f}_S$,

2. to aid in the understanding of the effect of x_S on $\hat{f}$, and

3. to observe the distributional properties of x_S, particularly in relation to the two previous points.

The function $\hat{f}_S(x_S)$ is often superimposed on the partial residual plot, to show the fitted model, conditional on $\boldsymbol{x}_C$. Regarding item 3 above, partial residual plots can afford the analyst some sense of the values of x_S for which model predictions should be considered extrapolation or interpolation. In standard partial residual plots, this usefulness diminishes with higher dimensionality of $\boldsymbol{x}_C$, where even the concept of interpolation and extrapolation is not straightforward.

Cook (1993) points out that the partial residual plot can be misleading if the model is misspecified for the effects of $\boldsymbol{x}_C$ (to use the notation of this work). The reason for this may be seen if we rewrite (2.1) as

$$(x_S,\, y - \hat{f}_C(x_C)) \tag{2.2}$$

and so the partial residual plot clearly depends on $\hat{f}_C$. Here, $\hat{f}_C(x_C)$ may refer to $\sum_{j \in C} \hat{f}_j(x_j)$ in an additive model.

Partial residual plots are implemented in the `plot.gam` methods for additive models from the gam (Hastie, 2013) and mgcv (Wood, 2015) packages. Some plots produced by the effects (Fox et al., 2016) package and the visreg (Breheny and Burchett, 2016) package involve partial residual plots.

Partial residual plots do not provide an extension to non-additive models, and rely on the conditioning predictors being modelled well to produce meaningful plots. The approach to conditional visualisation presented in this thesis aims to overcome both of these problems.

2.3.1 Literature

The term partial residuals appears to have its first use in Larsen and McCleary (1972), but they point out (subsequently to the article being accepted) that the basic idea had been previously published in Ezekiel and Fox (1959). Ezekiel (1924) arguably contains the earliest appearance of partial residual plots, where they are referred to as "dot charts showing relations of dependent variable [with independent variables], adjusted for approximate curvilinear effects of other variables," and are used as an aid to an iterative method of fitting several non-linear effects simultaneously. Cook (1998) provides a discussion of the origins of partial residual plots referring to work by Ezekiel (1924) and Bean (1929, 1930).

Larsen and McCleary (1972) point out the use of residual versus predictor plots, that is plots of the pairs (x_S, e), for

1. detection of outliers

2. assessing the presence or absence of inhomogeneity of variance, and

3. determining if a transformation or basis expansion is needed.

and go on to describe two further applications of partial residual plots:

4. assessing the importance (in terms of predicting power) of a predictor in the presence of all other predictors

5. assessing the importance of the non-linearity, if any, and helping the choice of a suitable transformation.

However, they concede that item 4 above may not be a fruitful endeavour, especially when there are large correlations between the predictor used in the plot and other predictors in the model (between x_S and $\boldsymbol{x}_C$ in the notation of this work).

Wood (1973) argues that "none of the forementioned [uses of partial residuals] has discussed dealing with problems that exhibit distributional pecularities [sic] in x_i," and uses the term "component-plus-residual" rather than partial residual. He summarises the applications of the partial residual plot as an aid

1. to choose an appropriate form of the equation

2. to observe the distribution of observations over the range of each predictor

3. to estimate the influence of each observation on each component of the equation.

Partial regression plots (Mosteller and Tukey, 1977) are similar to partial residual plots in that both show the same slope and residuals (in the sense of a regression on the plot itself), but differ in their construction and display. The partial regression plot displays the pairs $(\hat{r}_y, \hat{r}_x)$, where $\hat{r}_y$ are the residuals from regressing y on $\boldsymbol{x}_C$, and $\hat{r}_x$ are the residuals from regressing x_S on $\boldsymbol{x}_C$. Henderson and Velleman (1981) discuss the use of partial regression (also known as added variable) plots in building multiple regression models interactively. They briefly mention the use of partial residual plots for identifying suitable solutions to problems of non-linearity in predictor effects.

Atkinson (1982) reviews the use of partial residual plots and partial regression plots in assessing constructed variables and transformations, where they can be used to investigate the effect of adding a new predictor to a multiple regression. In a comment on the paper, Welsch (1982) draws a distinction between partial regression and partial residual plots, saying: "The choice about which partial plot to use should most certainly *not* be made on computational grounds. Partial residual plots seem to be better for the analysis of transformations, while partial regression plots help more with leverage and influential data."

Augmented partial residual plots from Mallows (1986) are partial residual plots with a quadratic term for x_S included in the regression to encourage non-linear effects to exhibit themselves in the plot. The motivation is to detect non-linearity with minimal computation, comparing partial residual plots with and without the

quadratic term, instead of examining all the possible basis expansions or transformations. Johnson and McCulloch (1987) discuss both partial residual plots and partial regression plots. They advocate the use of more flexible models in constructing partial residual plots.

O'Hara Hines and Carter (1993) suggest improved partial residual plots and partial regression plots for generalised linear models mainly concentrating on using partial regression plots for assessing influence. In a comment by Landwehr and Pregibon (1993), the use of partial residual plots for examining non-linearities is stressed, as opposed to being used as a cheap (and less effective) alternative to partial regression plots for assessing influence.

Cook (1993) discusses partial residual plots, placing emphasis on detecting curvature, and warning against the temptation to use them as omnibus diagnostic plots (e.g., for assessing influential observations, or the strength of a predictor effect). He points out that the plot can not only show the relationship between the fitted model and the predictor, but the relationship between the errors and the predictor, should one exist. He also introduces CERES (Combining Conditional Expectations and Residuals) plots for situations where the usual linear model may be poorly specified, and hence the conditioning provided by traditional partial residual plots may be misleading.

2.3.2 Example of partial residual plot

Consider again the exploratory analysis of the mtcars data in Section 2.2.1. Suppose we continue the analysis by fitting an additive model (Hastie and Tibshirani, 1990) relating `mpg` to `wt` and `cyl` (as categorical, with levels 4, 6 and 8), with a smoothing spline term for `wt`.

$$y = \beta_0 + \beta_1 x_{\text{cyl6}} + \beta_2 x_{\text{cyl8}} + f(x_{\text{wt}})$$

We produce a partial residual plot for `wt` as in Figure 2.4. This allows us to see the estimated effect of `wt` on `mpg`, controlled for the effect of `cyl` (insofar as the model can account for this effect). The problem of a partial residual plot relying on reasonable modelling of the effects of $\boldsymbol{x}_C$ (`cyl` in this case) is documented by Cook (1993). Becker et al. (1996) points out the benefits of using the trellis technique on partial residual plots, and given our exploratory analysis, it seems only natural to apply it to our partial residual plot. The partial residual plot conditions on $\boldsymbol{x}_C$ through a fitted model, and the trellis technique allows us to investigate this conditioning further.

We can apply trellis conditioning to a partial residual plot using the visreg package (Breheny and Burchett, 2016). Figure 2.5 shows the partial residual plot from Figure 2.4 decomposed into panels for each level of `cyl`. This conditional version of the partial residual plot makes two things clear. Firstly, there is a bivariate relationship between `wt` and `cyl`. This means that interpreting the smoothing spline

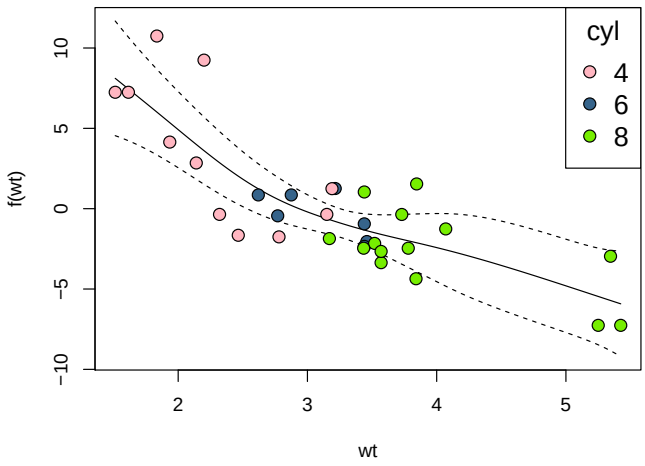


Figure 2.4: Partial residual plot for `wt` from the model in Section 2.3.2. The fitted model is shown on the plot as the solid line, with the dashed lines showing ±2 pointwise standard errors for the fitted mean. Levels of `cyl` are given by colour.

across the full range of `wt` as simply being the effect of `wt` may not be appropriate, and rather implies an extrapolation which may not be justified. This is not so clear from the original partial residual plot in Figure 2.4. Secondly, we can see that the confidence bounds are actually different for each level of `cyl`. This is certainly not obvious from the original partial residual plot. However, the trellis conditioning does not improve our direct understanding of the modelled effect of `wt` compared to the partial residual plot. This is an additive model, and so the modelled effect is the same in each panel, except for a different intercept.

2.4 Effect displays

Effect displays (Fox, 1987, 2003) visualise the response surface of complex additive parametric regression models by conditioning and slicing the surface, producing a sequence of two-dimensional line graphs. In a basic sense, effect displays apply trellis graphics to fitted additive models in a similar way to CARTscans (Section 2.2.3) and the example from Section 2.3.2. Effect displays are implemented in the effects package (Fox et al., 2016). Similar displays can be produced with the visreg package (Breheny and Burchett, 2016), with the chief difference being that medians are plugged into the model formula for ranges of continuous predictors in $\boldsymbol{x}_C$ where an effects plot would average over them. The visualisations provided by visreg are not limited to additive models.

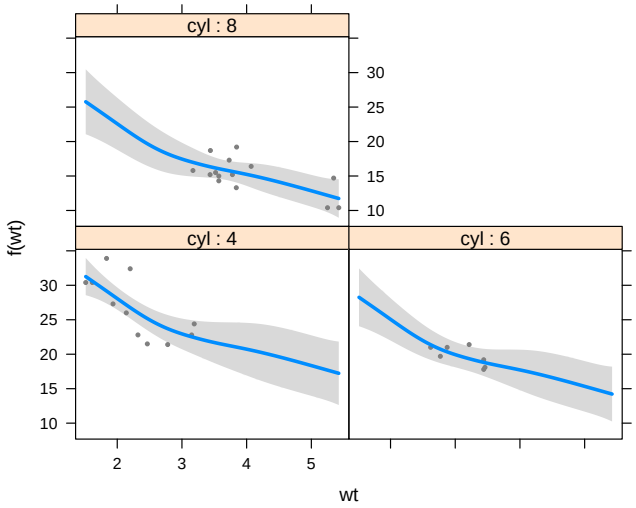


Figure 2.5: Trellis graphic where the partial residual plot from Figure 2.4 is conditioned on `cyl`. Produced by visreg package.

2.4.1 Effect displays with partial residuals

The functionality to add partial residuals to effect displays was added to the effects package in version 3.0-0 in March, 2014.

2.4.2 Example of effect display

Consider again the FEV data from Section 1.2. We now fit a linear model with all two-way interactions (in R, `fev ~ .*.`) in order to demonstrate the use of an effect display. Effect displays are not capable of visualising non-additive, 'black-box' models such as radial kernel models or gradient boosted trees at the time of writing. Figure 2.6 shows an effect display for the effect of smoking on FEV according to the linear model. As with our interactive approach for the support vector machine in Section 1.2, this display reveals that the bivariate relationship of lung health and smoking was misleading. This graphic lacks observed data, however, which makes it more difficult to understand the apparently 'positive' effects of smoking in the left-most panels. The wider confidence intervals for the smoking group at young ages hint at the lack of observed data, but visualising the observed data directly on the panels would make this more clear. It is worth noting that that partial residuals could indeed be added to this plot, and are likely planned for inclusion in such an effect display by the authors of the effects package.

2.5 ICE plots and PDPs

Section 2.3 discusses the use of partial residual plots for additive models. These plots can only be produced if $\hat{f}(\boldsymbol{x})$ can be decomposed into $\hat{f}_C(\boldsymbol{x}_C) + \hat{f}_S(\boldsymbol{x}_S)$, and

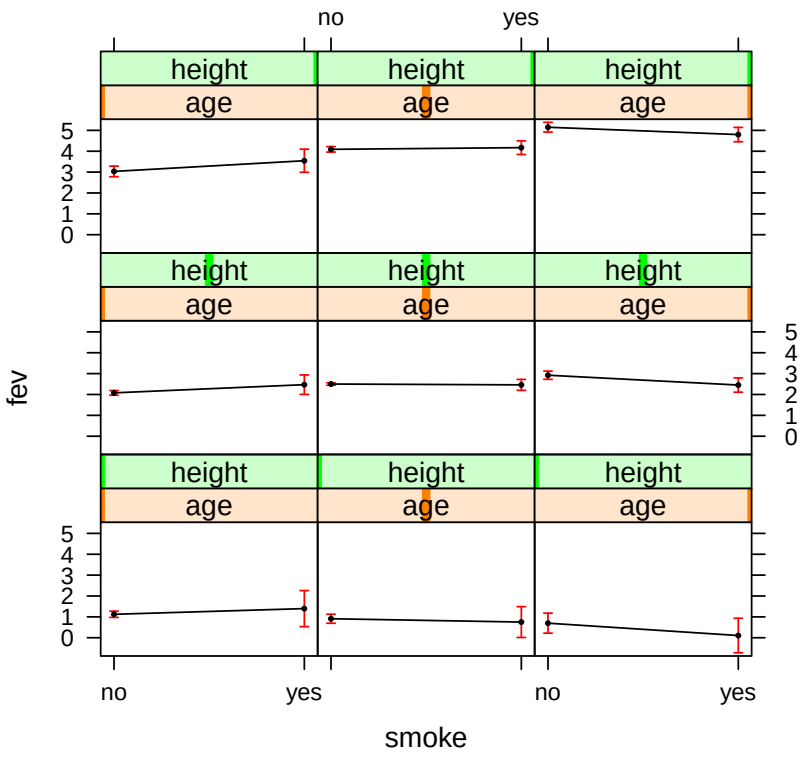


Figure 2.6: Effect display showing the effect of smoking from a linear model with interactions, conditioned on age and height. Produced by effects package.

are usually employed in cases where $\hat{f}(\boldsymbol{x}) = \sum_j \hat{f}_j(x_j)$. Clearly then, partial residual plots are not of any use in examining the non-additive models of modern machine learning techniques, such as gradient boosted trees (Friedman, 2001). Some non-additive alternatives to partial residual plots may be found in Individual Conditional Expectation plots (Goldstein et al., 2015) and Partial Dependence Plots (Friedman, 2001). These plots have some goals in common with partial residual plots, and are presented here in reverse chronological order for clarity of exposition. The common goals with partial residuals plots are the visualisation of the curvature of $\hat{f}$ with respect to x_S, and hence understanding of the effect of x_S, as well as the distribution of x_S.

2.5.1 Individual Conditional Expectation plots

Individual Conditional Expectation (ICE) plots, introduced by Goldstein et al. (2015), visualise sections through fitted models. A section is defined by rewriting $\hat{f}(\boldsymbol{x})$ as $\hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_C)$, considering $\boldsymbol{x}_S$ to be free variables and $\boldsymbol{x}_C$ to be fixed parameters of each section. By setting $\boldsymbol{x}_C$ to observed values, we can obtain functions which define sections through the fitted model at each observed data point, $\hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_{Ci})$. For the purpose of visualisation, these functions are evaluated at various values of

$\boldsymbol{x}_S$. Two choices for these values are the observed values of $\boldsymbol{x}_S$, or a grid of values in the range of $\boldsymbol{x}_S$. For each plotted section, the associated observation is plotted as $(x_S, \hat{y})$

The central purpose of ICE plots is a basic understanding of the fitted model. Other uses include exploring how the fitted model extrapolates in the data space, and assessing whether modelled effects might be additive. Goldstein et al. (2015) also present a few variations on the basic ICE plot, namely a centred ICE plot and a derivative ICE plot. A centered ICE plot vertically translates each section to place them on top of one another inasmuch as is possible, with the goal of making certain patterns more visually obvious. A derivative ICE plot shows an approximate first derivative of $\hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_{Ci})$ with respect to $\boldsymbol{x}_S$, and so attempts to highlight important regions of curvature in the fitted model. Goldstein et al. (2015) also point out that the observations or sections can be coloured in ICE plots to allow the display of a further dimension in the plot.

ICE plots are implemented in the ICEbox (Goldstein et al., 2016) package in R.

2.5.2 Partial Dependence Plots

Following the notation of Goldstein et al. (2015), Friedman's (2001) partial dependence function is defined as

$$f_S = \mathbb{E}\left[f(\boldsymbol{x}_S, \boldsymbol{x}_C)\right] = \int f(\boldsymbol{x}_S, \boldsymbol{x}_C) \mathrm{dP}(\boldsymbol{x}_C) \tag{2.3}$$

giving the average value of f when $\boldsymbol{x}_S$ is fixed and $\boldsymbol{x}_C$ varies over its marginal distribution $\mathrm{dP}(\boldsymbol{x}_C)$. Since f and $\mathrm{dP}(\boldsymbol{x}_C)$ are not known, the integral in Equation 2.3 is estimated by

$$\hat{f}_S = \frac{1}{N} \sum_{i=1}^{N} \hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_{Ci}) \tag{2.4}$$

where $\boldsymbol{x}_{Ci}$ are the observed values of $\boldsymbol{x}_C$ for the ith observation. A Partial Dependence Plot (PDP) visualises $\hat{f}$ against $\boldsymbol{x}_S$. From Equation 2.4, a PDP may be considered a simple average of the sections defined for an ICE plot. The main goal of the PDP is to show the average effect of a predictor in a 'black-box' model. The main drawback with the PDP is that it does not represent predictor effects well in situations where the curvature of the effect varies significantly throughout the predictor space. This seems to be a considerable drawback, because 'black-box' models are often employed in situations where their flexibility in modeling complex predictor interactions is required.

PDPs are implemented alongside ICE plots in the ICEbox (Goldstein et al., 2016) package. The `partialPlot` function in randomForest (Liaw and Wiener, 2015) produces a partial dependence plot for a random forest model.

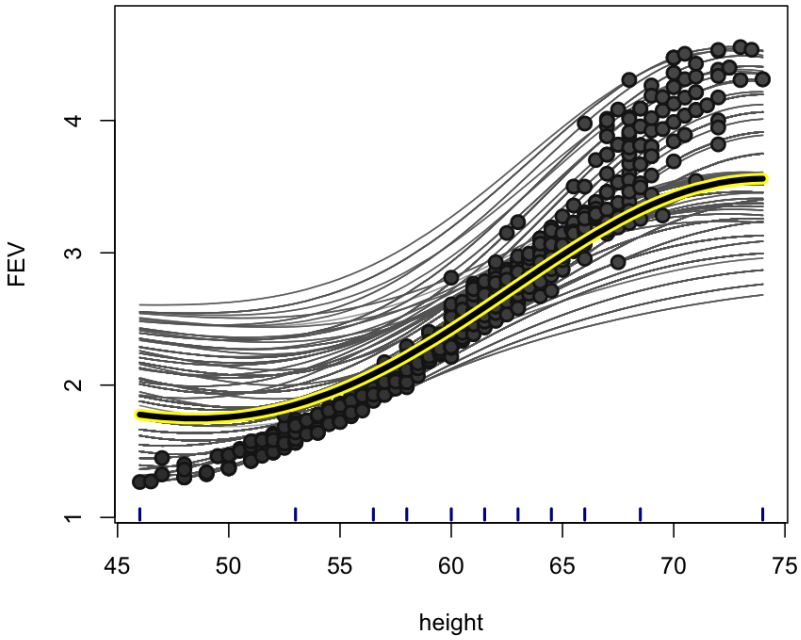


Figure 2.7: ICE plot of support vector machine fit to FEV data, with the PDP shown as a thick line highlighted in yellow. Shows the modelled effect of `height` on a sample of sections at observations. Produced with the ICEbox package.

2.5.3 Example of ICE plot and PDP

Consider again the FEV example from Section 1.2 and the CARTscans example in Section 2.2.3. Figure 2.7 shows an ICE plot of the same support vector machine model. In this case, we are visualising the effect of `height` controlled for the other predictors. Each grey curve represents a section through the fitted model at an observed data point. The thick line highlighted in yellow is the average of these curves, known as the PDP. From the sections in the ICE plot, we can conclude that the modelled effect of height on most sections is an increasing function with a plateau at higher values, but it is very difficult to tell if these effects are supported by the observed data. This is a disadvantage of ICE plots, that they do not show how curves relate to the conditioning predictors $\boldsymbol{x}_C$. The curves can be coloured, but this only allows one dimension of $\boldsymbol{x}_C$ to be represented.

Consider now the additive model we used to demonstrate a partial residual plot in Section 2.3.2. An ICE plot of this model (Figure 2.8), taking sections along the `wt` predictor, shows only three unique curves; the same smoothing spline with three different intercepts for different levels of `cyl`.

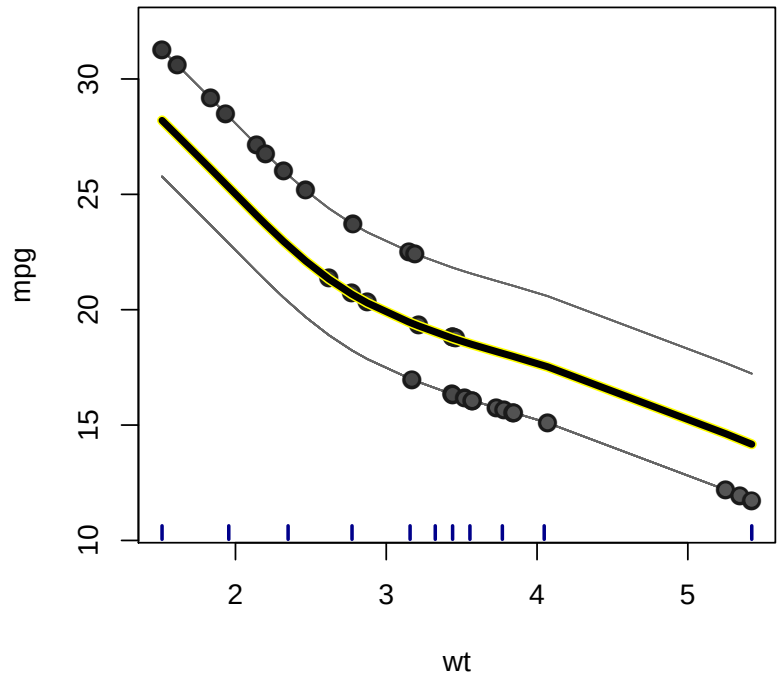


Figure 2.8: ICE plot of additive model fit to `mtcars` data, with the PDP shown as a thick line highlighted in yellow. Shows the modelled effect of `wt` on sections at each observation, of which there are only three unique curves. Produced with the ICEbox package.

2.6 Chapter summary

We split the predictor vector into section predictors $\boldsymbol{x}_S$ and conditioning predictors $\boldsymbol{x}_C$. We are particularly interested in visualising the response or fitted model against the section predictors given some value of the conditioning predictors. Trellis graphics provide a reasonable way to produce conditional plots, but come with the difficulty of choosing the conditioning for the panels. CARTscans apply trellis to fitted models. Partial residual plots provide a single, relatively straightforward graphic for each predictor (occasionally two predictors) in an additive model. Effect displays apply trellis to parametric models and combine partial residuals with the plots. ICE plots and the PDP provide a similar type of plot for more flexible models without the additive constraint.

Chapter 3

A new approach to conditional visualisation

3.1 Introduction

This chapter introduces a new approach to conditional visualisation. This approach involves taking sections in data space, visualising models where they intersect the section and displaying observed data points which are near the section. The proximity of data to the section is quantified using distance measures.

This method may be considered mainly as a flexible adaptation of trellis graphics (Becker et al., 1996), but it also draws on aspects of Nason et al. (2004) and Furnas and Buja (1994). Whereas methods for conditional visualisation in the past have been developed for the printed page, this new approach necessitates computer graphics, making use of interactivity and animation. The method works for classical statistical models, Bayesian models, and modern machine learning methods. Two approaches to the implementation of the ideas explored in this chapter are discussed in Chapters 4 and 5. These implementations can be found in my R package condvis, available from CRAN and hosted at github.com/markajoc/condvis. Examples of the application of these methods are presented in Chapter 6.

Chapter goal

The goal of this chapter is to introduce a novel approach to conditional visualisation.

Chapter outline

Section 3.2 discusses how to choose a section to visualise. Section 3.3 discusses how to visualise fitted models on the chosen section. Section 3.4 discusses how to display observed data in the section visualisation. Section 3.5 gives an example of visualising a section and nearby observed data. Section 3.6 describes how the ideas in this chapter can be applied to modelling problems. Section 3.7 describes the software

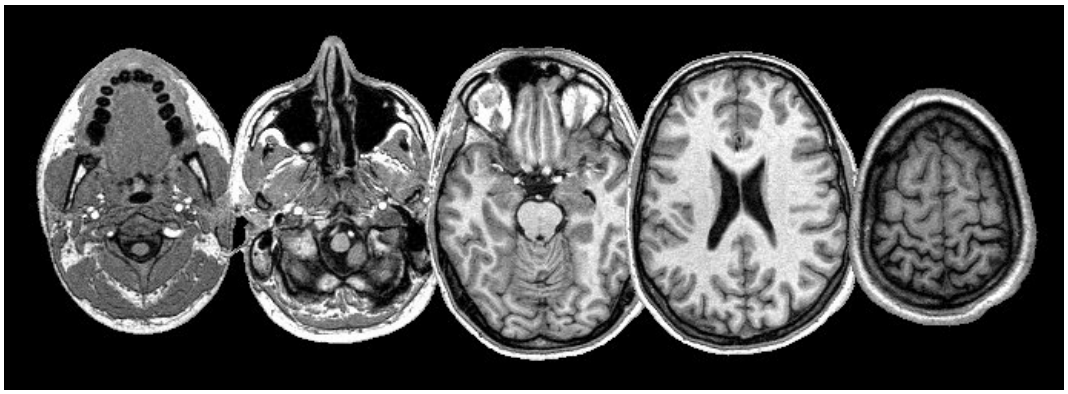


Figure 3.1: Example of sections in medical imaging. Five slices from an MRI scan of a human head. (Genesis12 Wikipedia contributor, reproduced under Creative Commons Licence CC-BY-SA-2.5)

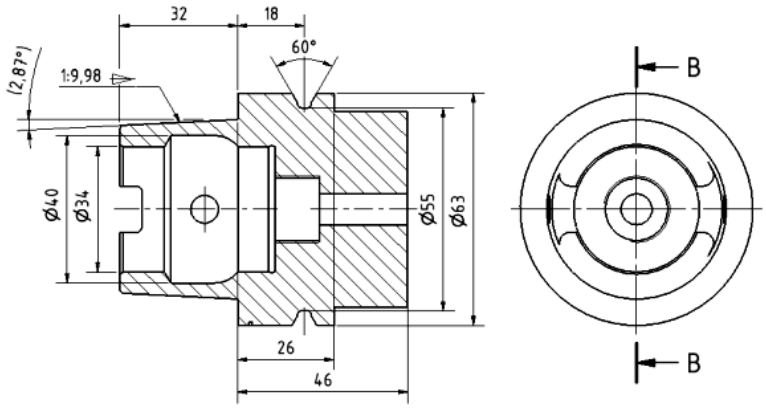


Figure 3.2: Example of a section view of a machine part in technical drawing. (Sven Gleich, reproduced under Creative Commons Licence CC-BY-SA-2.5.)

implementation of section visualisation in my R package condvis. Section 3.8 gives a summary of the chapter.

3.2 Choosing a section

Sections are used in many fields to reveal detail that would otherwise be obscured —in the study of anatomy, in medical imaging (Figure 3.1), and in engineering drawings (Figure 3.2). Sections are also useful in connecting simple geometric shapes to their analogues in higher dimensions. For example, the intersection of a disk and a plane (other than the plane in which the disk lies) gives a line segment. Similarly, the intersection of a sphere and a plane gives a disk. Continuing with this idea allows us to consider a hypersphere in terms of spheres.

Sections may not capture every important aspect of a high-dimensional object, but in return for this weakness, sections are typically quite simple to visualise and understand. Consider Figure 3.1 and the difficulty in creating a suitable 3-D visualisation of the cranium which would still provide a detailed view of its interior. In contrast, the 2-D sections are trivial to produce as monochrome images, and are relatively easy to understand.

In this work, we restrict ourselves to conditioning on all but one or two predictors, and so visualising two-dimensional and three-dimensional sections. It is, of course, possible to consider a section of dimension higher than three, and consider the intersection of yet higher-dimensional shapes with this section. Such high-dimensional sections are not the focus of this work, but it may be feasible to employ other multivariate visualisation techniques to visualise these sections —for example, projection tours (Asimov, 1985).

The general goal of taking a section may be summarised as reducing the amount of data to be visualised in order to reveal details that would otherwise be obscured.

3.2.1 Defining a section in data space

A section in the directions of $\boldsymbol{x}_S$ and y in the data space is defined by a single point in the space of $\boldsymbol{x}_C$. Considering a section as a point simplifies the treatment of sections so that we can consider simple distance measures between sections and observed data points, and make use of functions which take points in the space of $\boldsymbol{x}_C$ as input.

3.3 Displaying fitted models

In this work, fitted models are visualised on two-dimensional and three-dimensional sections only. Figure 3.3 shows how the basic section visualisations in condvis display fitted models. Regression models (top row of Figure 3.3) are represented as curves or surfaces (either a perspective view, or a colour map), and classifiers are visualised as colour maps (bottom row of Figure 3.3). Two broad types of regression models are considered: classical models, where the fitted model ($\hat{f}$) is determined by fixed parameter values (maximum likelihood estimates, for example), and Bayesian models, where $\hat{f}$ is determined up to probability distributions on its parameters.

3.3.1 Classifiers

For classification problems, we follow Hastie et al. (2009) in visualising a classifier as assigning colours to regions of the predictor space. When considering binary class probabilities, we can proceed as for a regression model below. If we are interested in multi-class probabilities (see Section 6.3), we can produce a grid of barcharts similar to embedded plots (Grolemund and Wickham, 2015) instead of the basic colour map in Figure 3.3.

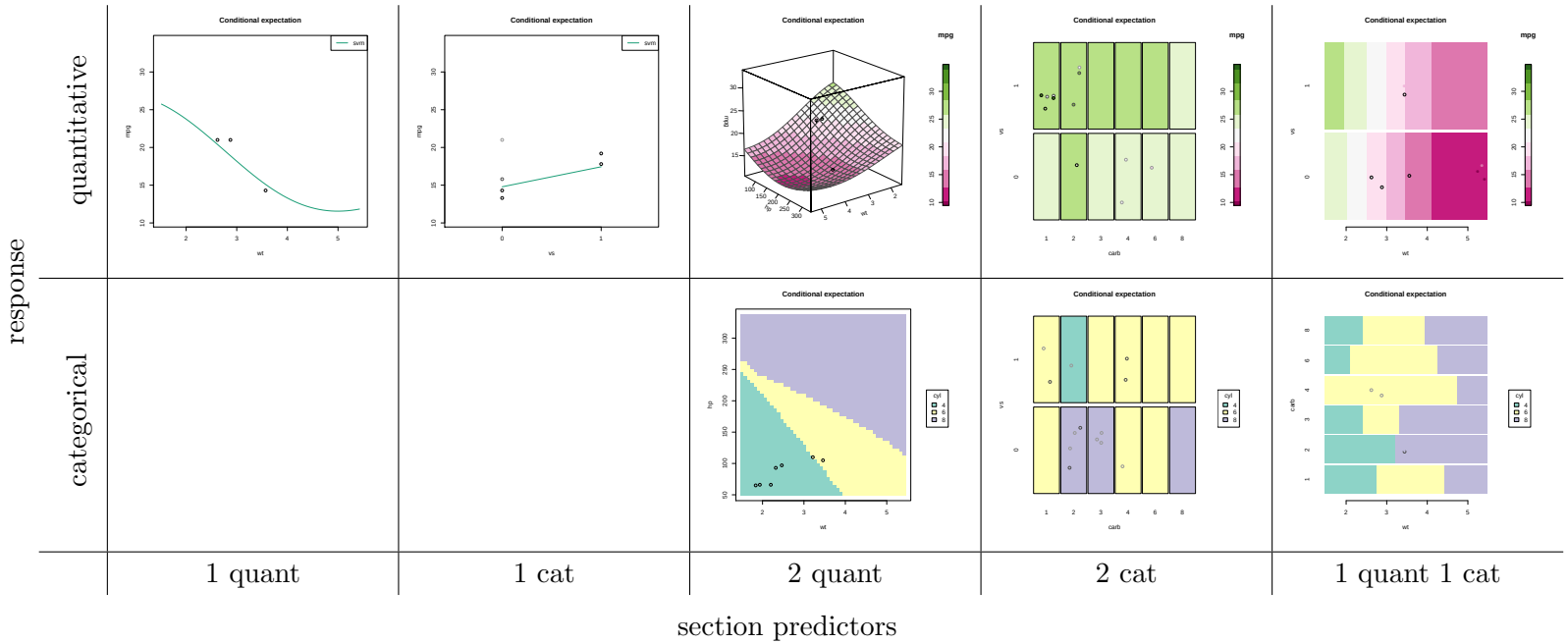


Figure 3.3: Displaying a fitted model on a section (quant = quantitative, cat = categorical). For a categorical response, and a single section predictor, it is preferable to consider the class probability as a quantitative response, or add an extra section predictor.

3.3.2 Classical regression models

The $\hat{f}$ from classical regression models can be interpreted as a surface in data space, and so we can easily show its intersection with the section we have chosen. We then simply have the task of visualising the function $\hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_C)$ where $\boldsymbol{x}_S$ are seen as free parameters which can vary across their range and $\boldsymbol{x}_C$ are fixed parameters defining the section. This is the same approach to visualising sections through fitted models as for ICE plots (see Section 2.5).

3.3.3 Bayesian regression models

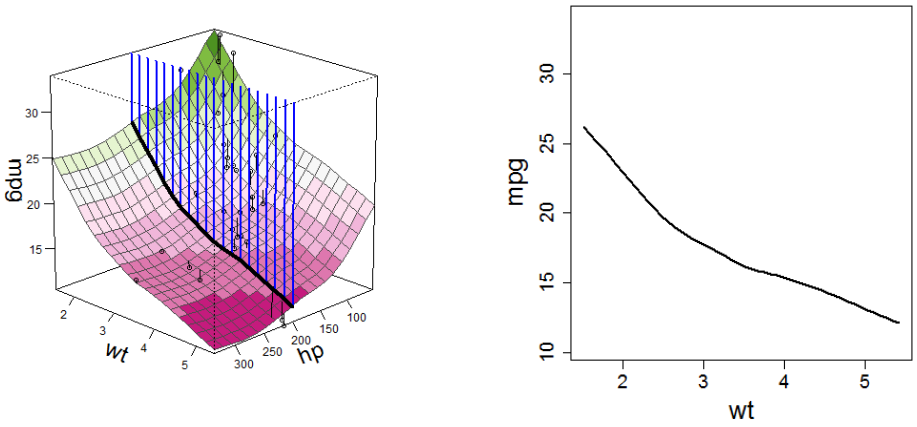
Bayesian models, on the other hand, provide a probability distribution for the expected response via a probability distribution on the parameters of the model. To visualise such models, a simple approach is to evaluate the posterior distribution on a grid, calculating the median (or some other measure of central tendency) each time, and finally visualise a surface joining these medians together.

If Markov Chain Monte Carlo (MCMC) is used when the posterior is difficult to calculate, we have a sample of predictive functions, and can produce a sample from the posterior of $\mathbb{E}[y \mid \boldsymbol{x}]$ on the section. One option here is to show all of the sampled predictive functions on the section and use transparency to give an impression of the probability distribution of $\mathbb{E}[y \mid \boldsymbol{x}]$ on the section. For implementation reasons, this option has not been explored in this work as transparency is not possible on some of the graphics devices needed for interactivity in R. Instead, for each of our points on the grid of $\boldsymbol{x}_S$ values, we have a sample of $\hat{f}$. We calculate quantiles of each of these samples and then show the median values, as well as percentiles for 2.5% and 97.5%. In this way, we can visualise a Bayesian model as a surface in data space, and therefore in the same manner as in Section 3.3.2. See Section 6.6 for an example of visualising sections through a Bayesian model.

3.3.4 Implementation: `predict`

In R, predictions for a fitted model are often calculated using a method for the generic `predict` function. This can be a function which accepts as arguments the fitted model object and a new data frame of observations, and returns the predictions of the model. See, for example, the documentation for `?predict.lm` in R.

In condvis, fitted models are evaluated on the section using the generic `predict` method. The `predict` methods are not implemented in a consistent manner across all model objects in R. Sometimes, there is simply no predict method, for example, when the parameters of a Bayesian model are sampled using external software. Model objects of class S4 can be very specific about their inputs, and fail with an error when unnecessary arguments are supplied, instead of ignoring them as in the less strict S3 paradigm. The best way to deal with these situations is to make an S3 object which acts as a wrapper to the fitted model object, and behaves like an



(a) (b)

Figure 3.4: Visualising a section. (a) Visualising a 2 predictor model, and taking a section at `hp` = 200. (b) Visualising the section through model at `hp` = 200. There are no observed data points on this section.

object of class `lm` in terms of input and output of its `predict` method. Examples of this kind of wrapper object can be seen in the examples in Chapter 6.

3.4 Displaying observed data

Whenever we visualise a fitted model, it is desirable to enhance the visualisation with the observed data, to see if the data support the fitted model or if model assumptions are violated. However, it is not useful to plot all observed data points on a section visualisation. We want to show only observed data which are relevant to the section being visualised. We achieve this by showing observed data which are *near* the section (recall the discussion in Section 1.1 of Figure 1.2, shown again in Figure 3.4), and so we must formalise what we mean by near. We do so using distance or dissimilarity measures, displaying only data within a certain distance of the section. This gives rise to a fixed radius near neighbour search.

The main tasks when displaying observed data points on a section can be summarised as follows:

1. Given a section and an observed data point, calculate the distance between the section and the observed point.

2. Convert the distance between the section and observed point to a *similarity weight*, which is a similarity measure scaled to be between 0 and 1 – observations on the section are assigned similarity weight 1, while observations with increasing distance from the section are assigned lower values.

3. Plot the observation with a colour intensity based on the similarity weight – observations with similarity weight 1 are plotted normally using a given

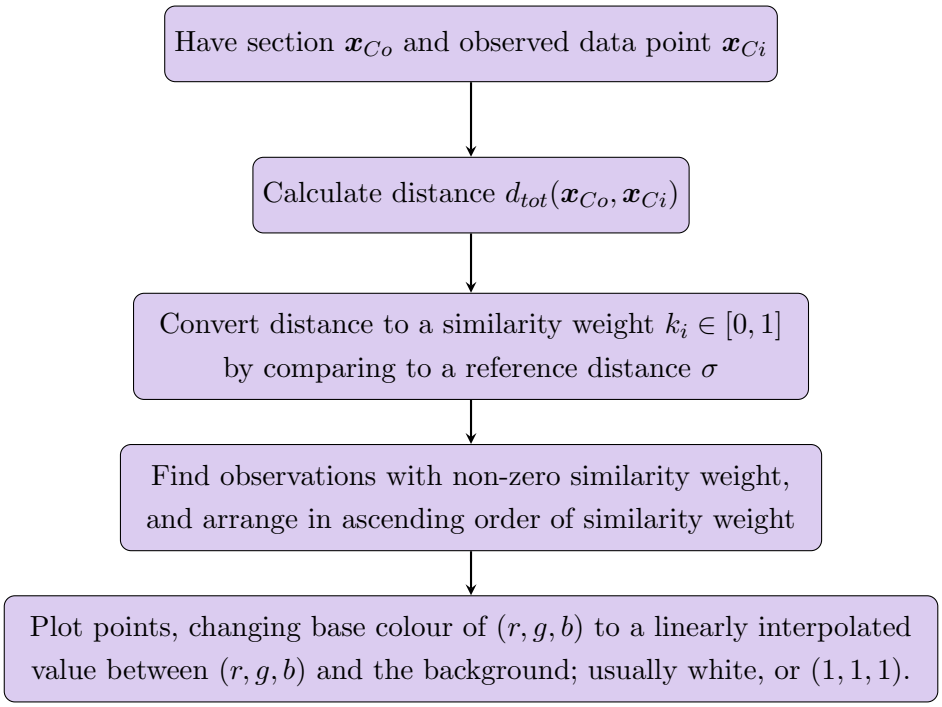


Figure 3.5: Process for displaying observed data on a section visualisation.

plotting colour, while observations with similarity weight 0 are plotted as the background colour (or not plotted at all, in practice). Colours for observations with intermediate similarity weights are chosen on a linear scale from the plotting colour to the background colour in Red, Green, Blue (RGB) space.

An overview of the process of displaying observed data on a section is shown in a flowchart in Figure 3.5.

3.4.1 Distance function

We consider both the section and observed data as points in the space of the conditioning predictors. We use a dissimilarity function of Minkowski distance on the quantitative conditioning predictors and a simple mismatch count on the categorical conditioning predictors.

We define d_{cont} as the Minkowski distance between the quantitative coordinates of two points in data space, $\boldsymbol{u}$ and $\boldsymbol{v}$, written as

$$d_{cont}(\boldsymbol{u}, \boldsymbol{v}) = \left(\sum_{j \in \mathcal{J}} |\boldsymbol{u}_j - \boldsymbol{v}_j|^q \right)^{1/q} : \mathcal{J} \text{ indexes the numeric elements of } \boldsymbol{u} \text{ and } \boldsymbol{v}. \tag{3.1}$$

where $q \geq 1$ is a real number. We define the dissimilarity between the categorical coordinates of two observations as the number of mismatches between those categorical coordinates, written as

$$M(\boldsymbol{u}, \boldsymbol{v}) \tag{3.2}$$

This is equivalent to the Hamming (1950) distance between the categorical coordinates, if categorical variables are recoded as single symbols (for example, integers).

We then define the total dissimilarity between two observations as

$$d_{tot}(\boldsymbol{u}, \boldsymbol{v}) = d_{cont}(\boldsymbol{u}, \boldsymbol{v}) + \lambda M(\boldsymbol{u}, \boldsymbol{v}) \tag{3.3}$$

where $\lambda \in \mathbb{R}^+$ is an arbitrary scaling constant. This distance is calculated after standardising the quantitative conditioning predictors to have zero mean and unit variance. Of course, other more robust methods of scaling the quantitative predictors could also be used. We do not apply any standardising to the categorical predictors, aside from the single constant λ. This implies a certain disregard for the relative importance of a level mismatch on one categorical predictor as compared to a level mismatch on another categorical predictor. For example, we would expect more level mismatches on a categorical predictor with many levels than on a balanced binary categorical predictor. There are alternatives to the mismatch count dissimilarity measure. One such example is Gower's (1971) general coefficient of dissimilarity, which allows special consideration for unbalanced class distributions, ordinal categorical variables, and other cases. For the purposes of this work, we find the mismatch count to be an adequate dissimilarity measure between categorical predictors. Changing the distance measure to Gower's (or any other distance measure) does not materially affect any of the steps in the methodology of this thesis, and so it has not been given any detailed discussion. In particular, none of the data sets discussed have enough complicated categorical variables to merit the use of Gower's distance/similarity.

3.4.2 Conversion from distance to similarity weight

We use the distance measure to obtain a similarity weight value as follows. The distance between the section and observation i is calculated as

$$d_i = d_{tot}(\boldsymbol{x}_{Co}, \boldsymbol{x}_{Ci}) \tag{3.4}$$

where $\boldsymbol{x}_{Co}$ describes the section in the coordinates of the conditioning predictors, and $\boldsymbol{x}_{Ci}$ describes an observation in the coordinates of the conditioning predictors. We then assign a weight k_i to an observed data point $\boldsymbol{x}_i$ based on its distance d_i from the section:

$$k_i = \max\left(0, 1 - \frac{d_i}{\sigma}\right) \tag{3.5}$$

where $\sigma > 0$ is a threshold parameter set by the user. Letting σ equal 0 gives exact conditioning; that is, we only plot points which lie exactly on the section with $d_i = 0$. Increasing σ gives more approximate conditioning. The choice of σ is rather arbitrary. An optimal choice of σ might be one that is as small as possible while maintaining enough data to visualise patterns in a plot. Setting $\lambda > \sigma$ means that only observations which match exactly the categorical levels of the section

can be assigned non-zero weight. Setting $\lambda = 0$ means that categorical predictors are ignored by the calculations for distance and similarity weight. Setting λ to a value in the interval $(0, \sigma]$ means that observations with some but not all categorical predictor levels matching may be considered for visualisation, and observations with less mismatches are given more similarity weight than those with more mismatches.

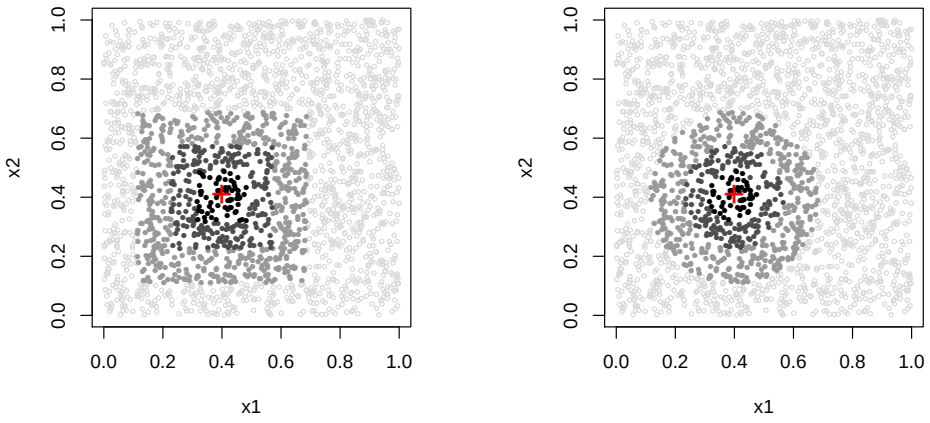
The plotting colour may be considered a point in three-dimensional RGB (Red, Green, Blue) space, (r, g, b). We then fade this colour in steps to the background colour of the plotting device with increasing distance from the section. A typical background colour is white, or $(1, 1, 1)$ in RGB. For a similarity weight of 1, we plot the observation with its full colour. For a similarity weight of 0, we give the observation the background colour, or in practice, do not plot it at all. For example, with a black plotting colour and a white background, points will be plotted as black, dark grey, light grey and white with decreasing similarity weight from 1 to 0.

The condvis package currently implements two special cases of the Minkowski distance (Equation 3.1), namely maximum norm and Euclidean distance. Euclidean distance is the Minkowski distance with q equal to 2. The maximum norm is the limit of the Minkowski distance as q tends to infinity. See Figure 3.6 for an example of both distance measures with two conditioning predictors. Using the maximum norm results in conditioning which is essentially the same as that of trellis graphics (see Section 2.2), where intervals are used to define conditioning. For example, consider the conditioning shown in Figure 3.6a by choosing points within a certain maximum norm distance of the section in the space of the conditioning predictors. If we ignore the colour intensities, we can obtain the same approximate conditioning using the shingles method from trellis to choose a subset of the data, that is, using the intervals $0.4 - \text{sd}(x_1) < x_1 < 0.4 + \text{sd}(x_1)$ and $0.4 - \text{sd}(x_2) < x_2 < 0.4 + \text{sd}(x_2)$, shown in Figure 3.7. The $\text{sd}(x_1)$ and $\text{sd}(x_2)$ arise from the fact that the threshold σ is set to 1, so that points within one standard deviation of the section are given similarity weight greater than zero.

3.4.3 Plotting observed data points

Observed data points with a similarity weight below a certain value are discarded and not plotted. The purpose of this is to avoid plotting observations which would be visually indistinguishable from the background colour if plotted. This saves on computation time for the plotting function, and reduces the storage space taken by a PDF version of a static plot. The points with sufficient weight to be visible are plotted in increasing order of similarity weight, so that the points nearest the section are plotted last, giving them more visual prominence.

When plotting observed data points on a colour map, the circles representing observations are coloured according to their similarity weight, as with the other section visualisations. These circles are then filled with the colour of their observed value, for both numeric and categorical responses. See the example in Section 6.3



(a) (b)

Figure 3.6: Choosing data to display using distance between section and observations. Condition on x1 and x2 by taking a section defined by x1 = x2 = 0.4. There are no points exactly at these coordinates, so we take points nearby ($\sigma = 1$). (a) Maximum norm distance. (b) Euclidean distance.

for an example of this for a categorical response.

When plotting observed data points on a perspective mesh plot, line segments representing the residual are also plotted to give extra context to the observations. If the observations are far from the section, the residuals may not touch the fitted model on the section.

3.4.4 Implementation: `similarityweight`

The `similarityweight` function calculates the similarity weight for observations given a section in data space. It is a wrapper to the internal `.similarityweight` function which first standardises the data, and returns another function which calculates the similarity weights for the observations given a single section. The wrapper function is vectorised to accept a data frame describing multiple sections in data space. The inputs to `similarityweight` are:

- `x` A dataframe describing sections in the coordinates of the conditioning predictors.
- `data` Observed data, a dataframe with same names as `x`.
- `threshold` Threshold distance outside which observations will be assigned similarity weight zero. This is numeric and should be > 0. Defaults to 1. Called σ in Section 3.4.2.
- `distance` Either `"euclidean"` (default) or `"maxnorm"`.

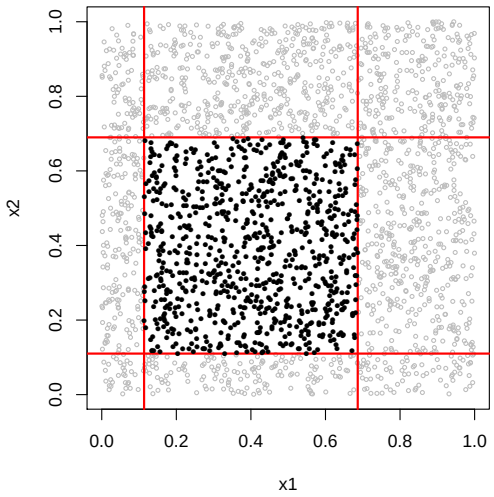


Figure 3.7: Obtaining a similar conditioning to Figure 3.6a using the intervals method from trellis.

- `lambda` A constant to multiply by the number of categorical mismatches, before adding to the Minkowski distance, to give a general dissimilarity measure. If left `NULL` (the default), behaves as though `lambda` is set larger than `threshold`, meaning that one factor mismatch guarantees zero weight. Called λ in Section 3.4.1.

Other people have found practical uses for `similarityweight`, for example, Grolemund and Wickham (2016) use it to evaluate the reliability of predictions on grids in data space.

3.5 Example of visualising a section and nearby data

This section presents a brief example with accompanying R code to visualise a section through a fitted model, and display nearby observed data points. This example serves to give a simple view of how a plot may be made conditional using the similarity weight concept, and demonstrates the central task of the condvis package without the need for trawling through source code. Furthermore, this example shows that the idea of making a plot conditional on a region of data space is not specific to visualising a section in data space, but rather the conditioning is achieved via the colouring of individual observations, and so any plot of individual observations can be made conditional. This is not evident in the condvis package, due to its organic development as a toolset for visualising models in data space, but we give an example of this broader view of conditional plots in Section 7.

The code for this example is in Code Snippet 3.1. We first generate some data from the process

$$y = x_1 + x_2 + \varepsilon$$

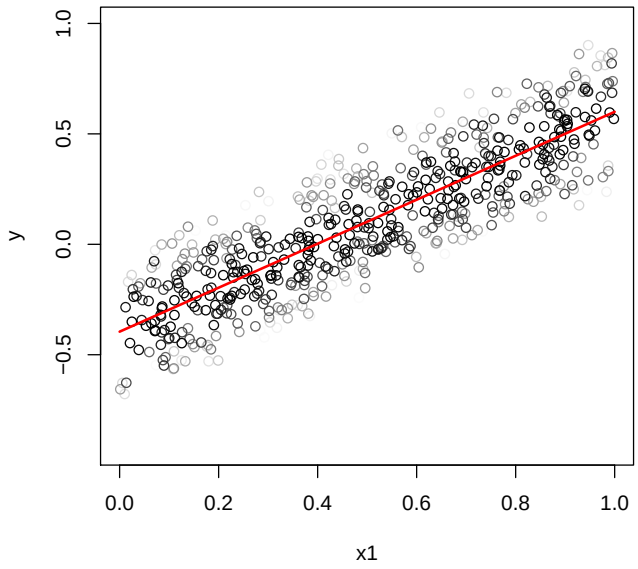


Figure 3.8: Visualising a section, showing the fitted model and nearby observed data. From Code Snippet 3.1.

$$\varepsilon \overset{iid}{\sim} N(0, 0.05), \qquad x_1, x_2 \overset{iid}{\sim} U(0, 1)$$

and fit a linear model. We define a section at $x_2 = 0.4$, and calculate the similarity weight of observations relative to this section. We use this to choose the plotting colours for the observations, giving prominence to observations near the section. We then visualise the section, showing the fitted model by evaluating the fitted response on a grid of values on the section. We plot the observations first to avoid obscuring the fitted model. Figure 3.8 shows this visualisation, which has the appearance of a simple linear regression with well behaved residuals.

Next, we try a section nearer the edge of the data at $x_2 = 0.05$, shown in Figure 3.9. This time, the plot suggests a strong bias towards negative residuals. This is not a problem with the model, as we know it to be well specified in this case. Rather, it is a symptom of approximate conditioning —showing nearby data. The observations near this section are subject to variability due to their x_2 coordinates, which is biased because we are at the edge of the data; we reduce the variability by visualising only nearby points, but cannot remove it completely for continuous conditioning predictors. This is something we must always be aware of when attempting conditional visualisation. Particularly, in the conditional visualisation in this thesis, the conditioning predictors are dealt with using the one-dimensional similarity weight. This means that we cannot discern information such as "which side of the section is this observation on?"

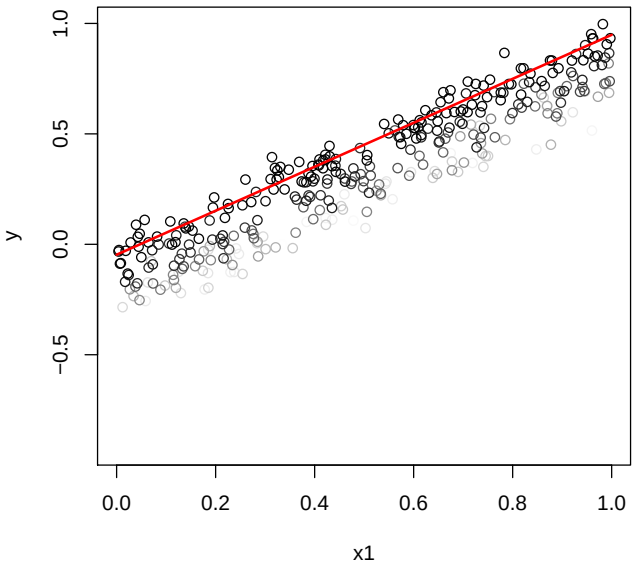


Figure 3.9: Visualising a section, showing the fitted model and nearby observed data. The nearby observed data seem to show a bias in this instance. From Code Snippet 3.1.

3.6 Practical applications of new method

3.6.1 Interpreting fitted models

Additive models have a comparatively straightforward interpretation by way of parameter estimates in the case of linear regression, or component function shapes in the case of additive spline models.

While such interpretable models are still popular, much of the focus in modern applied statistics lies in kernel methods, neural networks, tree methods, and ensembles of models. These methods allow for the fitting of large and complex models, typically trading interpretability for predictive performance. It would seem a shame to develop a successful predictive model, and make no effort to understand it on the basic level of predictor effects. Moreover, statistical analyses are arguably less valuable if their core messages cannot be communicated with ease to non-experts.

By visualising sections, we show fitted models as curves, surfaces and colour maps while taking advantage of the *caeteris paribus* ("all else equal") concept, which is reinforced by the fact that the user is visualising one section at a time. We can say to ourselves, and non-expert colleagues alike, that this graphic shows the modelled effect of one or two predictors, holding other predictors at fixed values.

3.6.2 Understanding how observed data support a fitted model

It is standard practice when regressing on a single predictor to plot the data, with the fitted model superimposed on the plot. This allows the analyst to identify potential problems with relative ease; for example, the suitability of the functional form of the model, or assumptions on the error structure (though, this is perhaps better left to residual plots).

As more predictors are added to a model, this becomes less practical. It quickly becomes a problem of simultaneously visualising a function as well as a point cloud in p-dimensional space. As discussed in Section 1.1, this is easy with one predictor, achievable with two predictors, and rather difficult otherwise.

Conditional visualisation, as presented in this thesis, produces simple two-dimensional and three-dimensional graphics which are intended to be as easy to interpret as direct visualisations of fitted models with one or two predictors.

3.6.3 Comparing fitted models

When visualising fitted models on a section along one predictor, fitted models are represented by curves, and so multiple models can be compared quite easily. This is not so easy for two section predictors, and as such, it is not currently implemented in condvis. Consider the perspective mesh plot for two quantitative section predictors and a quantitative response in Figure 3.3. If we add another regression surface to this graphic, it will either be in front of, or behind, the model already visualised. See Chapter 7 for further discussion of visualising multiple models.

3.7 Implementation: `plotxs`

The visualisation of sections in data space is implemented in the `plotxs` function in condvis. The `plotxs` function produces a section in data space (of each type shown in Figure 3.3), showing fitted models where they intersect the section, and observed data according to their similarity weights. The fitted model is calculated via the `predict` method, the model objects being supplied in a list argument to `plotxs`. The similarity weights are calculated with the `similarityweight` function, and supplied as an argument to `plotxs`. The `plotxs` function returns an S3 object which contains the relevant information for updating the plot, as is needed when changing the current section being visualised by user interaction. The updating is accomplished by the `update.xsplot` S3 method, which is not exported.

3.8 Chapter summary

This chapter introduced a new approach to conditional visualisation. The approach visualises two-dimensional and three-dimensional sections in data space, showing fitted models where they intersect the section and data which is near the section.

The concept of proximity in data space is formalised with a dissimilarity measure composed of Minkowski distance on numeric predictors and mismatch counts on categorical predictors. The sections can be visualised in R with the `plotxs` function in condvis.

```
library("condvis")

## Generate some data.
set.seed(746182481)
x1 <- runif(1000); x2 <- runif(1000)
y <- x1 - x2 + rnorm(sd = 0.05, n = 1000)
d <- data.frame(y, x1, x2)

## Fit a model
m <- lm(y ~ ., data = d)

## Define a section.
section <- data.frame(x2 = 0.4)

## Calculate similarity weight of observations, given the section.
sw <- similarityweight(x = section, data = d[, "x2", drop = FALSE],
  threshold = 1)
colours <- condvis:::weightcolor(col = "black", weights = sw)

## Initialise a plot with reasonable ranges.
plot(x1, y, col = NULL)

## Display nearby observed data.
plotorder <- attr(colours, "order")
points(x1[plotorder], y[plotorder], col = colours[plotorder])

## Display the fitted model on the section.
predgrid <- expand.grid(x1 = seq(0,1, length.out = 10), x2 = 0.4)
points(predgrid$x1, predict(m, predgrid), type = "l", col = "red", lwd = 2)

## Repeat the process for a section nearer the edge of the data.
section2 <- data.frame(x2 = 0.05)
sw2 <- similarityweight(x = section2, data = d[, "x2", drop = FALSE],
  threshold = 1)
colours2 <- condvis:::weightcolor(col = "black", weights = sw2)

## Initialise plot and display nearby observed data.
plot(x1, y, col = NULL)
plotorder2 <- attr(colours2, "order")
points(x1[plotorder2], y[plotorder2], col = colours2[plotorder2])

## Display the fitted model on the section.
predgrid2 <- expand.grid(x1 = seq(0,1, length.out = 10), x2 = 0.05)
points(predgrid2$x1, predict(m, predgrid2), type = "l", col = "red", lwd = 2)
```

Code Snippet 3.1: Visualise a section through a fitted model, and display nearby observed data. Related visualisations in Figures 3.8 and 3.9.

Chapter 4

Choosing sections interactively

4.1 Introduction

This chapter presents an interactive approach to implementing the conditional visualisation described in Chapter 3. The resulting graphics are called *interactive conditional expectation plots*, and are implemented by the `ceplot` function in my R package condvis.

An interactive conditional expectation plot consists of two main parts. The first part is a section through the fitted model, showing values of $\hat{f}$ evaluated on a grid of values of $\boldsymbol{x}_S$, with $\boldsymbol{x}_C$ held constant. We call this a *conditional expectation plot*. The second part is a collection of data visualisations, which show both the distribution of the predictors and the current condition (i.e., the values to which $\boldsymbol{x}_C$ are set). We call these *condition selector plots*. The user interacts with the condition selector plots in order to choose the section to be visualised.

This chapter shows one example of an interactive conditional expectation plot with some simulated data, deferring the real data applications to Chapter 6.

Chapter goal

The goal of this chapter is to present a way to implement the conditional visualisation techniques from Chapter 3 using interactive graphics to choose the section. The focus will be on ways to choose the section, and how to encourage the choice of *good* sections, that is, sections of interest to the analyst, or near observed data.

Chapter outline

Section 4.2 presents condition selector plots, the various graphics for interactively choosing a section to visualise. Section 4.3 discusses how condition selector plots can be arranged to provide some protection against unwitting extrapolations in the data space. Section 4.4 briefly discusses some of the difficulties encountered with condition selector plots. Section 4.5 gives an overview of the software implementation of interactive conditional expectation plots. Section 4.6 shows the use of interactive

conditional expectation plots on some simulated data. Section 4.7 concludes the chapter with a summary.

4.2 Condition selector plots

Condition selector plots depict the data in $\boldsymbol{x}_C$ and allow a section to be selected interactively.

4.2.1 Univariate and bivariate condition selectors

These plots depict the data in $\boldsymbol{x}_C$ by way of one-dimensional or two-dimensional marginal views. Examples of each are given in Figure 4.1.

- Histogram for one quantitative predictor in $\boldsymbol{x}_C$.
  The histogram gives a univariate view of the marginal distribution of a quantitative predictor. The selected predictor value is shown as a vertical line.

- Barplot for one categorical predictor.
  The barplot gives a univariate view of the marginal distribution of a categorical predictor. The selected predictor value is shown by highlighting one of the bars with colour.

- Scatterplot for two quantitative predictors.
  The scatterplot gives a bivariate view of the joint distribution of two quantitative predictors. The selected pair of predictor values is shown by a cross hairs of a vertical and a horizontal line. If there are a large number of observations, it is prudent to use a two-dimensional histogram or heatmap of data density instead of a scatterplot of individual observations.

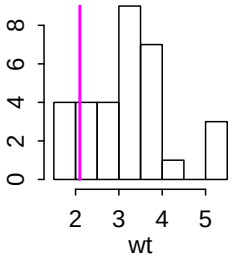
- Boxplot for one quantitative predictor and one categorical predictor.
  The boxplot gives a bivariate view of the joint distribution of a single quantitative predictor and a single categorical predictor. The selected pair of predictor values is shown by a cross hairs of a vertical and a horizontal line.

- Spineplot for two categorical predictors.
  The spineplot gives a bivariate view of the joint distribution of two categorical predictors. The selected pair of predictor values is shown by highlighting one of the rectangles with colour. A spineplot is also known as a segmented barchart.

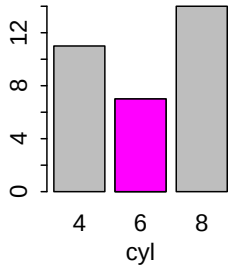
4.2.2 Full scatterplot matrix condition selector

A scatterplot matrix is a grid of bivariate data visualisations. This provides all possible bivariate views of the conditioning predictors. It provides the most views in which to detect extrapolation, but can be very cumbersome for selecting sections, and requires a lot of updating as an interactive graphic. It is difficult to fit any more

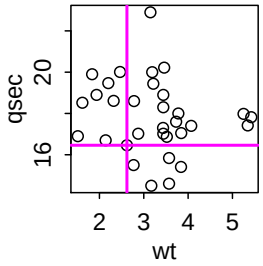
Histogram for one quantitative predictor in $\boldsymbol{x}_C$.



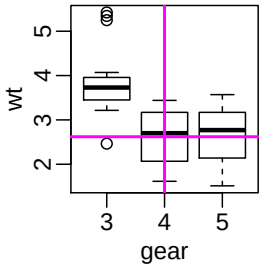
Barplot for one categorical predictor.



Scatterplot for two quantitative predictors.



Boxplot for one quantitative predictor and one categorical predictor.



Spineplot for two categorical predictors.

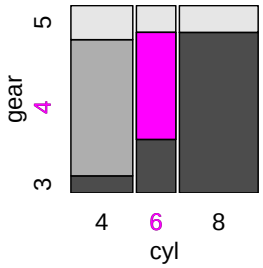


Figure 4.1: Five different types of condition selector plots.

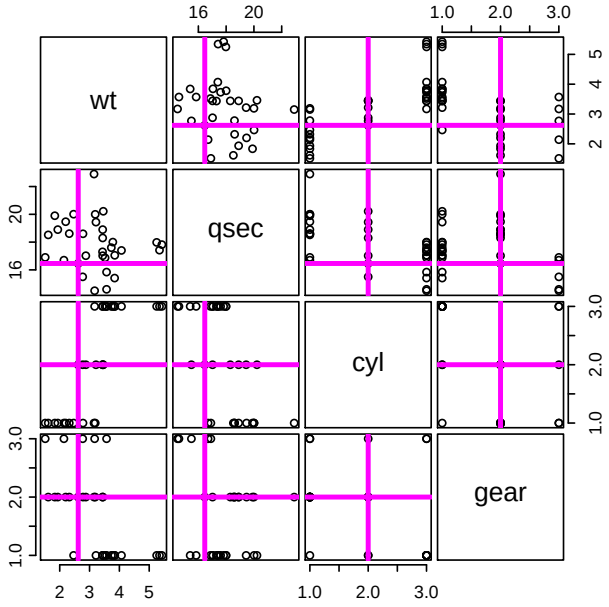


Figure 4.2: Full scatterplot matrix condition selector, showing the same condition as Figure 4.3.

than 15 predictors in a scatterplot matrix condition selector on regular computer displays. We coerce categorical predictors to integers when producing scatterplot matrices. It would be possible to use a generalised scatterplot matrix (Emerson et al., 2013) to better represent mixed quantitative and categorical data, but we opt for the standard scatterplot matrix here in order to have the current section represented by straight lines on the scatterplot matrix. Figure 4.2 shows a scatterplot matrix condition selector.

4.2.3 Parallel coordinates condition selector

A parallel coordinates plot (Inselberg and Dimsdale, 1990) is a visualisation tool for multivariate data. Whereas scatterplots arrange axes perpendicular to each other, a parallel coordinates plot places the axes for each dimension parallel to each other, allowing the visualisation of high-dimensional data. As condition selector plots, parallel coordinates plots are relatively neat and easy to understand, but spotting extrapolations is certainly not easy. As with the scatterplot matrix, categorical predictors are coerced to integers when producing a parallel coordinates plot. Figure 4.3 shows a parallel coordinates condition selector plot.

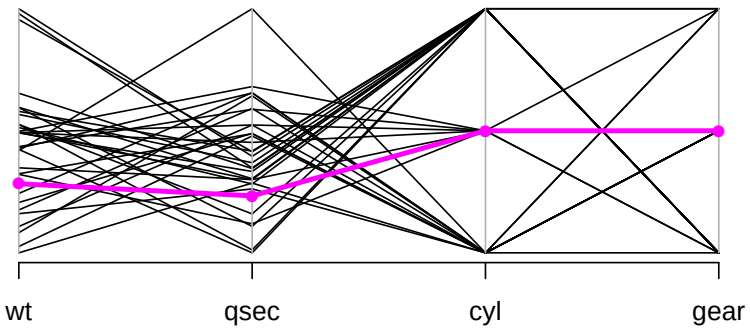


Figure 4.3: Parallel coordinates condition selector plot, showing the same condition as Figure 4.2.

4.2.4 Implementation: `plotxc`

The `plotxc` function in `condvis` produces the univariate and bivariate condition selector plots described in Section 4.2. The internal functions `plotxc.full` and `plotxc.pcp` produce a full scatterplot matrix condition selector plot and a parallel coordinates condition selector plot respectively. The `plotxc` functions all return an S3 object which contains the necessary information to update them, as is needed when reacting to user input. The updating is accomplished by the `update.xcplot` method, which is not exported.

4.3 Predictor pairings for condition selector plots

In producing an interactive conditional expectation plot, we are presented with a choice in our representation of $\boldsymbol{x}_C$. This ranges from $|C|$ univariate condition selector plots to around half as many plots when using bivariate condition selector plots. We have choices in what predictors we pair in bivariate plots, and the order in which we arrange the plots. In the absence of any more pressing criteria, we would like to choose bivariate pairings that minimise the chance of unwitting extrapolation in $\boldsymbol{x}_C$, and reduce the number of condition selector plots required.

Consider $\boldsymbol{x}_C$ consisting of two categorical predictors, `gear` and `cyl` from the mtcars data. To condition on $\boldsymbol{x}_C$, we set each of `gear` and `cyl` to a single value. If we consider each predictor in isolation when assigning a value, we run the risk of assigning a combination of values which has not been observed before, for example, setting `gear` to 4 and `cyl` to 8. This is what can happen when a univariate condition selector plot is used for each predictor – a barplot in this case. If we consider the two predictors together, we can limit ourselves to combinations of predictor values which have been observed, and eliminate extrapolation in $\boldsymbol{x}_C$. This can be achieved with a bivariate condition selector plot – a spineplot in this case (Figure 4.4). The

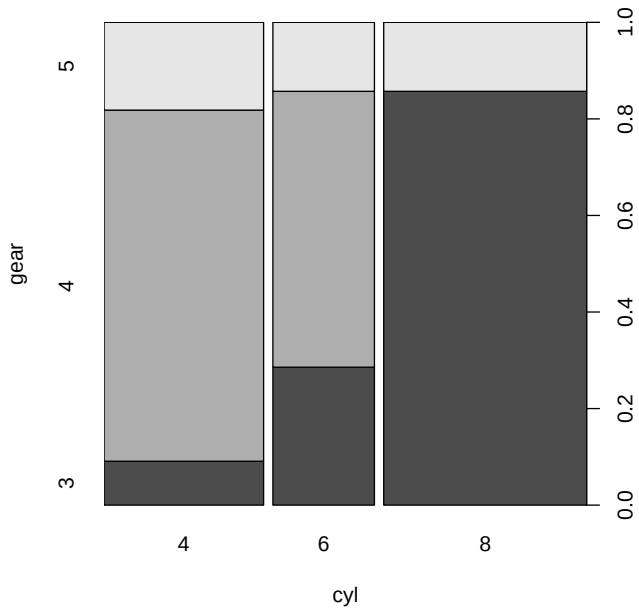


Figure 4.4: Two categorical predictors visualised in a spineplot. 8 predictor combinations are observed out of a total possible 9 combinations.

spineplot makes it plain to see that a case with `gear = 4` and `cyl = 8` has not been observed.

For a quantitative measure of the usefulness of a bivariate condition selector plot over two univariate condition selector plots, we calculate ratios that reflect the proportion of predictor value combinations which are shown to not be extrapolations by the bivariate view. The smaller this ratio, the more advantage is gained from using a bivariate condition selector plot over two univariate condition selector plots. A definition of extrapolation is required for this.

- For a single quantitative predictor, we consider any value outside the range of the observed data to be an extrapolation.
- For a pair of categorical predictors, we consider any combination of predictor levels that has not been observed before to be an extrapolation.
- For a pair of quantitative predictors, we may conservatively consider any combination of predictor values which fall outside the convex hull of the observed data to be an extrapolation. The convex hull is defined as the convex polygon of minimal area to contain all of the data. This is conservative because, when the data exhibit concave shapes, there may very well be regions inside the convex hull which would be considered to be extrapolations by most statisticians.
- For a categorical predictor paired with a quantitative predictor, we consider an extrapolation to be a combination of predictor values which lies outside the range of observed values of the quantitative predictor given the level of the selected categorical predictor.

4.3.1 A general comparison measure of bivariate and univariate distributions

Let $\mathbb{S}(x_j)$ be the support of a predictor x_j, and $\mathbb{S}(x_j, x_k)$ be the bivariate support of two predictors x_j and x_k. We then define a ratio

$$\frac{|\mathbb{S}(x_j, x_k)|}{|\mathbb{S}(x_j)||\mathbb{S}(x_k)|} \tag{4.1}$$

of which small values indicate an advantage to using a bivariate view of the predictors over two univariate views. Clearly, this ratio would be difficult to calculate directly (for example, we do not necessarily know the distribution of predictors, the support for a normally distributed predictor is $(-\infty, +\infty)$, and the cardinality of the support for any continuous predictor is infinity), so we use the following heuristics in calculating the ratio:

- For a quantitative predictor, we take $|\mathbb{S}(x_j)|$ to be the length of the range of observed values of x_j.
- For a categorical predictor, we take $|\mathbb{S}(x_j)|$ to be the number of levels for x_j.
- For two quantitative predictors x_j and x_k, we take $|\mathbb{S}(x_j, x_k)|$ to be the area of the convex hull of the observed data. We might alternatively use the area of a 'confidence region for future observations' (Geisser, 1993, Section 2.1).
- For two categorical predictors x_j and x_k, we take $|\mathbb{S}(x_j, x_k)|$ to be the total number of unique observed predictor level combinations.
- For one quantitative predictor x_j and one categorical predictor x_k, we take $|\mathbb{S}(x_j, x_k)|$ to be the sum of the ranges of x_j for each level of x_k, $\sum_c \mathbb{S}(x_j \mid x_k = c)$.

The following sections demonstrate the use of these heuristics with the `mtcars` data.

4.3.2 Two categorical predictors

To assess the advantage of a spineplot over two barplots, we first calculate the total number of possible predictor level combinations. We then calculate the number of these predictor level combinations we have observed. We calculate the ratio

$$\frac{\#\text{ possible combinations observed}}{\#\text{ possible combinations}} \tag{4.2}$$

of which small values indicate an advantage to using the bivariate plot. For example, consider the `gear` and `cyl` predictors from the mtcars data shown in a spineplot in Figure 4.4. Nine combinations of predictor values are possible with two predictors having three levels each. Eight of these combinations are observed in the data. Therefore, this bivariate pairing would not provide much improvement over two univariate plots in avoiding extrapolation on these predictors.

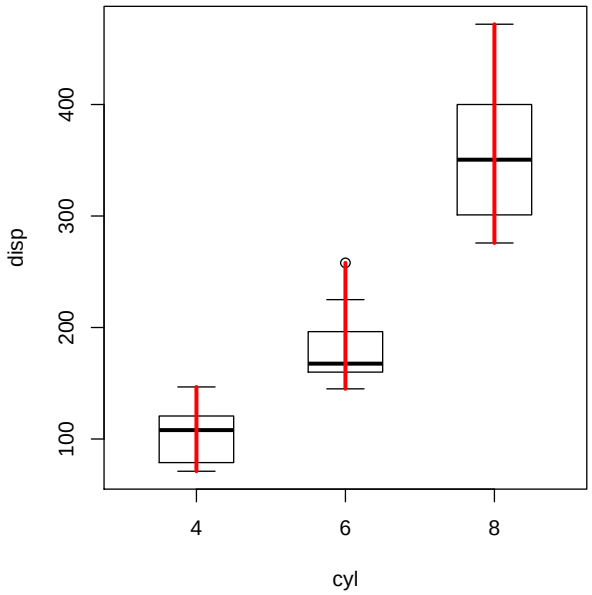


Figure 4.5: One categorical predictor and one quantitative predictor visualised with a boxplot. The average length of the conditional ranges of `disp` (in red) is considerably less than the overall range of `disp`.

4.3.3 One quantitative, one categorical predictor

To assess the advantage of a boxplot over a histogram and a barplot, we first calculate the average range of the quantitative predictor across the levels of the categorical predictor. We then divide this by the total range of the quantitative predictor to give the ratio

$$\frac{\sum \text{range(quantitative predictor) given categorical predictor level}}{(\#\text{ levels of categorical predictor})(\text{range(quantitative predictor)})} \tag{4.3}$$

of which small values indicate an advantage to using the bivariate plot. For example, consider the `cyl` and `disp` predictors from the `mtcars` data shown in a boxplot in Figure 4.5. The average length of the conditional ranges of `disp` (in red) is considerably less than the overall range of `disp`. So there is an advantage to using the bivariate pairing in this case, rather than two univariate plots.

4.3.4 Two quantitative predictors

There are a few possible approaches to assessing the advantage of a scatterplot over two histograms. The first option is the ratio of area of the convex hull to the bounding rectangle of data. The second option is the ratio of area of a confidence region from a kernel density estimate to the area of the bounding rectangle. The third option is a *scagnostics* (Wilkinson and Wills, 2008) measure, which is a number in $[0, 1]$ describing the shape of data in a bivariate view.

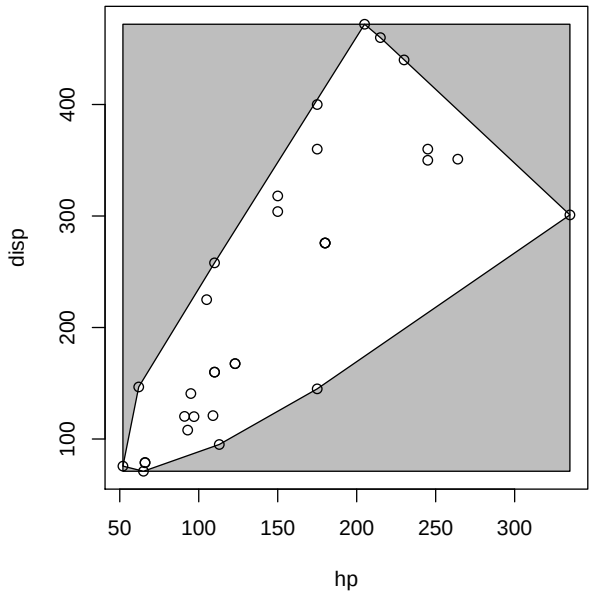


Figure 4.6: Two quantitative predictors visualised in a scatterplot, with the convex hull of the data shown. The grey region is where extrapolation might occur when using two univariate views.

Ratio of convex hull to bounding rectangle

We first standardise both predictors to have mean zero and unit variance. We calculate the area of the bounding rectangle as the product of the ranges of the two predictors. We calculate the area of the convex hull of the observed points in the two-dimensional projection. We then calculate the ratio

$$\frac{\text{area of convex hull}}{\text{area of bounding rectangle}} \tag{4.4}$$

of which small values indicate an advantage to using the bivariate plot. For example, consider the `hp` and `disp` variables from the mtcars data shown in a scatterplot in Figure 4.6. The grey area represents the region where extrapolation might occur accidentally when using two histograms to choose values for `hp` and `disp`.

It is possible that alpha-hulls (Edelsbrunner et al., 1983) would be more suited to the task of comparing a joint distribution of data to two marginal distributions, but this has not been explored in this work. Alpha-hulls are arbitrarily defined by the choice of parameter, while the convex hull is well defined for a set of points, making the convex hull a more firm concept to work with. An alpha-hull (or alpha-shape) may be thought of as a straight-line graph which captures the 'shape' of point sets.

Ratio of confidence region to bounding rectangle

Perhaps the most direct way to assess the bivariate distribution of quantitative data is to produce a kernel density estimate. We can extract from this density estimate

what Geisser (1993) refers to as a 'confidence region for future observations' for the data in the two-dimensional projection and substitute this for the convex hull in the ratio described previously, giving the ratio

$$\frac{\text{area of confidence region}}{\text{area of bounding rectangle}} \tag{4.5}$$

of which small values indicate an advantage to using the bivariate plot. It is possible to produce two-dimensional confidence regions in R using the hdrcde package (Hyndman et al., 2013).

Scagnostics

Scagnostics (Tukey and Tukey, 1985; Wilkinson et al., 2005; Wilkinson and Wills, 2008), or scatterplot diagnostics, are numeric values describing specific attributes of a scatterplot. While scagnostics do not fit neatly into the ratios described before, they certainly have potential for assessing bivariate relationships for the same purpose of avoiding extrapolations. They have descriptive names like *Stringy*, *Clumpy* and *Convex*. These measures are already scaled to the unit interval by their construction, and can be useful in assessing the advantage of a scatterplot over two histograms in avoiding extrapolations. Scatterplots which score highly on *Skinny*, *Stringy*, *Monotonic* or *Clumpy* would be good candidates for predictor pairings in condition selector plots, because these are the kinds of bivariate patterns which might be obscured in univariate visualisations. Scagnostics are implemented in the scagnostics package (Wilkinson and Anand, 2012) in R.

4.3.5 Implementation: `arrangeC`

The `arrangeC` function in condvis implements a default arrangement of conditioning predictors as discussed in Section 4.3. It relies on the `savingby2d` function which implements the bivariate relationship summary values described in the preceding sections. The predictors are paired in a greedy fashion until less than two predictors remain. If there are a large number of observations (say more than $20,000$ observations on 10 predictors), `arrangeC` performs its calculations on a sample of the data to keep computation time on the order of seconds on typical processors. There are two arguments to `arrangeC`:

- `data` A data frame containing observations on the conditioning predictors.
- `method` Character string describing the pairing criterion for quantitative conditioning predictors as discussed in the preceding sections. `"default"` uses the convex hull, `"DECR"` uses the confidence region, or it can be the name of a scagnostic measure.

It is difficult to suggest any single best way to organise condition selectors for all problems, so it is anticipated that users would supply their own ordering or

pairings, reflecting domain knowledge or variable importance. For example, with a motor insurance pricing model, a salesperson may wish to have quick access to the effect of voluntary excess or engine size (inputs which may be changed) in negotiating a policy, but may not have any need to understand the effect of age or claims history (inputs which cannot be changed).

4.4 Difficulties with condition selector plots

4.4.1 Large number of factor levels

Using barplots and boxplots to represent categorical predictors (as in Section 4.2.1) with a large number of levels is rather difficult. Fifteen or even ten levels would really start to crowd most instances of these data visualisations, and so make it impractical to choose sections with these visualisations. As it stands, there is no obvious solution to this problem, but it would certainly be worth considering collapsing some levels into each other to reduce the overall number of levels.

4.4.2 Large number of conditioning predictors

Clearly, there are limits to the number of conditioning predictors which can be used to choose sections interactively. For univariate and bivariate condition selectors, the limit arises from the number of plots we can fit on the screen. The scatterplot matrix is further restricted, owing to its rigid square structure. The parallel coordinates condition selector arguably allows the largest possible number of conditioning predictors, but it would be very difficult to use with any more than a few dozen predictors. Whichever condition selectors we use, we can certainly condition on more predictors than is possible with trellis graphics (Section 2.2).

When the number of conditioning predictors is too large, the only choice left is to set all elements of $\boldsymbol{x}_C$ to reasonable values (mean, mode, median, etc.) and then proceed to interactively change only the most important predictors, where importance is very much subject to the opinions and goals of the analyst. This amounts to essentially taking some conditioning predictors, setting them to a fixed value, and absorbing them into the intercept of the model.

Another solution is to give up on trying to choose interesting sections interactively, and use the automated approach, detailed in Chapter 5.

4.4.3 Skewed conditioning predictors

Skew is a common problem in data analysis and visualisation. Skew makes it difficult to simultaneously capture the full range of observed data as well as a detailed view of the distribution of the data. Condition selector plots are no exception, and can be rendered quite useless by a few extreme outliers, causing the more interesting parts of a plot to be compressed into a few pixels. This section offers a few suggestions to

overcome the problem of skew, although the solution will often need to be tailored to the specific application.

Logarithm transformations are often used to deal with skewed data as part of the modeling process, and so this can also solve the problem for condition selector plots. In tree models, the results are invariant under monotonic transformations of the predictors, and so logarithm transformations may be used to aid the construction of good condition selector plots without affecting the fitted model.

The main purpose of this thesis is to create an easily interpreted interface to fitted models and data spaces. Unless it has specific meaning in the context of the data, the logarithm transform is not helpful in this endeavour. It may very well be better to limit the range of the predictors to certain quantiles, for example, the 2.5% and 97.5% percentiles. In this way, the predictor is displayed in its natural units, ignoring any long tails or extreme values. This is implemented in the `plotxc` function in `condvis`, controlled by the `trim` parameter. In some cases, it may only be necessary to remove extreme values from the condition selector plot, but this would be more difficult to implement in an automated fashion.

Another solution might be to convert the predictor values to ranks for the visualisation, allowing the axis labels to show the actual predictor values.

4.4.4 Interactive arrangement of condition selector plots

Given the discussion in this chapter, it is fair to say that any attempt to objectively specify a single, effective arrangement of the conditioning predictors for visualisation will be met by considerable difficulty in general. It seems ideal then to allow the user to visually/interactively arrange and rearrange the condition selector plots. Due to the restrictive nature of interactivity in R graphics, this option has not been explored in this work, but would certainly be worth considering in further work.

4.5 Implementation: `ceplot`

The `ceplot` function is one of the main exported functions in the `condvis` package. It produces an interactive conditional expectation plot. The user has the option of using an R graphics device, or a Shiny web application. Within the R graphics device option, the user can place all graphics on one device, or place the section on one device with the condition selectors on another device. These options are set with the `type` argument. When using the R graphics device, we can alter the `threshold` parameter with the ',' (to decrease) and '.' (to increase) keys. In the Shiny implementation, we can change the threshold parameter with an interactive slider.

The most important arguments to `ceplot` are

- `data` The data frame containing the observed data.
- `model` A list of model objects to be passed to `plotxs` (see Section 3.7).

- `response` The character name of the response in `data`.
- `sectionvars` The character name of the section predictor(s) in `data`.
- `conditionvars` The character name of the conditioning predictors in `data`. If `NULL`, an attempt will be made to extract these from a fitted model in `model`.
- `threshold` The value for σ as in Section 3.4.2. The default value is 1. Larger values make the conditioning more approximate, smaller values make the conditioning more strict. Larger values show more data, smaller values show less.

4.6 Simulated data example: Interaction between conditioning predictors and section predictors

This section shows the use of interactive conditional expectation plots on some simulated data. We do not explore the idea of arranging condition selector plots, only the advantage of interactive conditional visualisation.

We consider a modified version of the simulated data from Goldstein et al. (2015) which they use to demonstrate ICE plots (Section 2.5.1), involving a data generating function with slightly more complex interactions than the original example.

$$Y = \begin{cases} 0.5X_1 - 5X_2 + \varepsilon & \text{if } -1 \leq X_3 < -0.5 \\ 0.5X_1 + |5X_2| + \varepsilon & \text{if } -0.5 \leq X_3 < 0 \\ 0.5X_1 - |5X_2| + \varepsilon & \text{if } 0 \leq X_3 < 0.5 \\ 0.5X_1 + 5X_2 + \varepsilon & \text{if } 0.5 \leq X_3 \leq 1 \end{cases} \tag{4.6}$$

$$\varepsilon \overset{iid}{\sim} N(0,1), \qquad X_1, X_2, X_3 \overset{iid}{\sim} U(-1,1)$$

We generate a data set with $2,000$ simulated observations from this process. We fit the same kind of model to the data as Goldstein et al. (2015), a gradient boosted tree using the gbm package (Ridgeway, 2013), and produce ICE plots using the ICEbox package (Goldstein et al., 2016). The code for this example is given in Appendix B.1. Figure 4.7 shows the marginal view of Y versus X_2. The ICE plot in Figure 4.8a shows sections (thin grey lines) through the fitted model along the X_2 predictor at observed data points. The Partial Dependence Plot (PDP) is the average of these sections, shown as a thicker line. Goldstein et al. (2015) argue that the ICE plot demonstrates a shortcoming of the PDP by showing sections through the fitted model that deviate strongly from the PDP. From the ICE plot, the analyst might be tempted to conclude that the relationship of Y with X_2 (conditional on other variables) consists of two linear functions, one increasing, the other decreasing. The centered ICE plot (Figure 4.8b) shows that this is not the case, and demonstrates that the standard ICE plot may obscure some conditional relationships. A centred ICE plot is an ICE plot where each section is vertically translated so that the fitted

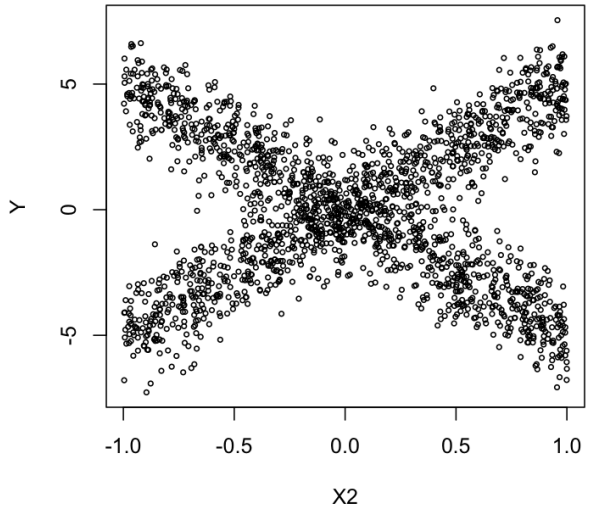


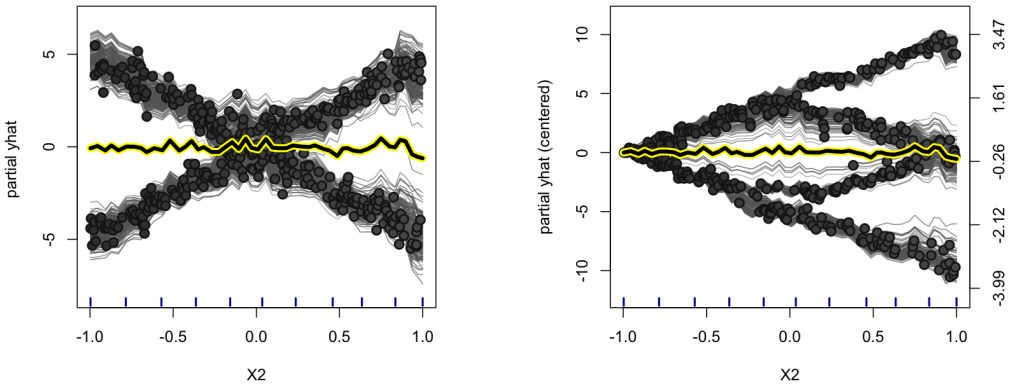
Figure 4.7: Scatterplot of Y versus X_2.

model equals zero for some specific value of the section predictor (for $X_2 = -1$ in this case). While the centered ICE plot may highlight a problem with the standard ICE plot, it does not clearly show the dependence of Y on X_2, conditional on X_1 and X_3. The most direct way to visualise the modeled relationship of Y and X_2 is to condition the ICE plot in Figure 4.8a on X_3 as with trellis graphics as in Figure 4.9. Now, four different effects of X_2 are plain to see. We can only produce this trellis graphic because we know the correct intervals of X_3 to condition on, and that we can safely ignore X_1 as having no effect.

We can arrive at a similar level of understanding of the fitted model by using an interactive conditional expectation plot, without any *a priori* knowledge of the data generating process. Figures 4.10 and 4.11 show snapshots from an interactive expectation plot, taking sections along X_2 and conditioning on X_1 and X_3, where the modeled effect of X_2 is shown clearly, with nearby observed data supporting the fitted model.

4.7 Chapter summary

This chapter presented an interactive approach to conditional visualisation as presented in Chapter 3. The approach is to interactively choose sections to visualise, with the aid of data summary graphics called condition selector plots. Condition selector plots can be univariate, bivariate, or more generally multivariate, and help the user to select sections to visualise which are relevant to the observed data.



(a) (b)

Figure 4.8: (a) ICE plot. (b) Centered ICE plot.

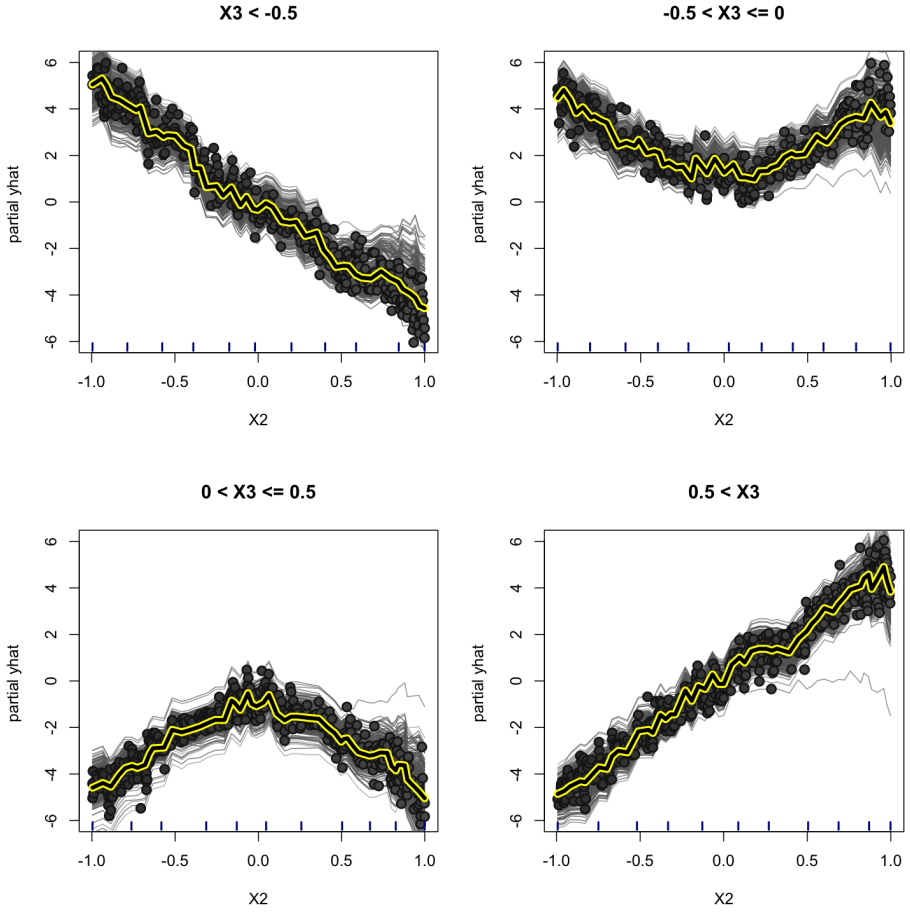


Figure 4.9: ICE plot from Figure 4.8a conditioned on X_3 in style of trellis.

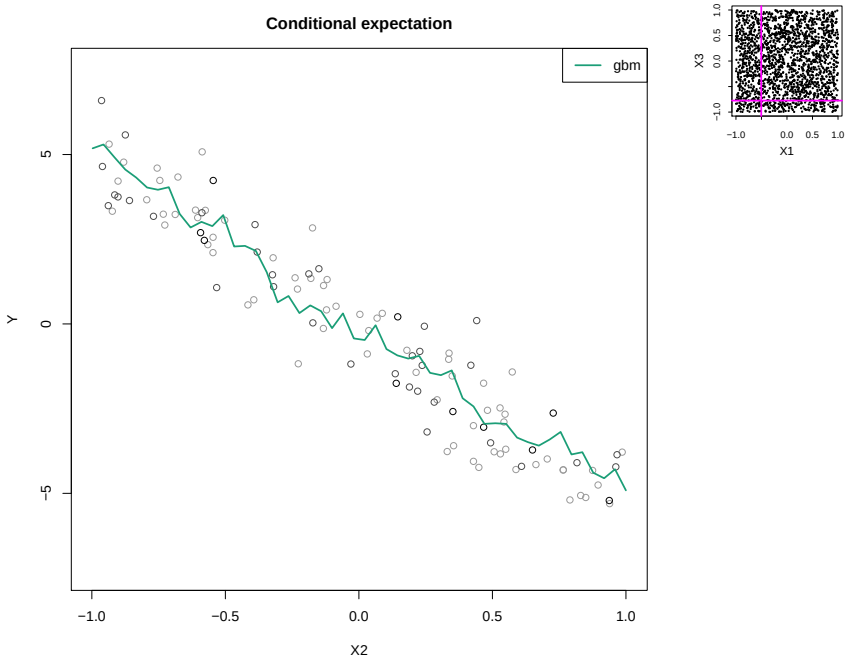


Figure 4.10: Conditional expectation plot. The left panel shows a section in data space. The upper-right panel shows the section in the space of conditioning predictors. From Code Snippet 4.1.

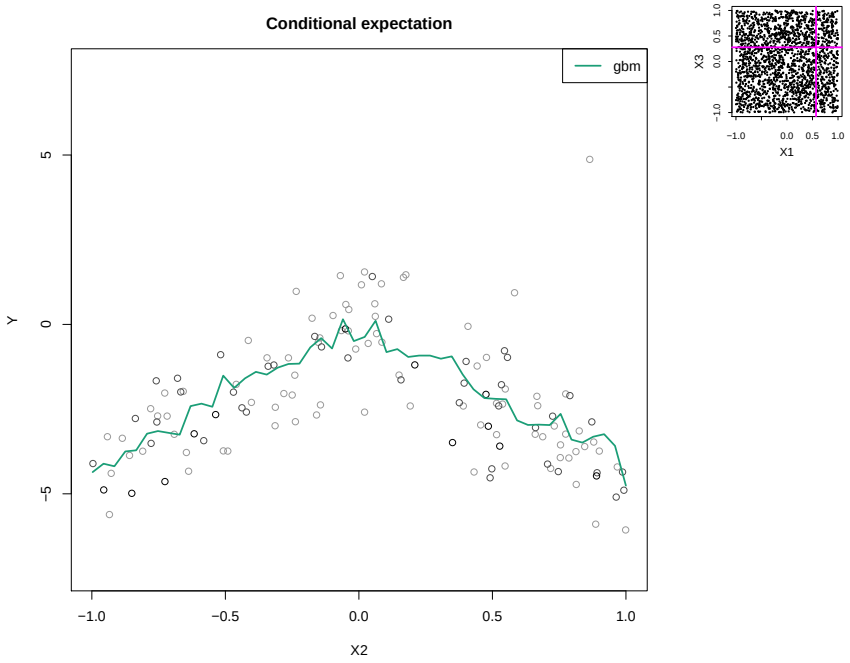


Figure 4.11: Conditional expectation plot. From Code Snippet 4.1.

```
library("condvis")
library("gbm")
load("interaction-workspace.rda") # from script in Appendix B.1

## Visualise sections along X2.
ceplot(data = dat, model = list(gbm = model), response = "Y", sectionvars =
  "X2", threshold = 0.5, xcplotpar = list(cex = 0.2))
```

Code Snippet 4.1: Code to produce interactive conditional expectation plot to explore gradient boosted model trained on simulated data. Related visualisations in Figures 4.10 and 4.11.

Chapter 5

Choosing sections in advance

5.1 Introduction

This chapter presents an automated approach to implementing the conditional visualisation described in Chapter 3, whereby all of the sections to visualise are chosen in advance, and then visualised one after the other. This method is called a *conditional tour*, named for its similarity in concept to projection tours (Asimov, 1985; Cook et al., 1995). Whereas a projection tour moves through projections of the data, the conditional tour moves through sections in data space. This method is especially useful when there are many predictors, or categorical predictors with many levels; essentially when it is difficult to choose sections interactively as in Chapter 4.

This chapter shows one example of a conditional tour with some simulated data, deferring the real data applications to Chapter 6.

Chapter goal

The goal of this chapter is to introduce an automated method of implementing the conditional visualisation technique described in Chapter 3. The focus will be on sensible ways to automate the choice of a sections, and ordering them so that they may be visualised in a meaningful or convenient way.

Chapter outline

Section 5.2 describes the conditional tour, the main concept of this chapter. Section 5.3 describes the *path*, the sequence of sections which are the main input to the conditional tour. Section 5.4 briefly discusses how to visualise the conditional tour. Section 5.5 describes some basic diagnostic plots that can be used to assess the usefulness of a given conditional tour configuration. Section 5.6 gives a brief overview of the software implementation of the conditional tour. Section 5.7 shows the use of the conditional tour on some simulated data. Section 5.8 concludes the chapter with a summary.

5.2 Conditional tour

The conditional tour involves visualising a number of sections in data space, one after another. Rather than choosing the section manually, as in Chapter 4, the sections for a conditional tour are all selected in advance. We give the name *path* to the sequence of $\boldsymbol{x}_C$ values defining these sections. We may order these sections so that each section visualised is near the previous section in predictor space. This is often the case, but not at all necessary. The path may be composed of points which are in no way meaningfully ordered in the predictor space.

5.3 The path

Let a *path* be defined by a sequence of points $(\boldsymbol{x}_{Ct})_{t=1}^{T}$ in the space of $\boldsymbol{x}_C$, giving a sequence of sections in data space. The path is used to guide the tour through the data space.

In the absence of any other requirements, one of the most desirable aspects of a section is proximity to observed data. Another useful section to visualise would be one near points which have a large contribution to the loss function, so problems with lack of fit might be addressed or diagnosed. Sections which show high model curvature may be of interest if the analyst is concerned about unstable extrapolations (see example in Section 6.2).

The remainder of this section suggests some different ways of selecting sections to form a path in predictor space for the conditional tour.

5.3.1 Path from clustering

A path derived from a clustering analysis allows the conditional tour to visit the parts of the predictor space where observations are located. It is important to note that we are not trying to identify actual clusters in the data. Consider two k-means (MacQueen, 1967) clusterings, one with k equal to 1 and another with k equal to the number of observations. If we visualise a section at the single centroid from the first clustering, it is unlikely to represent the entire data space well. If we visualise a section at every single observation, we are sure to see every section near observed data, but we may have to compute hundreds of thousands of sections! It seems reasonable then to use a clustering with 10-100 clusters, so that a large proportion of the data will be visited by the conditional tour, while keeping computation within feasible limits. It is worth noting that this approach can result in singleton clusters, and so it would be prudent to check the clustering and remove very small clusters before forming a path with the cluster centroids.

To produce a path from a clustering analysis, we first perform the clustering analysis, and then order the cluster centroids (or whatever data points are chosen to represent clusters). The methods mentioned here are suggestions. The field of

clustering is vast and diverse, and there may be many other methods of clustering data suited to the purpose of forming a path for the conditional tour.

Clustering quantitative data

For quantitative data, the path is chosen by first standardising and clustering the data in the space of $\boldsymbol{x}_C$ using k-means (MacQueen, 1967). The centroids of these clusters are guaranteed to be within the convex hull of the data, and so make for reasonable candidate locations for visualising sections. Theorem 2.2 from Rockafellar (2015) states that a subset of $\mathbb{R}^n$ is convex if and only if it contains all the convex combinations of its elements. A convex combination is defined as a vector sum with non-negative coefficients which sum to 1. A centroid of any cluster is a convex combination of the data, and hence must be within the convex hull of the data. By the same argument, line segments joining the cluster centres must also be contained within the convex hull of the data, so we can linearly interpolate between the centres without going outside the convex hull.

Clustering categorical and mixed data

Clustering categorical and mixed categorical and quantitative data can be accomplished with a general dissimilarity measure and a method for clustering which operates on a distance matrix. An example of this is partitioning around medoids (Kaufman and Rousseeuw, 2009) using the Gower (1971) dissimilarity coefficient. A full discussion of clustering for categorical and mixed data is beyond the scope of this work.

Ordering cluster centroids

The cluster centroids are then ordered using a seriation algorithm to find the shortest path through the centroids —dendrogram seriation (Earle and Hurley, 2015) is used here, implemented using the DendSer package (Hurley and Earle, 2013). The path is made by joining the ordered centroids with line segments. For a practical choice of path, we may take a sequence of evenly spaced points along each each line segment. For quantitative predictors, this means linear interpolation, and for categorical predictors, we simply transition from one category to the next at the midpoints on the linear scale.

Cluster centroids could also be ordered by number or density of observations in the clusters, or indeed a combination of shortest path while putting larger clusters as early as possible. See the 'lazy path length' in Earle and Hurley (2015) for an example of a path cost function combining path length and a second criterion.

Implementation: `makepath`

The `makepath` function in condvis provides a path for a conditional tour from cluster centres, as described in the previous paragraphs. The most important inputs are

- `x` A data frame containing the observed data, typically the columns of the conditioning predictors. Can contain numeric as well as factor types.
- `ncentroids` The integer number of cluster centroids to use.

5.3.2 Path from interesting observations

Another option for choosing sections to visualise is to select interesting observations; for example, observations with large contributions to a loss function. Taking a section along a predictor through a poorly fit observation allows us to see if it is feasible to make adjustments to the model's treatment of the section predictor to improve the fit. We can examine the observed data near the section to see if there is a pattern that the fitted model is missing, or if the poorly fit observation is just difficult to fit. See Section 6.4 for an example of a conditional tour with a path from poorly fit observations.

5.4 Visualising the tour

An obvious way to visualise a conditional tour, as with the grand tour (Asimov, 1985), is to produce an animation. This can be achieved by simply visualising the sections one after the other. Animation is not easily implemented in R at the time of writing, so this option has not been explored for this thesis.

The conditional tour can also be explored interactively, where instead of advancing the tour by time, the user can advance or reverse the tour at will. For ease of implementation, the conditional tour provided by condvis is controlled interactively.

5.5 Basic diagnostics

Three basic diagnostic plots are used to monitor aspects of the conditional tour.

- Plot of decile of maximum similarity weight (k) attained versus proportion of data. This plot is used to see what proportion of the data has been 'ignored' by the conditional tour. This is achieved by observing the proportion of data whose maximum k never exceeds the visibility threshold. See the upper panel of Figure 5.1 for an example of this diagnostic plot.
- Plot of approximate proportion of observations currently visible. This plot is used to give an overall impression of how exact or approximate the conditioning is. The approximate proportion of observations visible is calculated as $\sum k_i/n$. If the number is near 1, the conditional tour does not provide any improvement

over a basic marginal visualisation. If the number is too close to 0, no data can be seen on the section. In either case, the path or weighting function or both may need to be altered. See the lower panel of Figure 5.1 for an example of this diagnostic plot. The vertical line in the plot shows the current position of the conditional tour along the path.

- Data summaries of the conditioning predictors to show current values of $\boldsymbol{x}_C$. These are the same kind of plots as the condition selectors in Section 4.2.1, and shown in Figure 4.1.

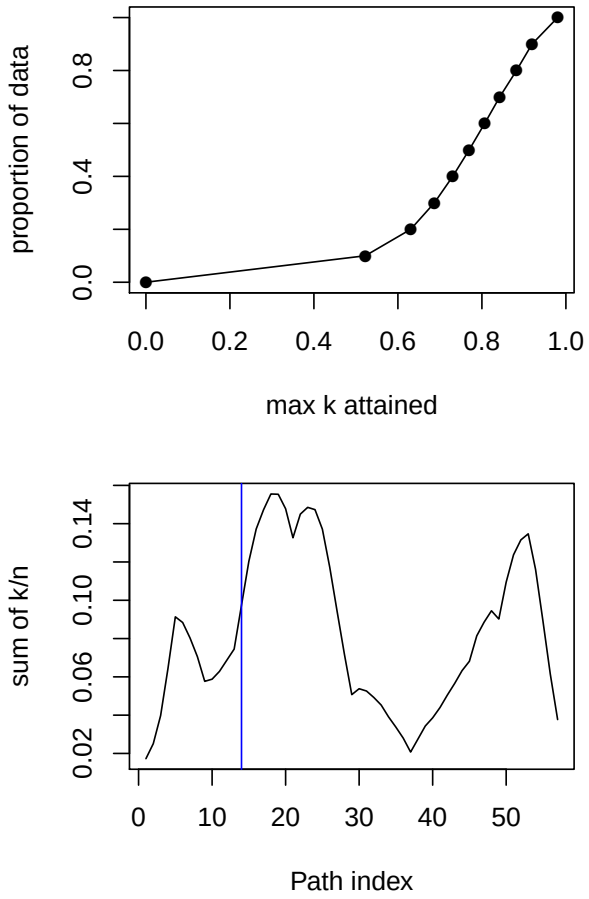


Figure 5.1: Diagnostic plots for conditional tour. The upper panel plots the proportion of data versus the maximum similarity weight attained. This can be considered an empirical cumulative distribution function for the maximum similarity weight given to the observed data. The lower panel plots $\sum_{i=1}^{N} k_i/N$ versus the path index of the tour, where k_i is the similarity weight for observation i. This shows an approximate proportion of data visible on each section of the tour. These plots can be seen in the context of a conditional tour in Figure 5.5.

5.6 Implementation: `condtour`

The conditional tour is implemented in the `condtour` function in `condvis`. Three graphics devices are opened when the function is called. One device shows the current section. A second device shows some diagnostic plots showing how much observed data is 'visited' by the tour (see Section 5.5). A third device shows univariate condition selector plots (as in Section 4.2) which highlight the location of the current section. The tour is animated by pressing the '[' and ']' keys. The arguments are quite similar to those of `ceplot`:

- `data` The data frame containing the observed data.
- `model` A list of model objects to be passed to `plotxs` (see Section 3.7).
- `path` A data frame describing the path for the tour as described in Section 5.3.
- `response` The character name of the response in `data`
- `sectionvars` The character name of the section predictor(s) in `data`.
- `conditionvars` The character name of the conditioning predictors in `data`. If `NULL`, an attempt will be made to extract these from a fitted model in `model`.
- `threshold` The value for σ as in Section 3.4.2. The default value is 1. Larger values make the conditioning more approximate, smaller values make the conditioning more strict. Larger values show more data, smaller values show less.

Figure 5.5 shows a snapshot from a conditional tour on a simulated data set.

5.7 Simulated data example: Correlation in conditioning predictors

Consider a data generating process

$$Y = \sin(10X_1) + X_2 + X_3 \tag{5.1}$$

$$X_1 \overset{iid}{\sim} U(0,1), \qquad (X_2, X_3) \overset{iid}{\sim} MVN(\mu, \Sigma)$$
$$\mu = (0,0) \quad \Sigma = \begin{pmatrix} 1 & 0.6 \\ 0.6 & 1 \end{pmatrix}$$

with no error component. We create a data set by simulating 500 observations from this process. Figure 5.2 shows a scatterplot matrix of the data. The scatterplot matrix shows that the bivariate view of Y versus X_1 does not reveal their underlying relationship, and that some of the variables are positively correlated.

As with the example in Section 4.6, we can try using trellis to reveal conditional relationships. Figure 5.3 shows a trellis plot of Y versus X_1 conditional on X_2 and X_3. This plot makes the relationship of Y and X_1 clear, but there are two problems with the choice of shingles arising from the correlation of the conditioning predictors:

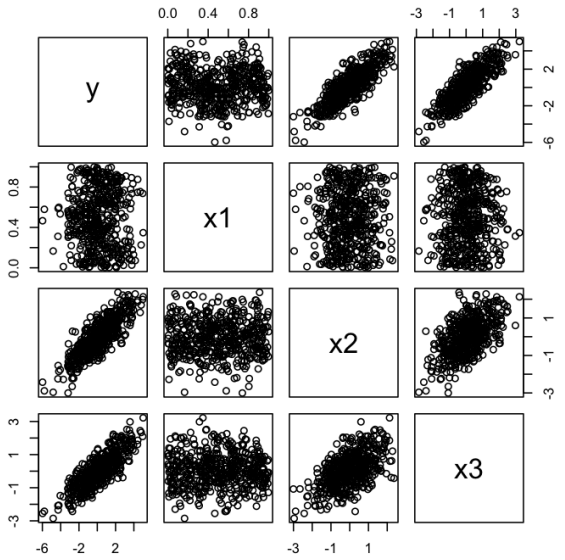


Figure 5.2: Scatterplot matrix of simulated data. The bivariate view of Y and X_1 reveals little about their relationship. X_2 and X_3 are positively correlated.

- Some panels contain very few data points, such as the top-left and bottom-right panels.
- In an effort to alleviate the problem just mentioned, some of the conditioning intervals are made very large. This means that different panels on the trellis plot are representing different amounts of the predictor space, and hence are difficult to compare.

The conditional tour offers a convenient solution to these problems. We begin by fitting a support vector machine model with a radial kernel, a flexible model which can easily handle the sinusoidal shape of the data. We then choose sections to visualise by the clustering method from Section 5.3; using 15 clusters and 3 evenly spaced points between each cluster centroid, resulting in 57 sections to visualise. The resulting path is shown in the space of conditioning predictors in Figure 5.4. This path gives a number of sections chosen in an automatic fashion, which are all near some observed data, in contrast to any grid-like approach as in trellis. We then visualise a conditional tour of sections along X_1 using this path. Figure 5.5 shows a snapshot of this conditional tour. The full tour can be seen online at youtube.com/nsr3edvblnU. The sections in the tour clearly show the relationship of Y and X_1, conditioned on X_2 and X_3, and each section has a reasonable amount of observed data nearby. We confirm this from the diagnostic plot of $\sum k_i$ for the tour (bottom-right panel of Figure 5.5), which shows a minimum value of around 5% of the data. The code for this example is in Code Snippet 5.1.

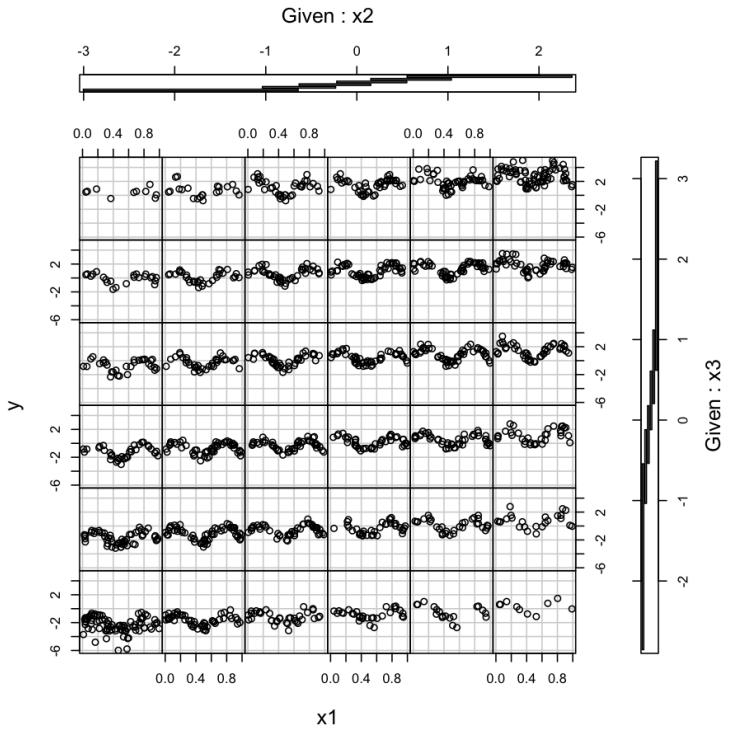


Figure 5.3: Trellis plot of Y versus X_1 conditional on X_2 and X_3. The relationship of Y and X_1 is made clear, but some panels lack data, and some panels represent very large intervals on the conditioning predictors (represented by the bars in the margins of the plot).

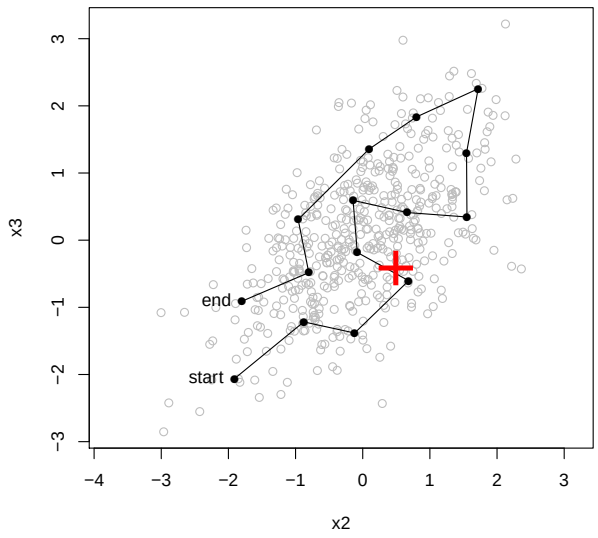


Figure 5.4: Path visualised in space of conditioning predictors. Filled circles are cluster centres, and the lines show the ordering and interpolation. The start and end points for the tour are rather arbitrary in this case. The red cross shows the point along the path giving the snapshot in Figure 5.5. Path from Code Snippet 5.1.

```
library("e1071")
load("correlated-workspace.rda") # from script in Appendix B.2

## Choose a path, the sections to visualise.

set.seed(746182481)
path <- makepath(x = d[, c("x2", "x3")], ncentroids = 15)

## Visualise sections along X1 with a conditional tour.

condtour(data = d, model = list(svm = model), path = path$path, response = "y",
  sectionvars = "x1")
```

Code Snippet 5.1: Code to visualise conditional tour on simulated data. The path is shown in Figure 5.4, and a snapshot of the conditional tour is in Figure 5.5.

5.8 Chapter summary

The conditional tour is the automated approach to visualising sections as described in Chapter 3. Sections to visualise may be selected in advance either by clustering the data and travelling from centroid to centroid, or by picking sections at interesting observations. The sections may be ordered and then visualised one after the other. This can be implemented using interactivity, animation or a combination of both.

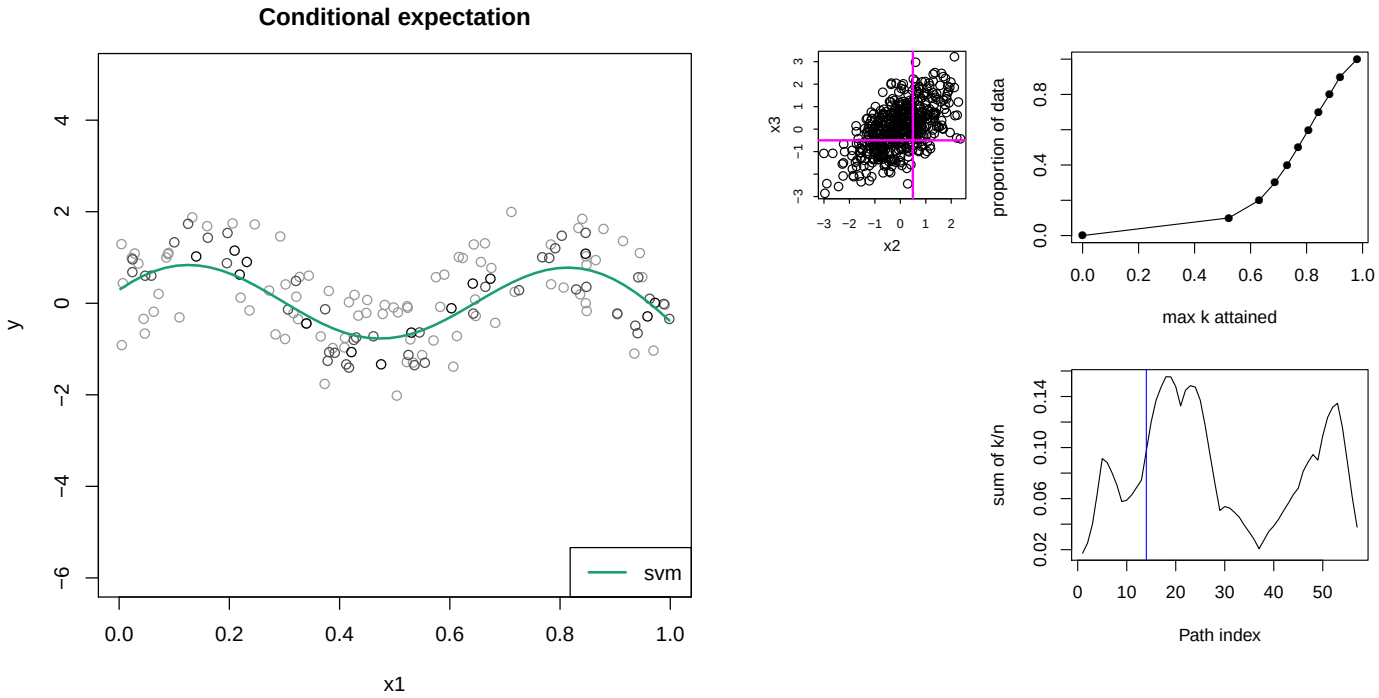


Figure 5.5: Snapshot of conditional tour. The left panel is a section in data space. The middle panel shows the current section in the space of conditioning predictors. The right panels are diagnostic plots. The upper-right panel shows how much of the data is visited by the tour. A high level of $\max(k)$ means that this proportion of data was close to at least one of the sections along the path. The lower-right panel shows the approximate proportion of data visible on each section, with the current section highlighted by a vertical line. From Code Snippet 5.1.

Chapter 6

Applications

6.1 Introduction

The conditional visualisation methods described in Chapters 4 and 5 are implemented in the condvis package in R. This chapter presents several applications of these methods to real data sets. The R code to reproduce the examples is spread between snippets labelled 'Code Snippet' in the text and scripts in Appendix B. In most cases, the code to fit the models is in the appendix, and the workspaces from these scripts are loaded by the code snippets in the chapter where the visualisation code is.

Chapter goal

The goal of this chapter is to demonstrate the use of conditional visualisation in exploring statistical models using real data.

Chapter outline

- Section 6.2 examines an additive spline model, a radial kernel support vector machine, and a neural network in a regression setting.
- Section 6.3 looks at a classification problem in six-dimensional space.
- Section 6.4 looks at the use of a gradient boosted tree model with a binary response.
- Section 6.5 examines an ensemble of gradient boosted tree models applied to count data with 280 predictors.
- Section 6.6 examines a penalised Bayesian linear regression.
- Section 6.7 presents an example where statistical models are used to explore a computationally expensive function as found in simulation studies. We can then learn about the expensive function by applying conditional visualisation to these models.

6.2 Power plant data

This example uses conditional visualisation to investigate the differences between an additive spline model, a radial kernel support vector machine, and a neural network trained on the same data.

Data

The power plant data from Tüfekci (2014) were collected for the purpose of predicting full load electrical power output of a power plant. The underlying processes are well understood, and can be modelled using differential equations —however, the computation of these differential equations is quite difficult. Tüfekci (2014) suggests machine learning techniques as an alternative and proceeds to fit several different 'black-box' models. One interesting aspect of the original article is that, in presenting these complex models to an arguably non-statistical audience, there are no graphics produced to visualise a predictor effect. The response is power output (`PE`), and there are four continuous predictors; ambient temperature (`AT`), vacuum pressure (`V`), relative humidity (`RH`), and atmospheric pressure (`AP`). There are $9,568$ observations in total, with no missing values. Figure 6.1 shows a scatterplot matrix of the power plant data.

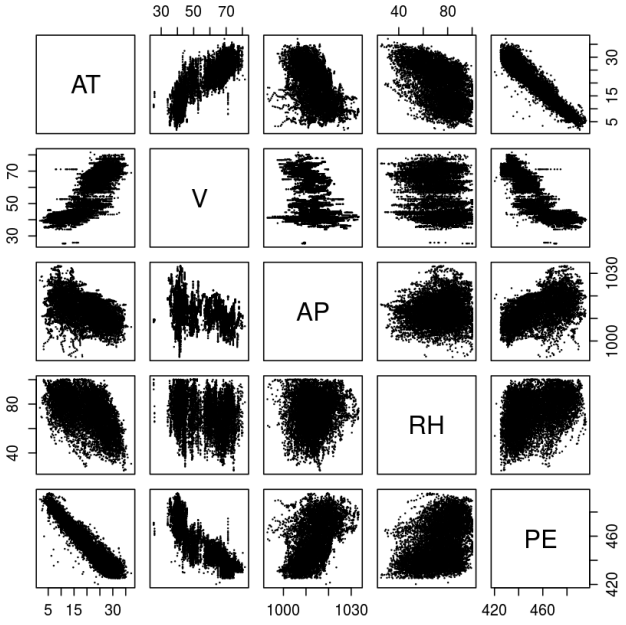


Figure 6.1: Scatterplot matrix of power plant data.

Models

We fit three models to the power plant data; an additive spline model (as in (Hastie and Tibshirani, 1990)), a support vector machine (Smola and Vapnik, 1997) with

a radial kernel, and a neural network (McCulloch and Pitts, 1943; Widrow and Hoff, 1960). We fit the additive spline model using the default settings provided by the `gam` function in `mgcv` (Wood, 2015). We fit the support vector machine using `e1071` (Meyer et al., 2014), using epsilon regression with `epsilon = 0.4`, and `gamma = 0.25`. We fit a single-hidden-layer neural network using `nnet` (Ripley, 2016; Venables and Ripley, 2002). The network has 30 nodes, and we train the network with `decay = 0.5`, stopping training at 300 iterations. These parameter choices give three models with similar train errors, but not necessarily the best predictive models. The parameters for the additive spline model are chosen automatically by penalized likelihood methods within `mgcv`. The parameters for the support vector machine and the neural network were then chosen by hand to provide a similar training error. The code to fit the models is in Appendix B.3.

Visualisation

The fitted model in this example can be considered as a surface in five-dimensional Euclidean space. This would be rather difficult to visualise directly, but using conditional visualisation we can produce simple, interpretable sections in data space.

Figure 6.2 shows an interactive conditional expectation plot, with a section along the `AT` predictor ($x_S = \{\texttt{AT}\}$, $\boldsymbol{x}_C = \{\texttt{AP}, \texttt{RH}, \texttt{V}\}$), with all three fitted models shown. The conditioning predictors have been arranged as in Section 4.3. The observed data points on the section are coloured according to their similarity weight as defined in Equation 3.5 (darker if near the section, lighter if further away). The threshold is set to 0.3 (see Section 3.4). We can see that the fitted models capture the shape of the data well, and that the models are all quite similar to each other on this section. Figure 6.3 shows another section along `AT` from a different part of the predictor space, where the models exhibit different extrapolation behaviour. Near this section, the observed data are distributed in a restricted range of `AT`, and so we are seeing extrapolations for the fitted models to parts of the predictor space where no observations have been made. Although the fitted models are quite near each other in the region near observed data, they are further apart in the region away from observed data, most notably the kernel support vector machine. This means we have models that can produce very different predictions for extrapolations to new cases, despite having similar training errors. This fact is well known to most experts in statistical modeling, but would certainly not be obvious to the broader range of end-users of statistical models. A graphic like Figure 6.3 can make this point quite directly to a non-expert audience. Figure 6.4 shows another section where the fitted models differ considerably in extrapolation, this time along the `AP` predictor.

Figure 6.5 shows a three-dimensional section taken along the `V` and `AT` predictors ($\boldsymbol{x}_S = \{\texttt{V}, \texttt{AT}\}$, $\boldsymbol{x}_C = \{\texttt{AP}, \texttt{RH}\}$), with the regression surface from the support vector machine. The residuals are shown in this plot as line segments to give extra context to the observed data. This graphic allows us to consider the interaction of `V` and `AT`

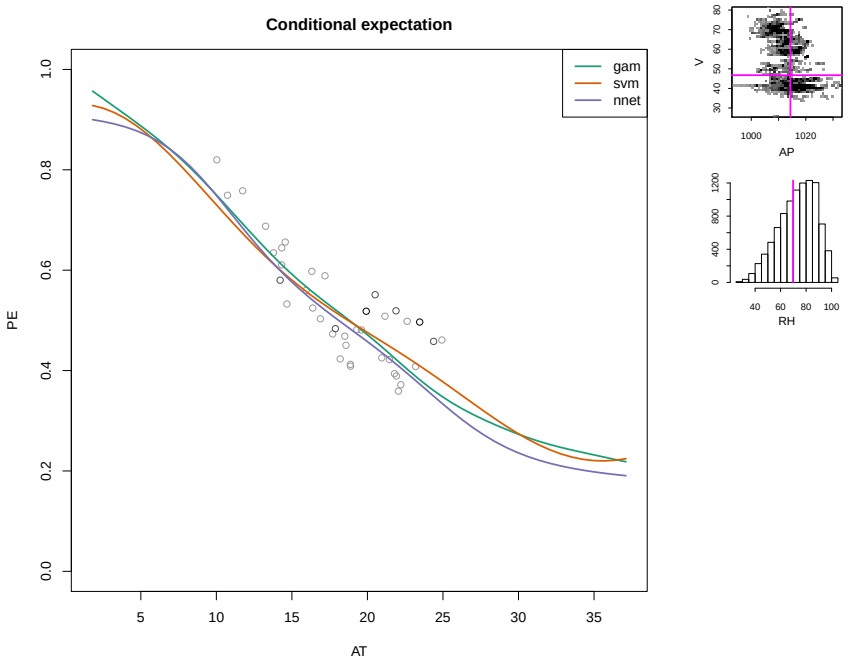


Figure 6.2: Power plant data. Snapshot from call to `ceplot` in Code Snippet 6.1. Section along `AT`, showing similar model predictions. From Code Snippet 6.1.

given by the model, and certainly this section suggests that the effect of `V` changes from relatively flat at high values[1] of `AT` to a pronounced curvature at low values of `AT`. This is what the model is telling us, but what is the observed data telling us? The observed values of `V` and `AT` near this section seem to be correlated with each other, and we can see that most of the observed data are located along a relatively narrow band across the fitted regression surface in this section. In this way, the changing curvature that we observe in the fitted model in Figure 6.5 is not directly supported by nearby observed data, and so might be no more than an artefact of the model's underlying kernel smoothing structure. This demonstrates how we can visualise an interaction on a three-dimensional section, and the importance of enriching section visualisations with observed data.

Figure 6.6 shows a conditional tour exploring sections along `AT` throughout the whole predictor space. From the top-right diagnostic plot in Figure 6.6, we can see that the tour ignores approximately 20% of the data. From the bottom-right diagnostic plot, we can see that each sections displays somewhere in the region of 0.5 – 1.5% of the data.

[1]Please note that the value of `AT` is increasing from the back to the front of the plot.

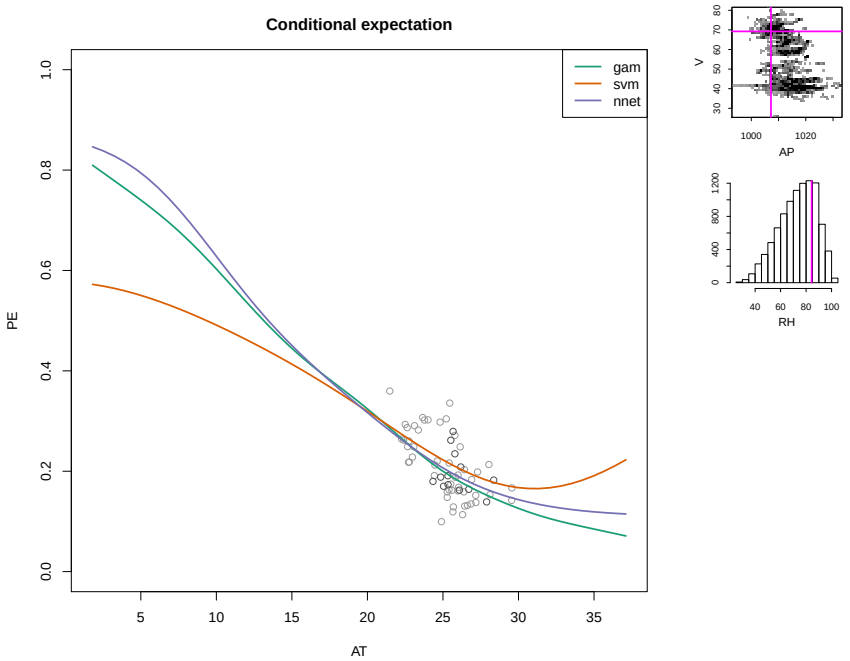


Figure 6.3: Power plant data. Snapshot from call to `ceplot` in Code Snippet 6.1. Section along `AT`, showing differing model predictions in extrapolation. From Code Snippet 6.1.

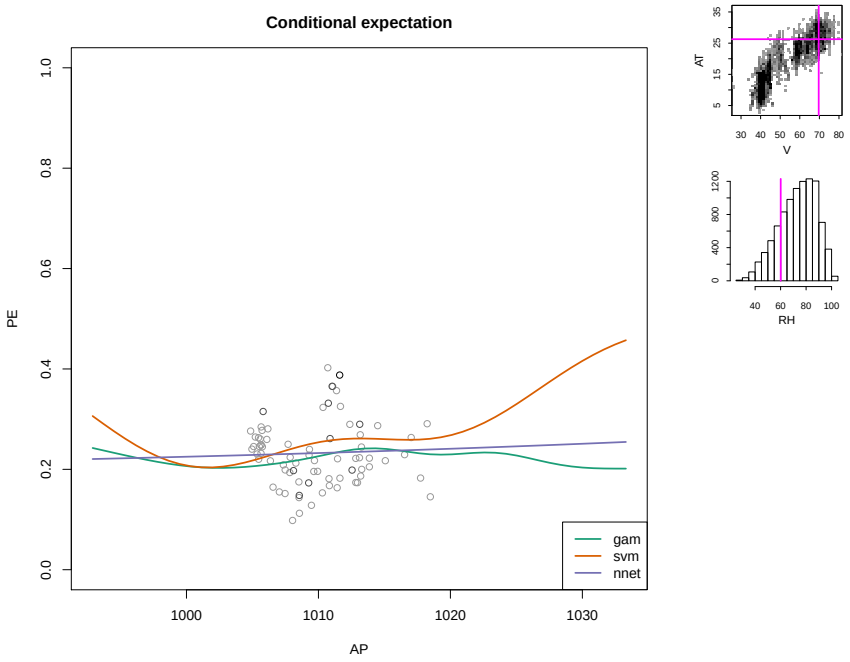


Figure 6.4: Power plant data. Snapshot from call to `ceplot` in Code Snippet 6.1. Section along `AP`, showing differing model predictions in extrapolation. From Code Snippet 6.1.

```
library("condvis")
library("mgcv"); library("e1071"); library("nnet")
load("powerplant-workspace.rda") # from script in Appendix B.3

## Look at 2-D sections along AT showing all three models. Change ‘sectionvars‘
## to AP, V or RH to take sections along those predictors.

ceplot(data = powerplant, model = models, response = "PE", sectionvars = "AT",
  threshold = 0.3)

ceplot(data = powerplant, model = models, response = "PE", sectionvars = "AP",
  threshold = 0.3)

## Look at 2-D sections along AT for the neural network, across the whole
## predictor space using a conditional tour.

set.seed(746182481)
path <- makepath(powerplant[, c("AP", "V", "RH")], ncentroids = 35)$path

condtour(data = powerplant, model = models["nnet"], path = path,
  response = "PE", sectionvars = "AT", threshold = 0.5)

## Look at 3-D sections along predictor pairs. Get an impression of 2-way
## interactions for the support vector machine.

ceplot(data = powerplant, model = models["svm"], response = "PE", sectionvars
  = c("AT", "V"), threshold = 0.2, view3d = TRUE)
```

Code Snippet 6.1: Power plant data. Related visualisations in Figures 6.2, 6.3, 6.4, 6.5, and 6.6.

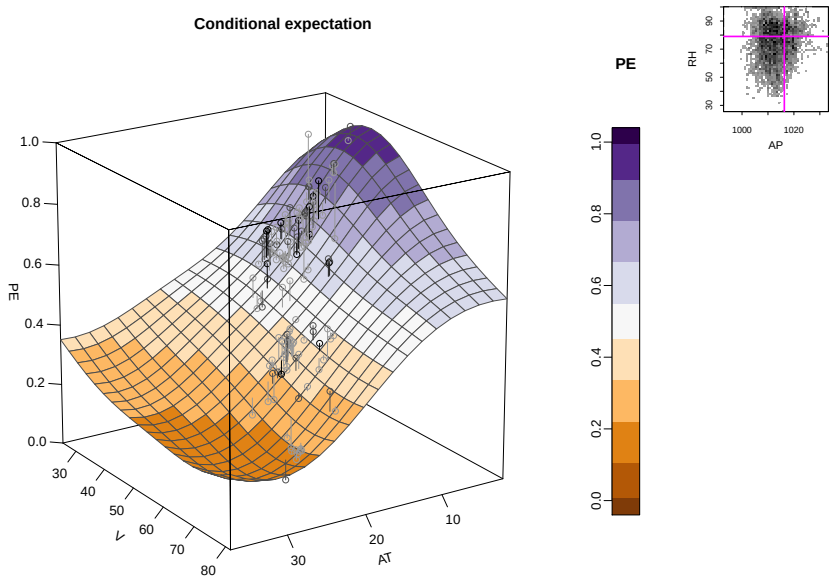


Figure 6.5: Power plant data. Snapshot from call to `ceplot` in Code Snippet 6.1. Three-dimensional section along `AT` and `V`. From Code Snippet 6.1.

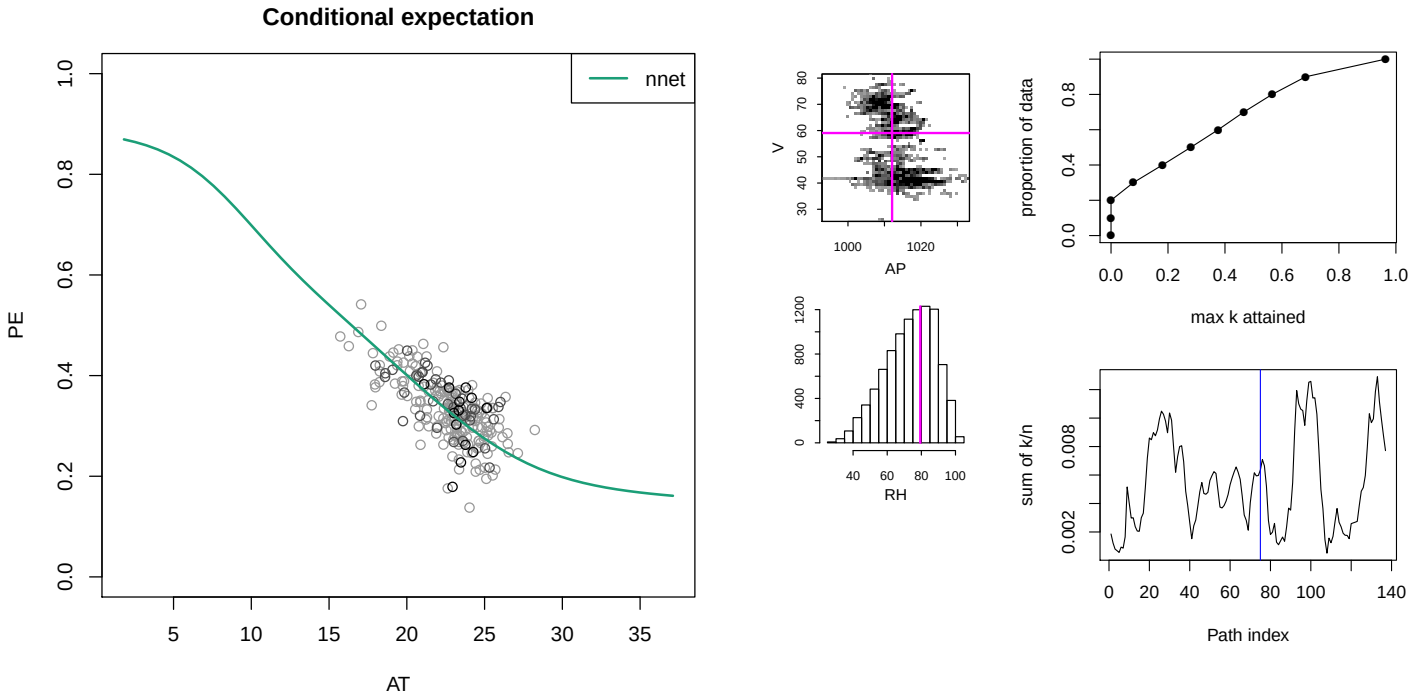


Figure 6.6: Snapshot of conditional tour on power plant data. The path is given by k-means clustering, attempting to visit most of the predictor space occupied by data. The left panel shows a section in data space. The middle panels show the current section in the space of conditioning predictors. The top-right panel shows how much of the observed data is 'visited' by the tour. The bottom-right panel shows approximately how many observations are near each section given by the path. From Code Snippet 6.1.

6.3 Wine data

This example uses conditional visualisation to explore a classifier in six-dimensional predictor space.

Data

The wine data (Aeberhard et al., 1992) are the results of a chemical analysis of wines grown in the same region of Italy but derived from three different cultivars. The basic task is to produce a classifier which can take the numeric chemical measurements and identify the correct cultivar. The response is a categorical variable denoting the cultivar by 3 levels. There are 13 continuous predictors describing the chemical measurements; alcohol content, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, non-flavanoid phenols, proanthocyanins, colour, hue, OD280/OD315 of diluted wine, and proline. There are 178 observations, with no missing values. Figure 6.7 shows a scatterplot matrix of six predictors from the wine data, with the cultivar shown by colour.

Model

We choose six of the predictors —arbitrarily, to reduce dimensionality for a simpler discussion— and train three classifiers on the wine data; a support vector machine (Cortes and Vapnik, 1995) with a radial kernel, a random forest (Breiman, 2001), and a gradient boosted tree (Friedman, 2001). In each case, we choose the relevant parameters by 5-fold cross-validation, with the help of the caret package (Kuhn, 2015). The code to fit the models is in Appendix B.4.

Visualisation

It is difficult to comprehend how a classifier assigns regions to different classes in six dimensions, but visualising this on a two-dimensional section is straightforward ($|S| = 2$). We can then use interactive conditional expectation plots to explore aspects of the fitted models which might be implicitly known by expert statisticians, but not at all familiar to non-experts. This example does not involve a large number of observations, so we will concentrate more on examining the behaviour of the classifiers themselves.

We can see that the support vector machine's radial kernel gives a classifier with smooth boundaries, and many spherical shapes intersecting in high-dimensional space. Figure 6.8 shows a section that we might expect to see from such a classifier. Figure 6.9 shows some of the more peculiar classification boundaries that can arise from a radial kernel method in six dimensions. On these sections, the colour of the circle representing an observation is controlled by its similarity weight as in Equation 3.5(darker circles near the section, lighter circles away from the section), and the circle is filled according to the observed value. We might be tempted then

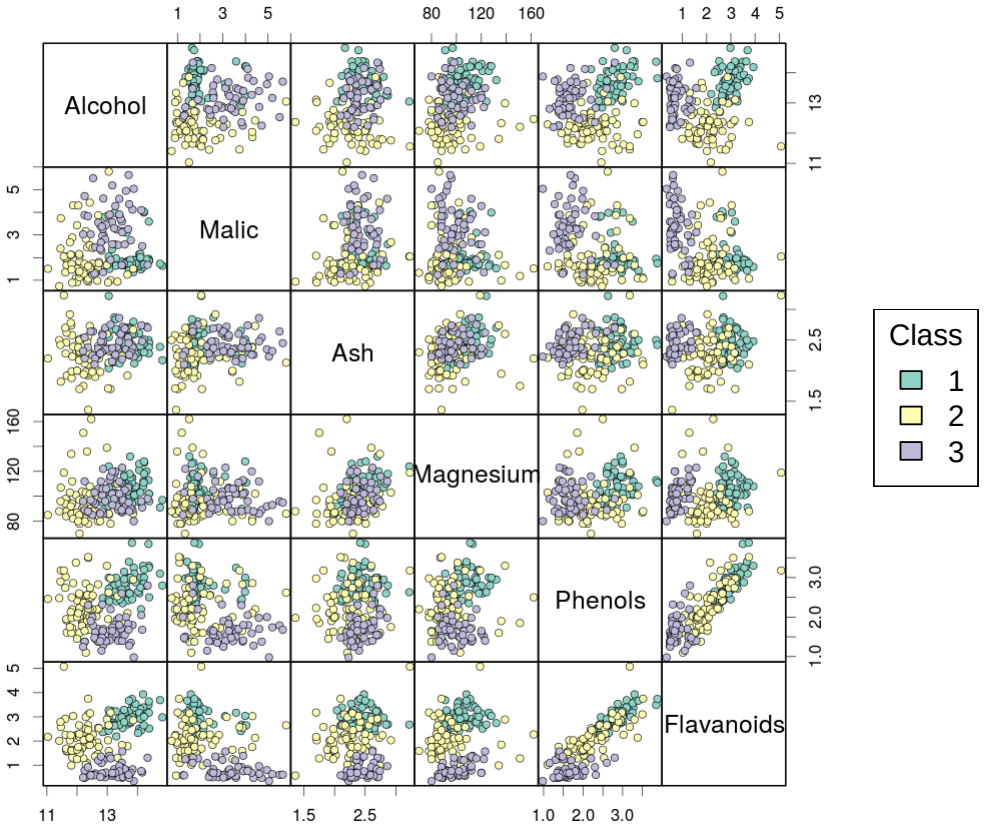


Figure 6.7: Scatterplot matrix of six variables from the wine data. Class of cultivar is shown by colour according to the legend.

to conclude that an observation whose filled colour does not match the predicted class is misclassified. This is not necessarily true, since the observation may not lie exactly on the section, and the classification boundaries may change quite abruptly across the space of conditioning predictors.

The random forest model (Figure 6.10), on the other hand, makes for more blocky and square classification boundaries. The boundaries of the random forest are also quite rough, with small 'islands' of a predicted class being separated from the main region for that class, on some sections through the predictor space. Of course, these regions may not be disjoint in the full data space, but it is indicative of rough and complex classification boundaries.

The gradient boosted tree (Figure 6.11) has a similar appearance to the random forest model, but has smoother class boundaries, owing to the ensembling inherent in gradient boosting. Looking only at predicted class ignores any uncertainty the model might have in making its prediction, so it is also useful to look at predicted class probabilities if the model provides them. We can produce a grid of small bar charts —similar to embedded plots (Grolemund and Wickham, 2015)— to represent predicted class probabilities on the section. An example of this is shown for the

gradient boosted tree in Figure 6.12. This section shows considerable uncertainty in the predictions for the upper-right part of the section, whereas the predicted class alone would be 2 on this whole section. Observed data is not shown on this kind of section, to avoid the graphic becoming too visually 'busy.'

Figure 6.13 shows a conditional tour of sections along `Magnesium` and `Flavanoids` across the whole predictor space. The top-right diagnostic plot tells us that approximately 20% of the data are ignored by our tour configuration (choice of path and threshold). We can see this from the top-right panel of Figure 6.13, where the maximum similarity weight (k_i) attained is zero for 20% of the data. The bottom-right diagnostic plot tells us that the sections show roughly between 1% and 6% of the observed data at any given time.

```
library("condvis")
library("kernlab"); library("randomForest"); library("gbm")
load("wine-workspace.rda") # from script in Appendix B.4

ceplot(data = wine, model = final.svm, response = "Class",
  sectionvars = c("Phenols", "Malic"), conditionvars = c("Ash", "Magnesium",
  "Alcohol", "Flavanoids"), threshold = 1.2)

ceplot(data = wine, model = final.rf, response = "Class",
  sectionvars = c("Alcohol", "Malic"), conditionvars = c("Ash", "Magnesium",
  "Phenols", "Flavanoids"), threshold = 1.2)

ceplot(data = wine, model = final.gbm, response = "Class",
  sectionvars = c("Alcohol", "Malic"), conditionvars = c("Ash", "Magnesium",
  "Phenols", "Flavanoids"), threshold = 1.2)

ceplot(data = wine, model = final.gbm, response = "Class",
  sectionvars = c("Alcohol", "Ash"), conditionvars = c("Malic", "Magnesium",
  "Phenols", "Flavanoids"), threshold = 1.2, probs = TRUE)

## Visualise sections along Magnesium and Flavanoids for the support vector
## machine across the whole predictor space using the conditional tour.

set.seed(746182481)
path <- makepath(wine[, c("Alcohol", "Ash", "Malic", "Phenols")], ncentroids =
  15)$path

condtour(data = wine, model = final.svm, path = path, response = "Class",
  sectionvars = c("Magnesium", "Flavanoids"), conditionvars = colnames(path))
```

Code Snippet 6.2: Wine data. Related visualisations are in Figures 6.8, 6.9, 6.10, 6.11, 6.12, and 6.13. Figure 6.12 uses the additional parameter `probs = TRUE`.

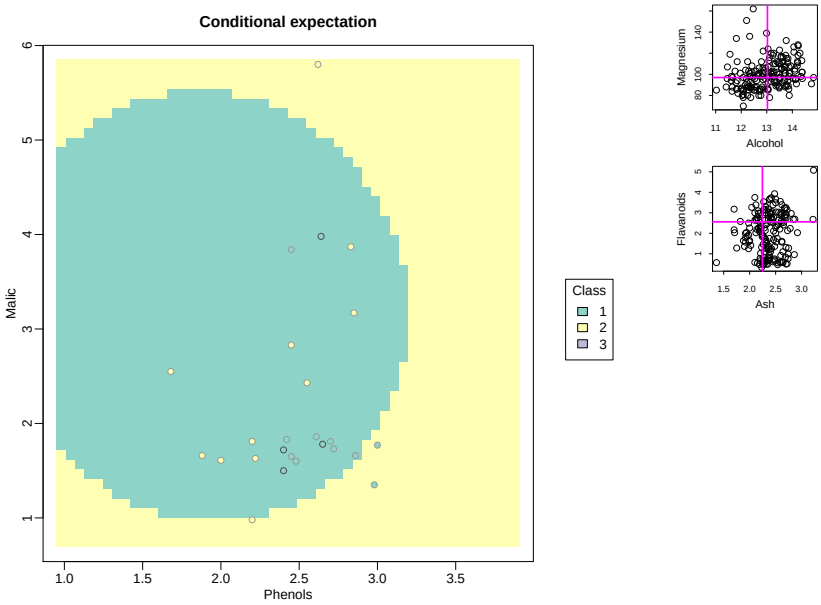


Figure 6.8: Wine data. Snapshot from Code Snippet 6.2. Section on the support vector machine classifier with radial kernel, showing spherical classification boundaries.

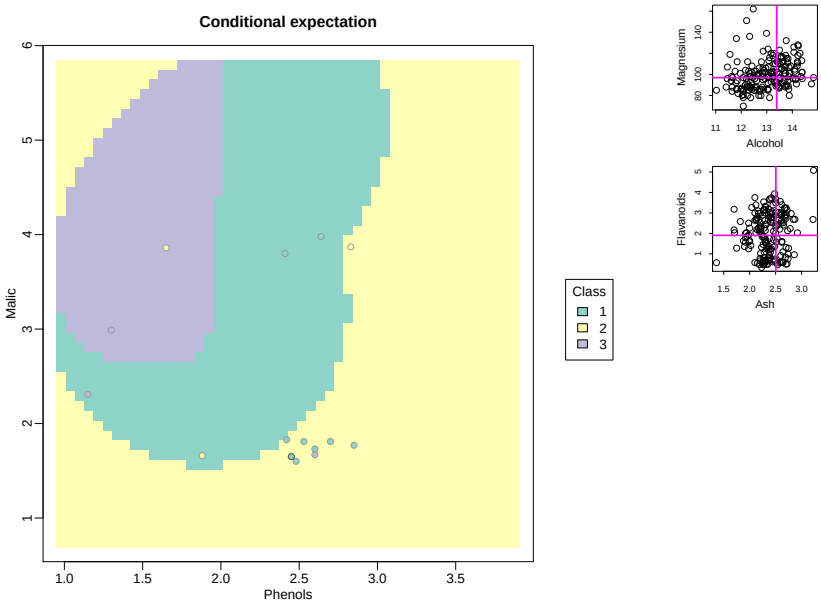


Figure 6.9: Wine data. Snapshot from Code Snippet 6.2. Section on the support vector machine classifier with radial kernel, showing peculiar shapes of the classification boundaries. Same model as Figure 6.8.

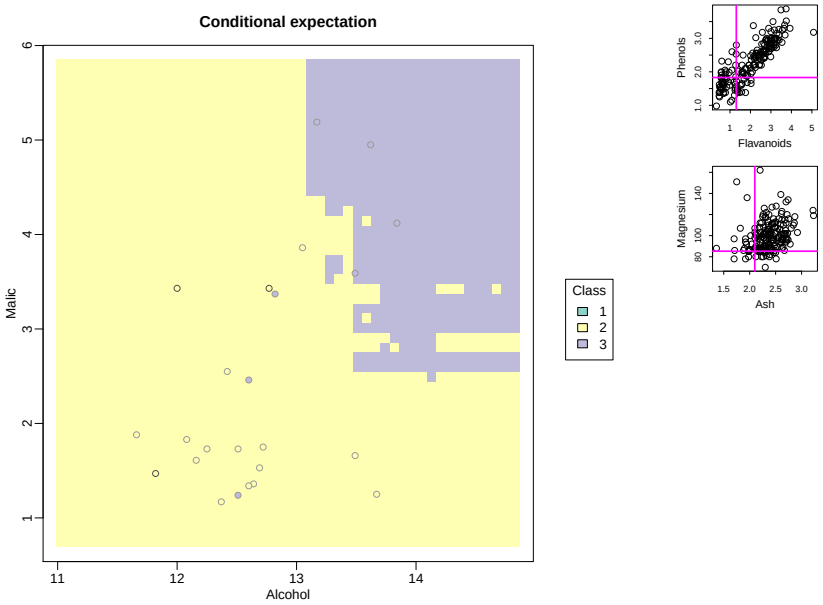


Figure 6.10: Wine data. Snapshot from Code Snippet 6.2. Section on the random forest classifier. Note the rough classification boundary.

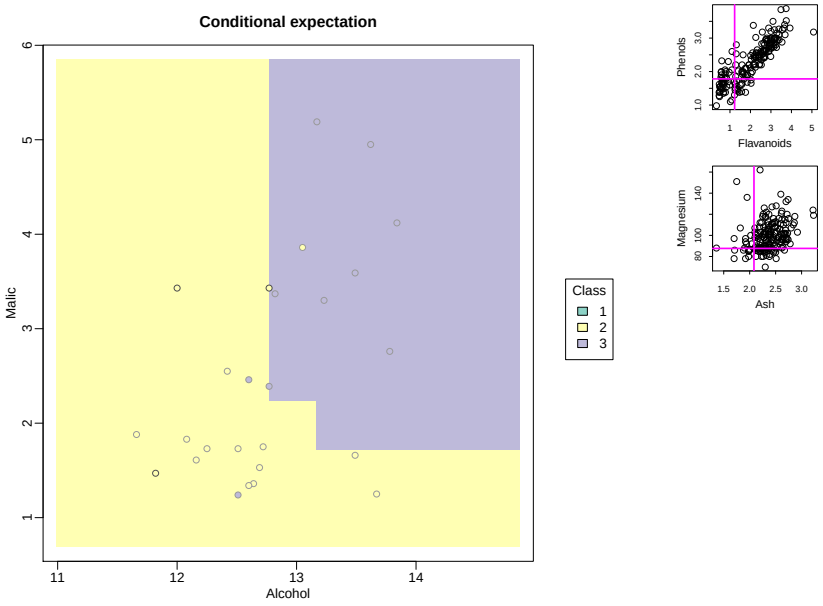


Figure 6.11: Wine data. Snapshot from Code Snippet 6.2. Section on the gradient boosted tree classifier. Note the classification boundaries are smoother than those of the random forest in Figure 6.10, where the section is in the same part of data space.

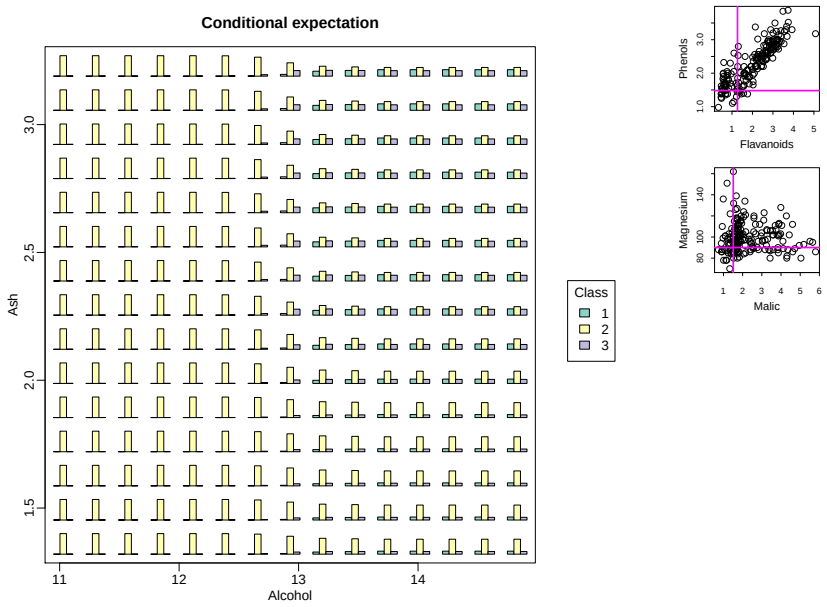


Figure 6.12: Wine data. Snapshot from Code Snippet 6.2. This section shows the actual predicted class probabilities from the gradient boosted tree classifier on a section by setting `probs = TRUE`. The predicted class probabilities in the upper-right part of this section suggest uncertainty in the class prediction.

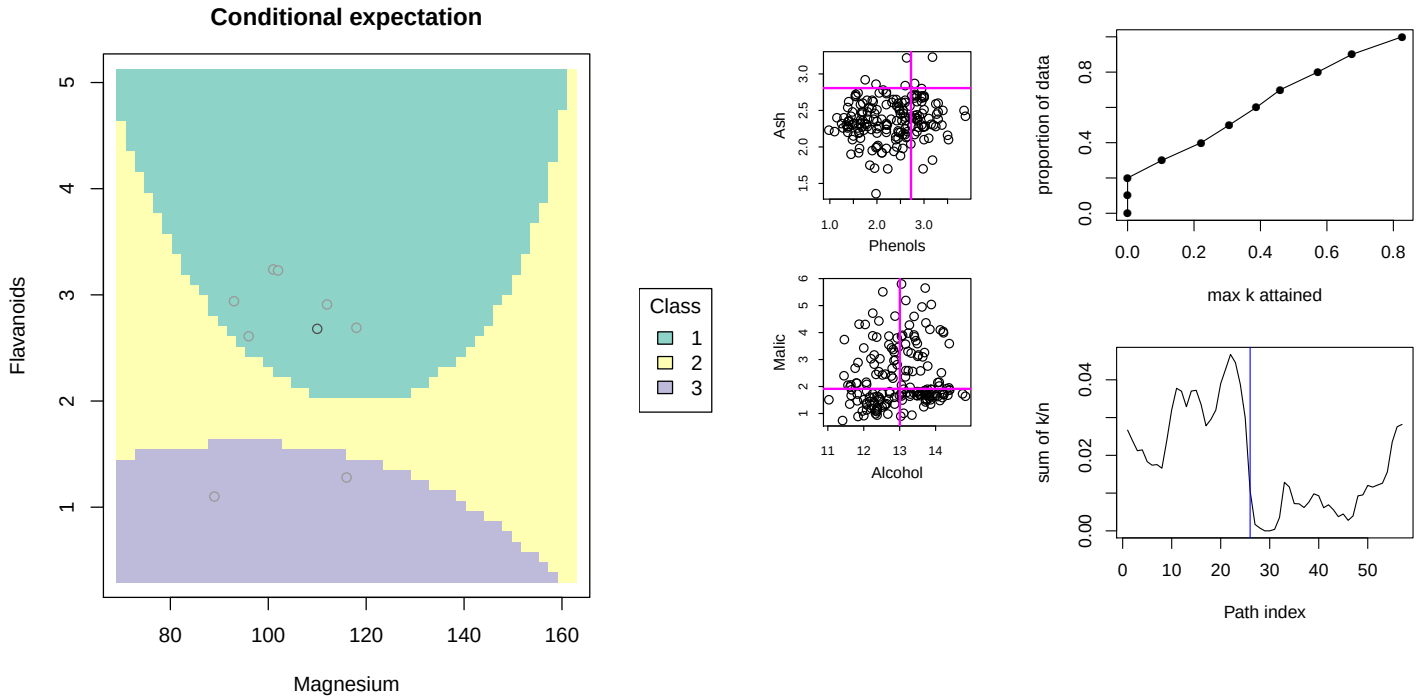


Figure 6.13: Snapshot of conditional tour on wine data. The path is given by k-means clustering, attempting to visit most of the predictor space occupied by data. From Code Snippet 6.2.

6.4 Credit card defaults data

This example uses conditional visualisation to explore a gradient boosted tree model.

Data

Yeh and Lien (2009) present a comparison of data mining techniques on credit card defaults data. We use the data for an example of a moderately sized machine learning exercise, with 30,000 observations on 23 predictors describing gender, marital status, education, age and various measures of payment history on an account. Defaults are given by a binary response (default = 1), but the probability of default is of primary interest in risk management, rather than classification. We hold out 8,000 cases at random for testing, and train a model on the other 22,000 cases.

Model

We train a gradient boosted tree (Friedman, 2001) on 22,000 cases from the credit card data. We set the parameters for shrinkage, minimum observations in a node, and the interaction depth to reasonable values and choose the number of boosting iterations by 5-fold cross validation. The final model is trained using the `gbm` package (Ridgeway, 2013), with 402 boosting iterations, `shrinkage = 0.03`, `interaction.depth = 4`, `n.minobsinnode = 10`, and `bag.fraction = 0.7`. This gives a model that achieves a logarithmic loss[2] of 0.4162 on the training data, and 0.4258 on the 8,000 test cases that we held out. The code to train the gradient boosted model is in Appendix B.5.

Visualisation

First, we apply a jitter to the binary response in order to alleviate overplotting when visualising a large number of observations. It is difficult to explore this high-dimensional predictor space by manually choosing sections, hoping that observed data may be nearby. This is a symptom of the well-known curse of dimensionality. Figure 6.14 shows an interactive conditional expectation plot, exploring the twenty most important predictors (where importance is given by the `gbm` package, using the reduction in the total loss function attributable to each predictor). While it is easy to interpret the model with this graphic, it is difficult to find sections near observed data, both because of the high-dimensionality of the predictor space and the strong skew of some of the conditioning predictors. Indeed, Figure 6.14 shows no observations nearby.

As an alternative way to explore the space, we use a conditional tour (Section 5) which visits some of the worst fit observations and see how the model is failing

[2]Logarithmic loss (abbreviated to logloss, and equivalent to cross entropy) is a loss function used in binary classification problems, defined as $-\frac{1}{n}\sum_i [y_i \log(p_i) + (1-y_i)\log(1-p_i)]$, where $y_i \in \{0,1\}$ is the observed class, and $p_i \in (0,1)$ is the predicted class probability.

in these situations. Figure 6.15 shows a snapshot of such a conditional tour. The poorly fit observation in this case is a default which was assigned a low probability of default. There are around 100 observations near this section (see bottom-right panel of Figure 6.15, the vertical blue line gives the current section, and $0.005n \approx 100$), many of which are defaults. The nearby observed data do not suggest that a different curvature for the this section predictor could improve the fit to the data, rather that we need to look along some other dimension to separate these cases. The code for these visualisations is in Code Snippet 6.3.

```
library("condvis")
library("gbm")
load("creditcard-workspace.rda") # from script in Appendix B.5
train$default <- jitter(train$default, amount = 0.05)

## Take a 2-D section on the most important predictor, conditioning on the
## next 14 most important predictors.

ceplot(data = train, model = list(gbm = model), response = "default",
  sectionvars = varimp[1], conditionvars = varimp[c(2:15)], view3d = TRUE)

## Tour through sections taken on the same predictors, at the locations of the
## worst fit observations, according to log loss contribution.

logloss.order <- order(logloss.contrib.train)
condtour(data = train, model = list(gbm = model), path = train[logloss.order[
  1:30], varimp[2:15]], response = "default", sectionvars = varimp[1],
  conditionvars = varimp[2:15], view3d = TRUE)
```

Code Snippet 6.3: Credit card data. Related visualisations in Figures 6.14 and 6.15.

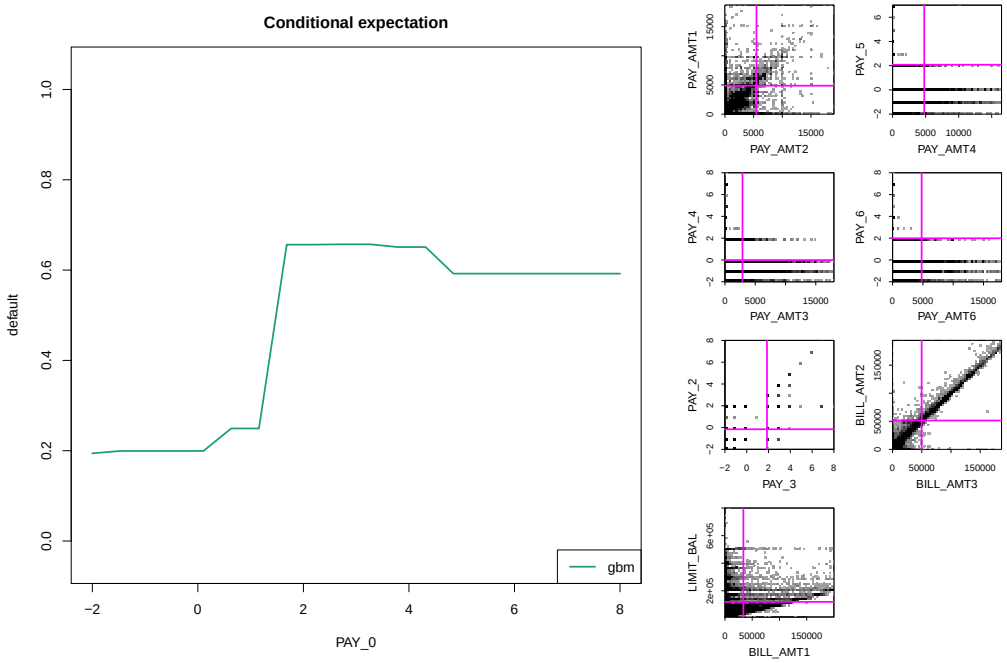


Figure 6.14: Credit card data. Snapshot of interactive conditional expectation plot from Code Snippet 6.3. It is difficult to choose sections which are near observed data in this predictor space .

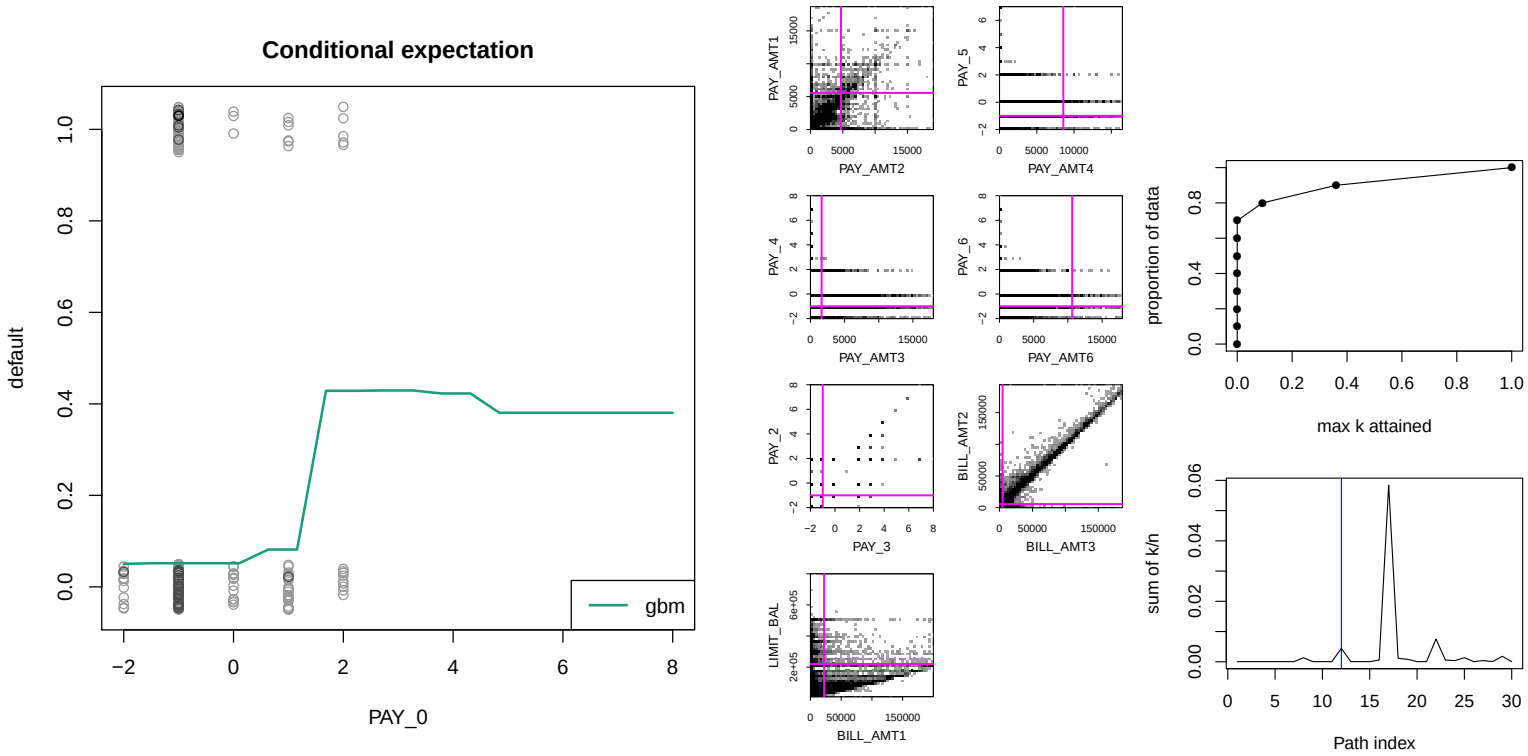


Figure 6.15: Snapshot of conditional tour on credit card data. The tour allows us to locate interesting sections; in this case, a section through a poorly fit observation. The left panel shows a section in data space. The middle panels show the section in the space of conditioning predictors. The top-right panel is not of much interest here, as we are not trying to explore the whole data space. The bottom-right panel shows approximately how many observations are near each section. Many of these poorly fit observations are clearly isolated, but some have a large number of other observations nearby, as can be seen from the peaks in the bottom-right panel. From Code Snippet 6.3.

6.5 Blog comments data

This example uses conditional visualisation to explore an ensemble of gradient boosted tree models trained on a large data set.

Data

Buza (2014) presents an application of machine learning to the prediction of blog feedback. The original task of interest is to predict the number of comments a blog will receive in the next 24 hours (relative to the base time, when measurements are taken).

There are $52,397$ observations on 280 predictors, which are described by Buza (2014) as follows.

- 62 basic features – number of links and feedbacks in the previous 24 hours relative to base time; number of links and feedbacks in the time interval from 48 hours prior to base time to 24 hours prior to base time; how the number of links and feedbacks increased/decreased in the past (the past is seen relative to base time); number of links and feedbacks in the first 24 hours after the publication of the document, but before base time; aggregation of the above features by source.
- 200 textual features – measuring occurrence of a pre-determined bag of words.
- 14 weekday features – binary indicator features that describe on which day of the week the main text of the document was published and for which day of the week the prediction has to be calculated (i.e., the base time).
- 4 parent features – a blog post P is a parent of blog post B, if B is a reply (trackback) to blog post P. Parent features are the number of parents, minimum, maximum and average number of feedbacks that the parents received.

Models

Rather than addressing the original prediction problem for number of comments, we train a classifier ensemble for the simpler task of predicting the probability of there being *any* comments on a blog over the next 24 hours. The code for training these models is in Appendix B.6.

We first create a new binary response variable which is 1 if there are any comments, and 0 otherwise. We split the training data into 5 randomly selected folds, and train 5 boosted tree models with a logarithmic loss objective function, holding out one fold each time. We set the shrinkage (0.03), interaction depth (6), minimum number of observations in a node (1), and other parameters to reasonable values, and choose the number of boosting iterations for each model which produces the best logarithmic loss on the held-out fold. Rather than fitting another boosted model

with the number of iterations chosen from these models, we directly ensemble these models by taking an equally weighted mean of their predictions. The final result is a model ensemble giving predicted probability of there being any comments on a blog. This is called a *cross-validated committee* by Parmanto et al. (1996).

The gradient boosted tree models for this example are trained using the `xgboost` package (Chen et al., 2016b), which provides an interface to the XGBoost library (Chen and Guestrin, 2016). The XGBoost library provides fast, efficient boosting algorithms which have become very popular amongst competitors in open machine learning competitions such as those run by Kaggle (www.kaggle.com) or Numerai (numer.ai).

In order to clarify, we are not considering the boosted models as ensembles, though they are indeed ensembles of trees themselves.

Visualisation

We can visualise the entire ensemble directly on a section along a single predictor (as discussed in Section 7.3). We apply a jitter to the binary response in order to alleviate overplotting when visualising a large number of observations. Figure 6.16 shows a section along the most important predictor (according to the measure of variable importance provided by `xgboost`, using frequency in trees and logloss improvement). It is easy to visualise the ensemble as individual curves on this section, using colour to identify each model. The modelled effect of predictor `V52` (number of comments received in the last 24 hours) is easy to interpret on this section. For `V52 = 0` the predicted probability of any comments in the next 24 hours is around 0.2, with the predicted probability increasing to a plateau of around 0.8 for values of `V52` above 20. Further to this, we can see that the five models from our cross-validated committee are essentially giving the same effect shape on this section, with some variability.

Figure 6.17 shows a similar section to Figure 6.16, but this time we have added the average ensemble prediction. This visualisation really shows the value of ensembling on the level of predictor effects. Clearly, these models have returned fitted predictor effects with substantial variability, even for the most important predictor, and so it makes sense to ensemble their predictions.

As with the example in Section 6.4, some of the conditioning predictors are rather skewed, so the basic condition selector plots are not easy to use in this case. There are a large number of observations, so we prefer two-dimensional histograms to scatterplots for the condition selectors. Due to the skew present in many of the conditioning predictors, we must also be careful to choose sensible bins and colouring for the two-dimensional histogram. We achieve this by capping the observation counts for the bins with the `fullbin` parameter that is passed to `plotxc`. This loses some information regarding the precise density of observations in areas with a high density of observations, but allows us to see areas of low density more easily.

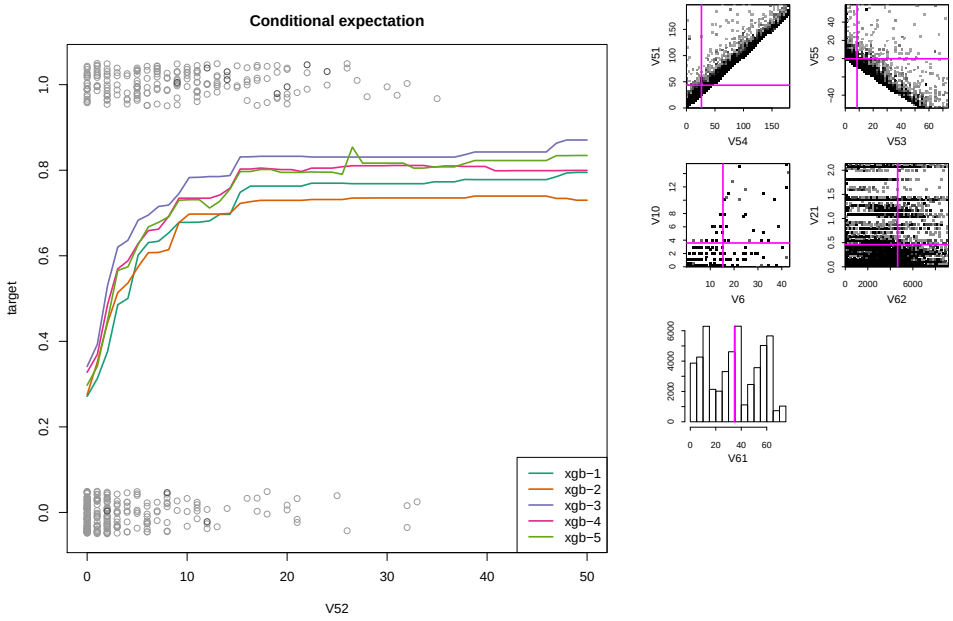


Figure 6.16: Blog data. Snapshot from Code Snippet 6.4. We visualise the model ensemble directly on the section, with a single curve for each model in the ensemble.

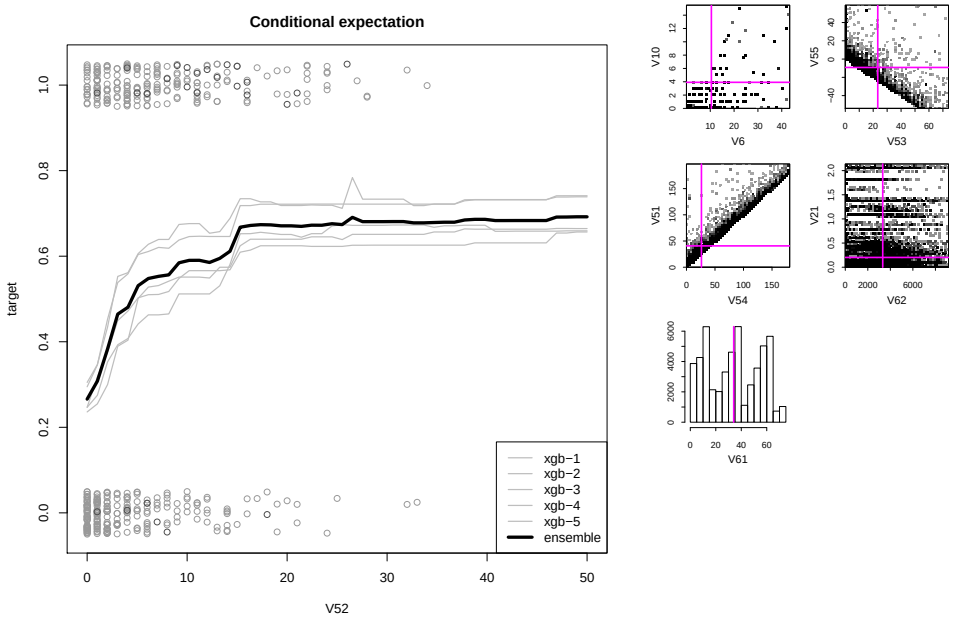


Figure 6.17: Blog data. Snapshot from Code Snippet 6.4. On this section, we show the ensemble prediction with a thick black line. We can clearly see the value of ensembling here, with such broad model disagreement.

```
library("condvis")
library("xgboost")
load("blog-workspace.rda") # from script in Appendix B.6

train$target <- jitter(train$target, amount = 0.05)

ceplot(train, classifiers, "target", varimp$Feature[1], varimp$Feature[2:10],
  threshold = 0.6, xcplotpar = list(fullbin = 10), xsplotpar = list(xlim =
  c(0, 50)))

model.colours <- c(rep("gray", length(classifiers)), "black")
model.lwd <- c(rep(1.5, length(classifiers)), 4)

ceplot(train, ensemble, "target", varimp$Feature[1], varimp$Feature[2:10],
  threshold = 0.6, xsplotpar = list(xlim = c(0, 50)), modelpar = list(col =
  model.colours, lwd = model.lwd), xcplotpar = list(fullbin = 10))
```

Code Snippet 6.4: Blog data. Related visualisations in Figures 6.16 and 6.17.

6.6 Prostate data

This example looks at a penalised Bayesian linear regression, applied to data concerning prostate cancer. This is a straightforward example, and mainly serves to demonstrate how the conditional visualisation in this thesis extends to a Bayesian model involving MCMC samples from the posterior distribution (see Section 3.3.3). In this way, we can visualise Bayesian models alongside classical models in data space.

Data

The prostate data (Stamey et al., 1989) are used to demonstrate variable selection and shrinkage methods by Hastie et al. (2009). The goal is to predict the log of prostate specific antigen (`lpsa`) from a number of clinical measurements. There are 97 observations on 9 clinical measurements.

Model

We apply Bayesian inference to a penalised linear regression model for the prostate data, preferring MCMC sampling in order to demonstrate visualisation of models described by posterior parameter samples. We use Stan (Stan Development Team, 2015) for the MCMC sampling, via the rstan package (Stan Development Team, 2016). We set up a ridge regression model as per the Stan manual, and take samples with the ridge parameter set to three different values —0.5, 15, and 50. The code to perform this analysis is in Appendix B.7.

Visualisation

As mentioned in Section 3.3.3, we need to give special consideration to the visualisation of a fitted model which is defined by a sample from the posterior distribution of its parameters. In R, this means we need to construct a wrapper object with its own `predict` method, so that it behaves like other standard model objects. For this example, we follow the style of an `lm` object, providing posterior sample medians instead of expected values, and 2.5% and 97.5% posterior sample percentiles instead of a 95% confidence interval.

Figure 6.18 shows a section through one of the penalised regression models, with the ridge parameter set to 0.5 (i.e., not shrinking the parameters very much). The posterior distribution of the expected value of `lpsa` is summarised by sample medians and 2.5% and 97.5% percentiles. Figure 6.19 shows a similar section to Figure 6.18 with two more models with the ridge parameter set to 15 and 50 respectively. We can see that increasing the ridge parameter (changing our prior on the other parameters) reduces the magnitude of the modelled effect from the posterior distribution, while also reducing the variance of the posterior distribution.

```
## Create object containing posterior parameter estimates. The 'stanpred' class
## allows us to create our own predict method, and 'custompred' lets ceplot
## know that it can look for confidence intervals in the style of predict.lm.
stanobj <- structure(beta.postsample, class = c("stanpred", "custompred"))

predict.stanpred <- function (object, newdata, interval = "none", ...){
  ## Create design matrix. 'f' is a formula object assumed to be in the global
  ## environment here.
  newx <- model.matrix(f, data = newdata)
  ## Make posterior samples for the response.
  y <- newx %*% t(object)
  ## Return fits like predict.lm, replacing means with medians and confidence
  ## intervals with sample quantiles.
  if (identical(interval, "none")){
    out <- apply(y, 1, median)
  } else if (identical(interval, "confidence")){
  out <- data.frame(
    fit = apply(y, 1, median),
    lwr = apply(y, 1, quantile, 0.025),
    upr = apply(y, 1, quantile, 0.975)
  }
  out
}
```

Code Snippet 6.5: Custom predict method for MCMC samples from posterior of a Bayesian linear model. This is a simplified version of the predict method used for the prostate data, which can be found in Appendix B.7.

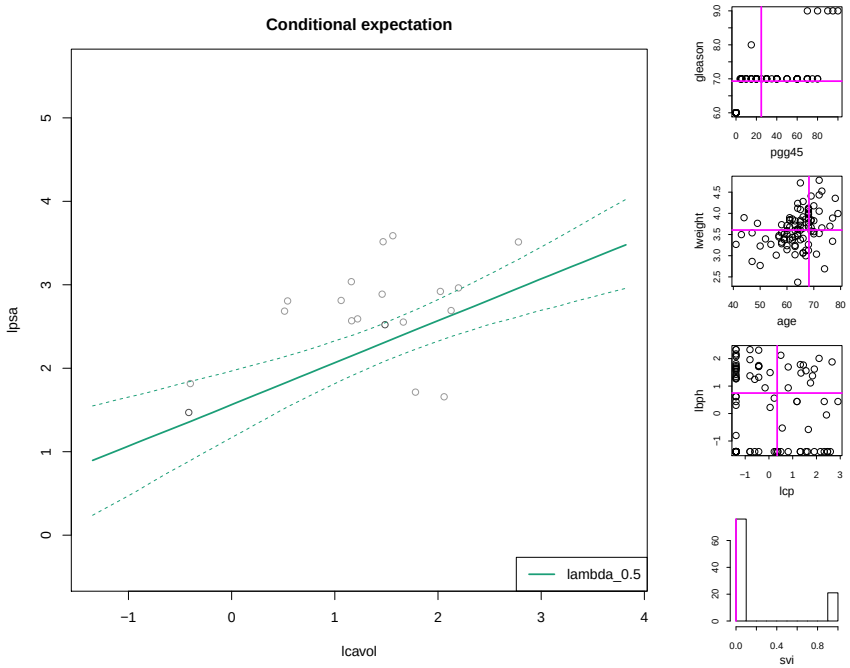


Figure 6.18: Prostate data. Section showing Bayesian ridge regression model with shrinkage parameter set to 0.5. The solid line is the posterior sample median on the section, and the broken lines are the 2.5% and 97.5% posterior sample percentiles. From Code Snippet 6.6.

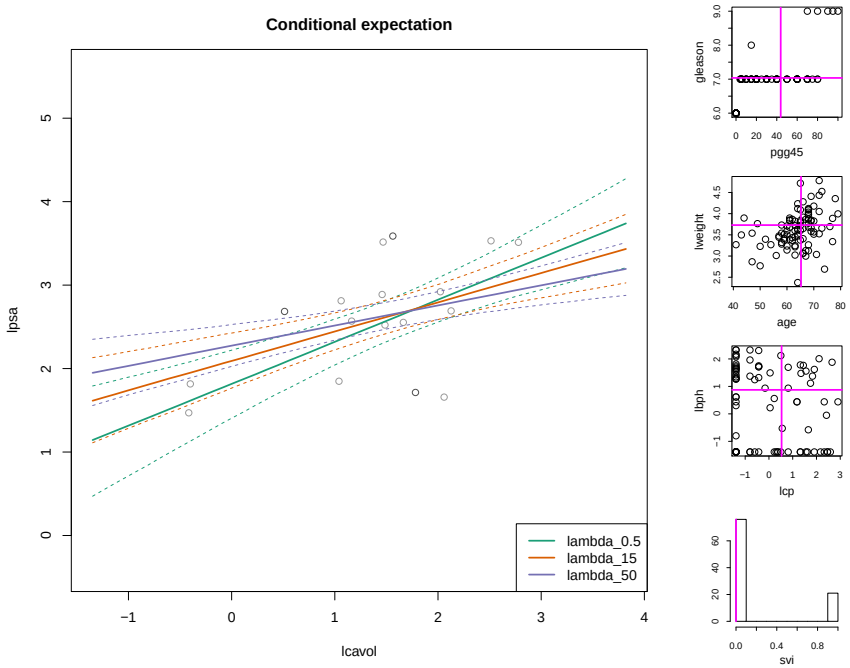


Figure 6.19: Prostate data. Section showing Bayesian ridge regression models with shrinkage parameters 0.5, 15 and 50. Larger shrinkage parameters flatten the slope and reduce the variance of the posterior distribution. The solid lines are posterior sample medians on the section, and the broken lines are the 2.5% and 97.5% posterior sample percentiles. From Code Snippet 6.6.

```
library("condvis")
load("prostate-workspace.rda") # from script in Appendix B.7

ceplot(data = prostate, model = models[1], response = "lpsa", sectionvars =
  "lcavol", conf = TRUE, threshold = 2)

ceplot(data = prostate, model = models, response = "lpsa", sectionvars =
  "lcavol", conf = TRUE, threshold = 2)
```

Code Snippet 6.6: Prostate data. Related visualisations in Figures 6.18 and 6.19.

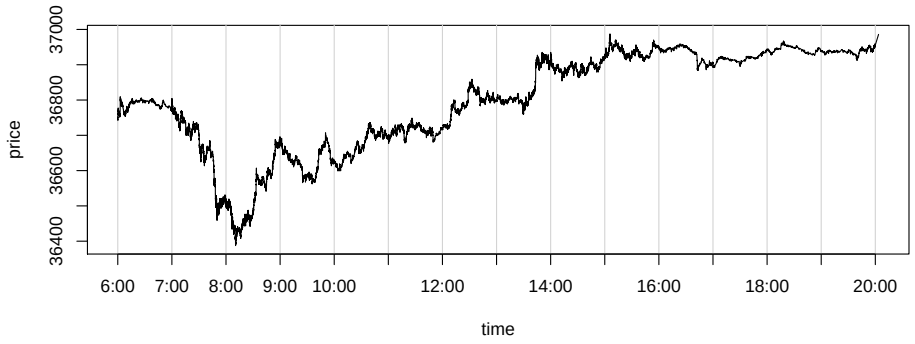


Figure 6.20: Traded prices for an asset across one day at one second intervals.

6.7 Trading strategy data

This example presents an application of conditional visualisation which is slightly different to the previous examples. The data we are considering represent the output and inputs to some complex computer simulation, and so by using conditional visualisation in that data space, we can explore the effect of the simulation parameters on the simulation outcomes.

We address a situation where a statistical model is used to explore a computationally expensive simulation which is given by a deterministic algorithm (see, for example, the discussion by Chen et al. (2016a)). The basic problem is that of understanding (with a view to optimising) a function which is computationally expensive to calculate.

For illustration, we consider the assessment of a financial trading strategy based on a simulation of that strategy on a single day of one-second quote and price data. For a quick view of some of the underlying raw data, Figure 6.20 shows the full time series of traded price data for one day.

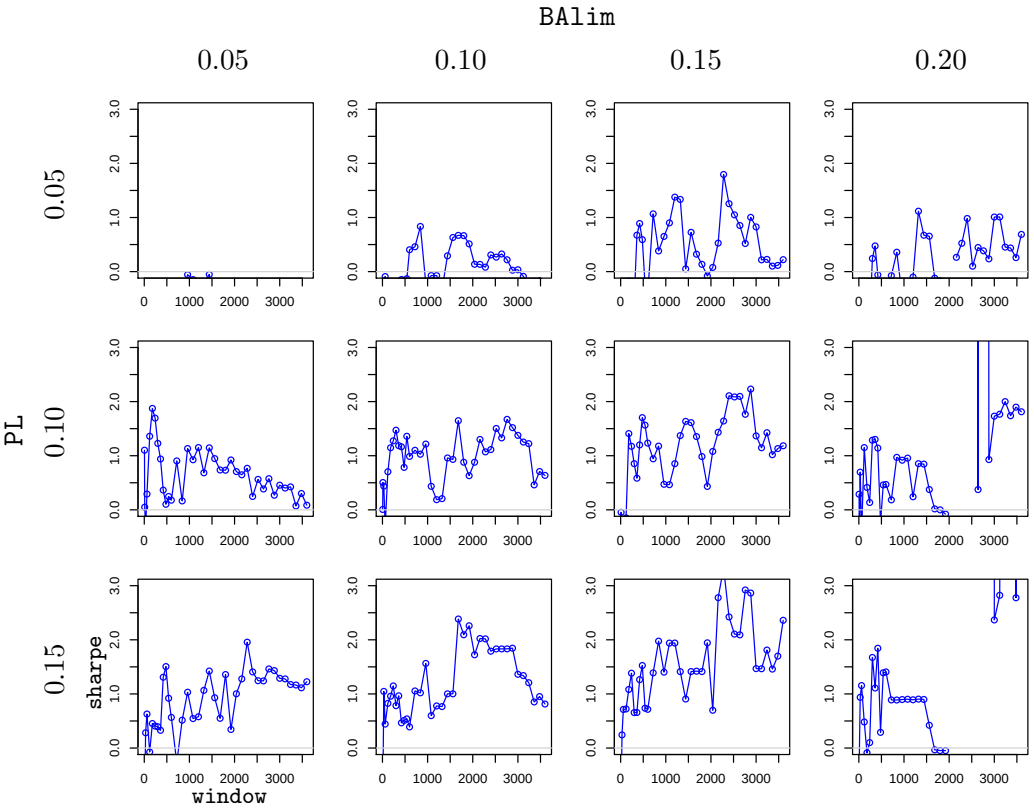


Figure 6.21: Trellis plot of `sharpe` versus `window`, conditioned on `PL` and `BAlim`. Circles are simulation results, lines are for illustration. The vertical axis is the same in each plot. Some negative Sharpe ratios and large positive Sharpe ratios are not visible.

Data

The data for this example are the result of a simulation which is described in Appendix A, where the reader may also find a glossary of financial terms. The response is Sharpe (1994) ratio (`sharpe`) which is a measure of return controlled for variability of return. There are 3 predictors, the input relating to opening trades (`BALim`), the time window over which to calculate a moving average of BA (`window`), and the input relating to closing trades (`PL`). We run the simulation on a three-dimensional grid of these input values to create the response. For more information on the predictors and the simulation, refer to Appendix A. There are 456 observations, 7 of which have missing responses. We simply ignore the missing responses, which represent situations where simulated trades were not closed.

Figure 6.21 shows a trellis plot of `sharpe` versus `window`, conditional on `BALim` and `PL`. These positive Sharpe ratios suggest that the strategy has some 'edge' on this trading day, but not very much. The Sharpe ratios do not look particularly stable across any of the inputs, but there is some promise there, for example, for `window` between 1500 and 2500, with `PL = 0.15` and `BAlim = 0.10`.

Model

Gaussian processes (Williams and Rasmussen, 1995) are a popular way to model the output of computer simulations. These models are based on the assumption that nearby observations convey information about each other. This is a reasonable assumption in modelling the output of a trading strategy simulation, in that small changes in the inputs most often result in small changes of the output.

We fit a Gaussian process to the simulation output data using the `gausspr` function from kernlab (Karatzoglou et al., 2004, 2016). The code for training this model is in Appendix B.8.2.

Visualisation

We now have a data set consisting of observed simulation outcomes in the input space of the trading strategy, and a fitted model which can be used to predict simulation outcomes in other parts of the strategy input space, at a much lower computational cost than running the simulation again – fractions of a second for the model, on the order of minutes for the simulation. The model not only allows us to make predictions, but by using conditional visualisation, we can also try to directly interpret the effects of the parameters on the simulation outcome.

Figure 6.22 shows a section along the `window` predictor taken between observed data points (grid points with evaluations from the simulation (x_S = {`window`}, $\boldsymbol{x}_C$ = {`BAlim`, `PL`}). The Gaussian process model here is telling us that `sharpe` may be increasing with increasing `window`, but the curvature is complicated, showing two peaks across the range of `window`. There are certainly no quick and easy conclusions to draw from this kind of section visualisation, but the observed data nearby in the predictor space seem to support the fitted model.

A three-dimensional section allows us to consider two-way interactions. Figure 6.23 shows a section along `BAlim` and `PL` ($\boldsymbol{x}_S$ = {`BAlim`, `PL`}, x_C = {`window`}). The fitted model on this section suggests very little interaction between `BAlim` and `PL`, and the nearby observed data support the model quite well in this case (in contrast with the power plant example in Figure 6.5).

```
library("condvis")
library("kernlab")
load("trading-workspace.rda") # from script in Appendix B.8.2

ceplot(data = dat, model = list(gp = gp_wrapper), response = "sharpe",
  sectionvars = "window", threshold = 0.8)

ceplot(data = dat, model = gp_wrapper, response = "sharpe", sectionvars =
  c("BAlim", "PL"), view3d = TRUE, threshold = 0.3)
```

Code Snippet 6.7: Trading strategy data. Related visualisations in Figures 6.22 and 6.23.

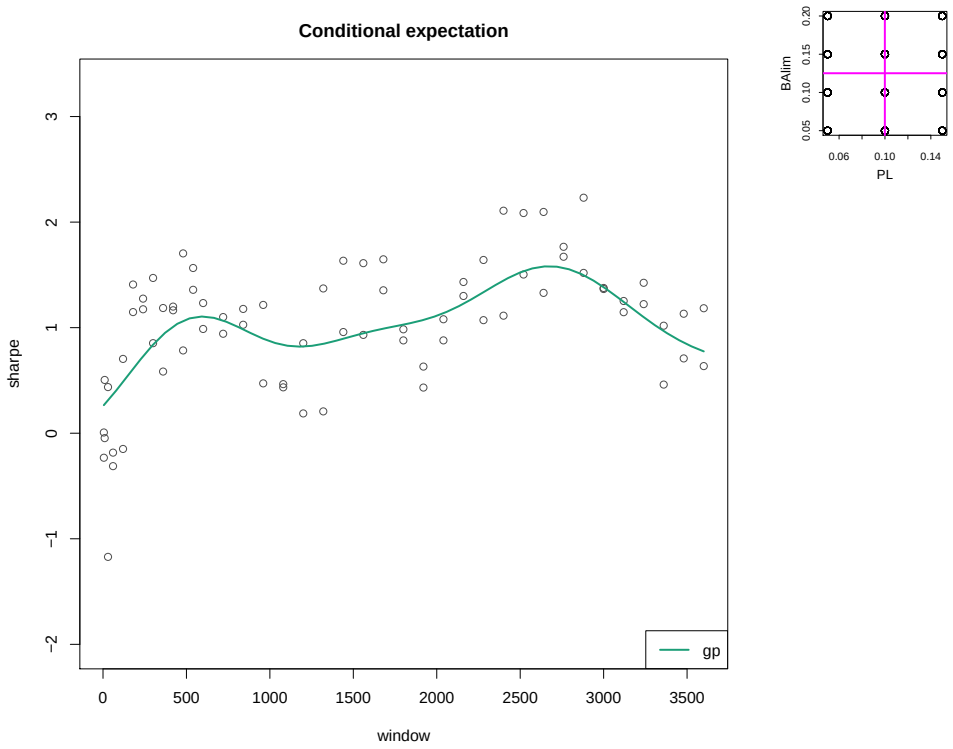


Figure 6.22: Snapshot from Code Snippet 6.7. Conditional expectation plot showing a section through a Gaussian process fit to the trading strategy data. We can interpret the modeled conditional effect of `window` quite simply on a two-dimensional section, although it is not telling us much useful here! The section shown here is taken in between two actual simulation runs, as shown by the condition selector plot on the right.

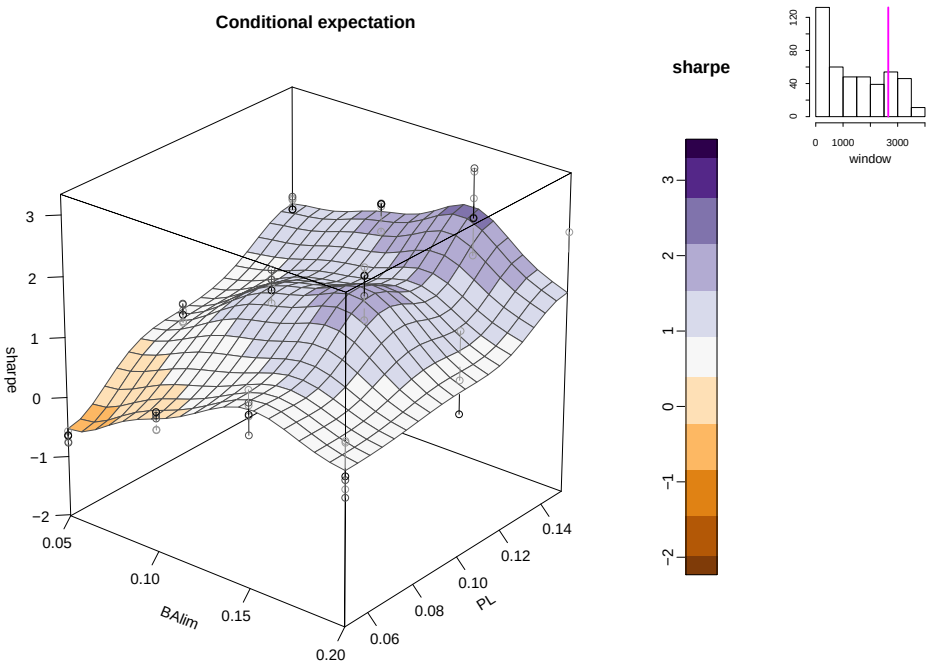


Figure 6.23: Snapshot from Code Snippet 6.7. Conditional expectation plot showing a three-dimensional section through a Gaussian process fit to the trading strategy data. A three-dimensional section allows us to investigate the modeled interaction of `BAlim` and `PL`, if any.

Chapter 7

Visualising model ensembles

7.1 Introduction

Model ensembles are of great importance in modern statistical learning (see, for example, the discussion by Rokach (2010) or Dietterich (2000)). From gradient boosted models to Bayesian model averaging, ensembling can provide a practical approach to variable selection, reduced bias, and improved out-of-sample predictions.

The preceding chapters focus on producing conditional versions of response versus predictor plots. This chapter looks at more abstract visualisations involving residuals from an ensemble of different models.

Chapter goal

This chapter discusses the use of conditional visualisation on ensembles of fitted models. The goal of ensemble visualisation is to discover the basic behaviour of a model ensemble: Are the models telling us the same thing? Are some observations fit poorly by all models, and hence perhaps difficult to fit? The message of the chapter is that conditional visualisation can be used to reveal detail anywhere that individual observations are the input to the visualisation.

Chapter outline

Section 7.2 discusses a small selection of research articles that address visualisation for model ensembles. Section 7.3 discusses the direct visualisation of a model ensemble in the data space. Section 7.4 discusses the application of multivariate visualisation to data derived from a model ensemble. Section 7.5 concludes the chapter with a summary.

7.2 Literature

Unwin et al. (2003) apply multivariate visualisation – specifically, parallel coordinates plots (Inselberg and Dimsdale, 1990) – to data arising from model ensembles.

The data take the form of parameter estimates or residuals.

A ridge trace plot (Hoerl and Kennard, 1970) visualises the parameter estimates from an ensemble of models resulting from fitting penalised linear regressions across a range of values for the penalty parameter. Friendly (2013) applies multivariate visualisation to the parameter estimates from this same model ensemble, using both scatterplot matrices and dimension reduction by principal components, as well as visualising the covariance of parameter estimates.

Wickham et al. (2015) provide a broad discussion of model visualisation and point out the advantage of considering model ensembles over single models. They refer to the idea of visualising a model in data space, which is the approach taken in the other chapters of this thesis. It is also feasible to visualise models in parameter space, or residual space, as mentioned in Section 7.4. In this thesis, we make plots conditional on regions of data space, so we only consider model ensemble visualisations that involve plotting values relating to individual cases (e.g., fitted values or residuals) rather than parameter values.

7.3 Direct visualisation of a model ensemble in data space

If we are only interested in the effect of a single predictor, we can visualise a model ensemble directly in data space as described in Chapters 3, 4 and 5. Each model is represented as a curve, and each curve can be made distinct using colour, line type, or line thickness. See Figure 7.1 for an example of visualising a model ensemble directly in data space; three models trained on the power plant data, as introduced in Section 6.2.

Direct visualisation of model ensembles is far more difficult in the case of two section predictors ($|S| = 2$), where we are actually visualising a two-dimensional projection of the three-dimensional section, and one regression surface can easily obscure another. Consider the colour map and perspective mesh in Figure 3.3, and note that a second model would necessarily be 'in front of' or 'behind' the first model visualised in these cases. One alternative is to arrange the section visualisations for each model from an ensemble in a grid as with trellis (see Section 2.2), though this has not been implemented in condvis at the time of writing.

7.4 Multivariate visualisation of residuals

A typical approach to visualisation for model ensembles is to consider meta-data relating to the models, for example, a data set containing residuals where rows are observations with a column for each model in the ensemble.

Friendly (2013) examines the parameter estimates from penalised regressions (columns are parameter estimates, rows are models given by choice of shrinkage

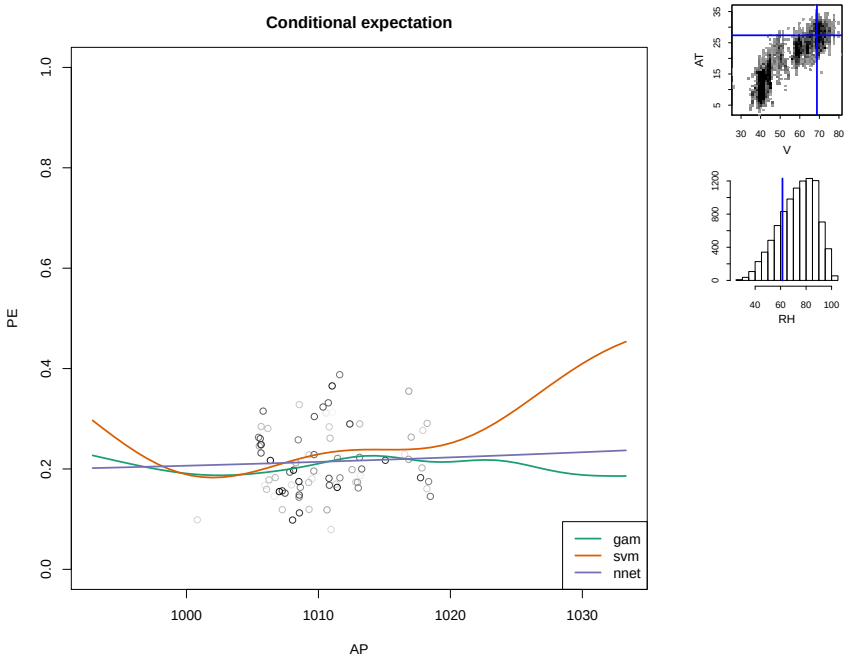


Figure 7.1: Direct visualisation of a model ensemble in data space. It is relatively easy to visualise several models on a two-dimensional section, using colour to identify the models.

hyperparameter), and employs dimension reduction in order to produce a smaller number of scatterplots of parameter estimates than the full scatterplot matrix. This is useful in cases where a model has more parameters than are practical for producing a full scatterplot matrix of parameter estimates (say ten or more parameters, for the sake of argument).

One distinct advantage to working with residuals is that they are more comparable across different models than parameter estimates. For example, we can easily compare the residuals from a linear model to the residuals from a tree model, but the parameters that make up the respective models cannot be compared directly. For the purpose of this discussion, we use the residuals from the regression ensemble fit to the power plant data (Section 6.2) —an additive spline model, a radial kernel support vector machine, and a neural network predicting the power output of a power plant— and the logarithmic loss contributions from the classifier ensemble trained on the blog comments data (Section 6.5) —five gradient boosted tree models which differ mainly in the training and validation data supplied to them, predicting the probability of there being future comments on a blog.

7.4.1 Conditional visualisation of scatterplot matrix of residuals

Scatterplot matrices of residuals plot a point for each observation, and so can be made approximately conditional on a part of predictor space in the same way as the conditional plots in Section 3, using the concept of similarity weight to colour

observations.

The conditional visualisations presented in this section are not implemented in condvis at this time, but they could certainly be incorporated in the same framework as the interactive conditional visualisations presented in Chapters 4 and 5.

Power plant data

Figure 7.2 shows a scatterplot matrix of the residuals from three different models in a regression setting (power plant data, see Section 6.2). The fact that the points are all tightly concentrated around a line of slope 1 shows that the three models fit the data in quite a similar way. Even the most poorly fit observations are all showing a similar residual for each model (see the long sparse tail of large negative residuals in the lower-left of each panel in Figure 7.2). It is clear from the scatterplot matrix that ensembling cannot help to improve predictions on these kinds of observations, because the models all seem to be agreeing with each other.

Figure 7.3 shows a conditional version of the same scatterplot matrix, showing only observations near (in predictor space) an arbitrary observation; the first row of the data (highlighted by a red cross). The distribution of data in this plot looks slightly different to the full plot in Figure 7.2, suggesting that there is more model agreement in this part of the predictor space, because the data are more tightly distributed around a straight line of slope 1. We typically expect these kinds of bias; for example, we expect there to be more model agreement in parts of data space with a high density of observations. See Section 6.2 for a further discussion of this concept. The code for this example is in Appendix B.9.2.

Blog comments data

Residuals are only one example of a measure of contribution to a loss function. We can also consider more general concepts of residuals. Figure 7.6 shows a scatterplot matrix of the logarithmic loss contributions of training observations for five boosted tree classifiers (blog comments data, see Section 6.5). Logarithmic loss contribution values near zero denote good predictions, with large negative values meaning over-confident, incorrect predictions. In this situation, it seems there is more model disagreement than in the previous example, as can be seen by the increased dispersion of the points around a line of slope 1. Of course, we cannot take a visual comparison of plots of residuals with plots of logarithmic loss contributions too seriously, as they are measuring different things. All the same, models with lower correlations between their predictions bode well for ensembling the models in order to achieve more robust predictions.

Figure 7.6 displays over $50,000$ observations, and so can only reveal some overall patterns in the data. Conditional visualisation allows us to reveal extra detail in this situation. Figure 7.7 shows the same scatterplot matrix, but showing only observations near (in predictor space) one of the worst fit cases for all models (high-

lighted by a red cross, with worst meaning the maximum contribution to logarithmic loss averaged across the models). A reader familiar with the logarithmic loss objective function will not be surprised by this picture, as this loss function punishes over-confident, incorrect predictions the most. We see that many of the observations nearest this poorly fit observation score quite well on logarithmic loss, which is likely part of the reason for this over-confident, incorrect prediction on the training data. The code for this example is in Appendix B.9.3.

7.4.2 Conditional visualisation of parallel coordinates plots of residuals

Parallel coordinates plots (Inselberg and Dimsdale, 1990) provide an alternative way to visualise the same data that a scatterplot matrix can display, and are advocated by Unwin et al. (2003) for visualising the residuals from a model ensemble. Figures 7.4 and 7.5 show the parallel coordinates versions of Figures 7.2 and 7.3 respectively. Likewise, Figures 7.8 and 7.9 show the parallel coordinates versions of Figures 7.6 and 7.7, respectively. These parallel coordinates plots demonstrate one of the great strengths of conditional visualisation —to reveal detail by plotting less data— because a parallel coordinates plot of a full dataset can be overcrowded with a lot of overplotting (see Figures 7.4 and 7.8), making it very difficult to see patterns. By plotting a subset of points and colouring them according to their proximity to a point in predictor space, the parallel coordinates plots become easier to read (see Figures 7.5 and 7.9).

7.5 Chapter summary

This chapter briefly discussed the research area of visualising model ensembles. The conditional visualisation of models as described in the preceding chapters can be applied directly to model ensembles, as long as the fitted model is represented as a curve on a two-dimensional section. Conditional visualisation can also be used to reveal detail in different kinds of plots involving individual observations, for example, a scatterplot matrix or parallel coordinates plot of residuals from a model ensemble. Conditional visualisation is particularly useful for parallel coordinates plots where overplotting is a common problem.

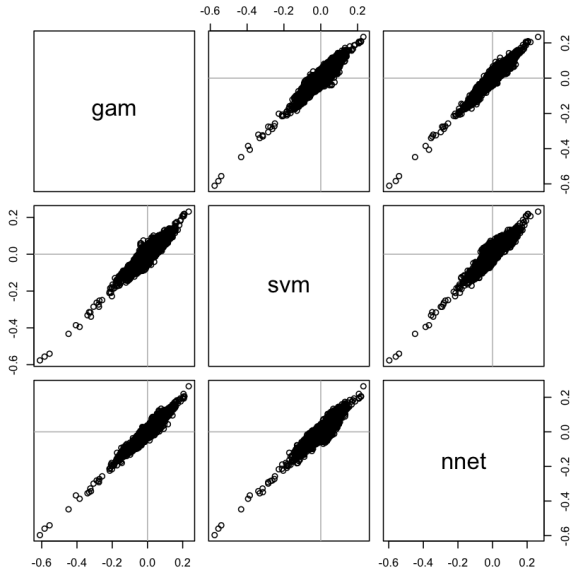


Figure 7.2: Scatterplot matrix of residuals from a model ensemble; an additive spline model, a radial kernel support vector machine, and a neural network, trained on the power plant data from Section 6.2. The horizontal and vertical lines show zero residuals.

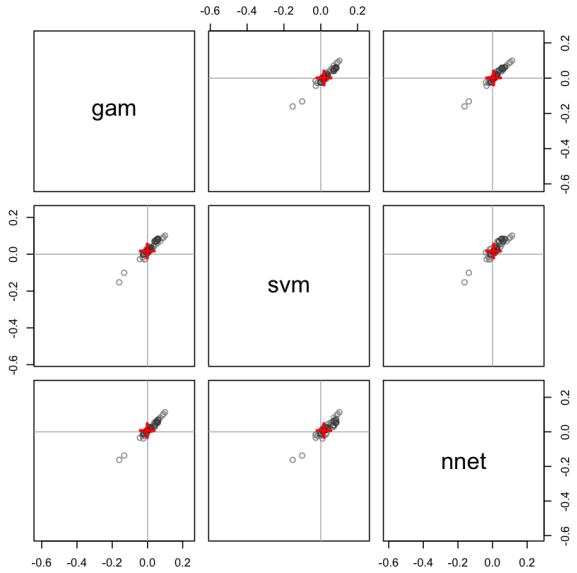


Figure 7.3: Conditional scatterplot matrix of residuals from a model ensemble trained on the power plant data. The same scatterplot matrix as in Figure 7.2 but showing only observations near (in predictor space) the first observation (the red cross) in the dataset. The horizontal and vertical lines show zero residuals.

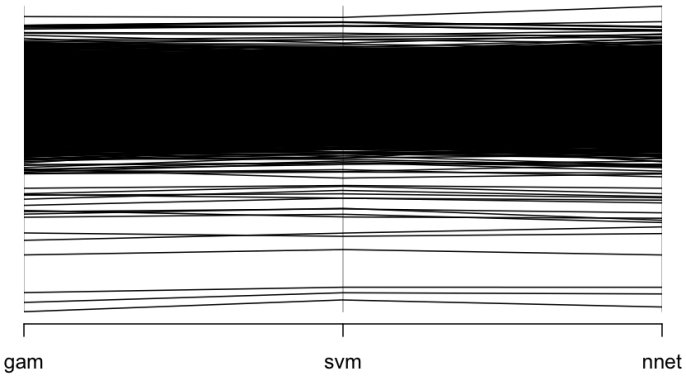


Figure 7.4: Parallel coordinates plot of residuals from a model ensemble; an additive spline model, a radial kernel support vector machine, and a neural network, trained on the power plant data from Section 6.2.

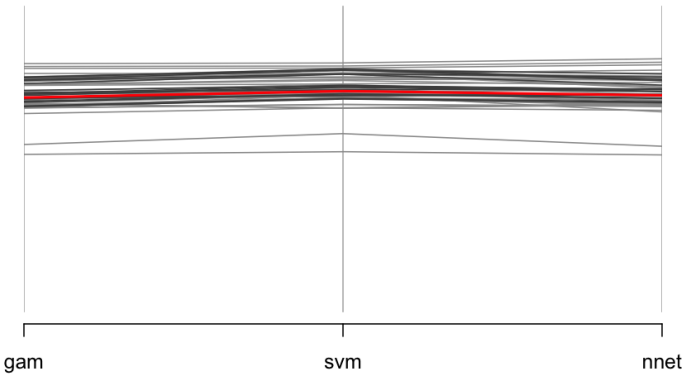


Figure 7.5: Conditional parallel coordinates plot of residuals from a model ensemble trained on the power plant data. The same parallel coordinates plot as in Figure 7.4 but showing only observations near (in predictor space) the first observation (in red) in the dataset.

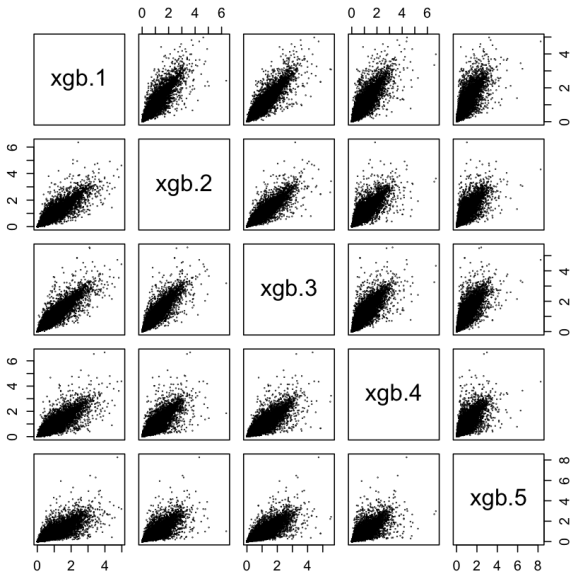


Figure 7.6: Scatterplot matrix of logloss contributions from a model ensemble; five gradient boosted tree models trained on the blog data from Section 6.5.

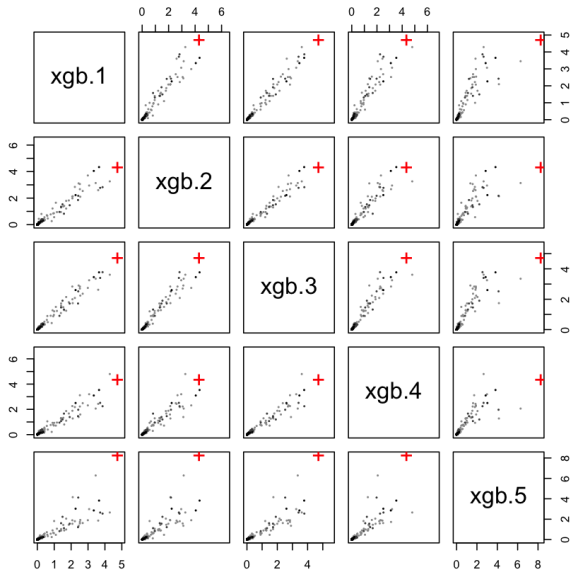


Figure 7.7: Conditional scatterplot matrix of logloss contributions from a model ensemble trained on the blog data. The same scatterplot matrix as in Figure 7.6 but showing only observations near (in predictor space) one of the worst fit observations (the red cross).

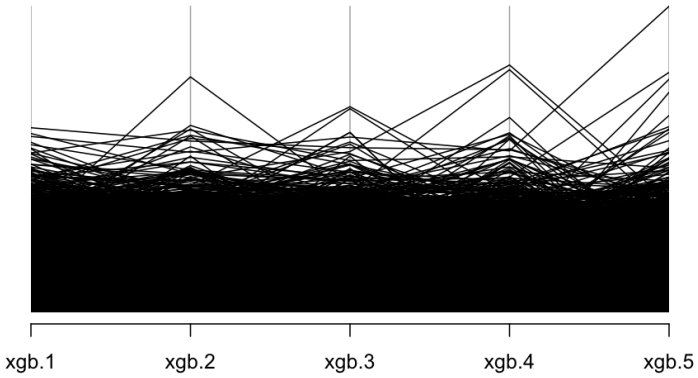


Figure 7.8: Parallel coordinates plot of logloss contributions from a model ensemble; five gradient boosted tree models trained on the blog data from Section 6.5.

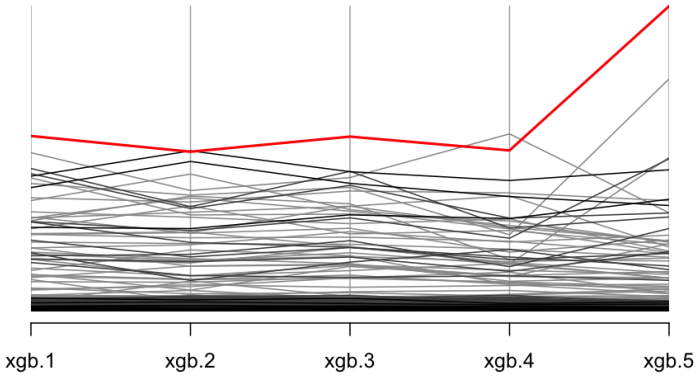


Figure 7.9: Conditional parallel coordinates plot of logloss contributions from a model ensemble trained on the blog data. The same parallel coordinates plot as in Figure 7.8 but showing only observations near (in predictor space) one of the worst fit observations (in red). This observation scores worst on model `xgb.5`, because it was in the validation set for that model.

Chapter 8

Software for interactive graphics

8.1 Introduction

This chapter discusses some of the software options available for creating interactive statistical graphics, and discusses the choice of R for the implementation of the graphics presented in this work.

Chapter goal

The goal of this chapter is to describe the software implementation of interactive graphics in R, which form part of the work of this thesis. The interactive graphics in this chapter use the same techniques as the interactive graphics in condvis, but are taken out of the context of conditional visualisation for the sake of clarity. Some persisting problems with the interactive graphics implementations are mentioned. Alternative implementations are discussed, and reasons are given for not following them up.

Chapter outline

Section 8.2 outlines the reasons for choosing R as a development platform. Section 8.3 gives an outline of the production of an interactive graphic in R, with Section 8.3.6 describing some persisting problems when producing interactive graphics in R. Section 8.4 discusses some alternative software for interactive graphics; packages which extend the basic graphical and interactive capabilities of R, including the use of Shiny, Java, and OpenGL. Section 8.5 concludes the chapter.

8.2 R statistical software environment

The R (R Core Team, 2015) statistical software environment has been used to develop, test and produce all statistical graphics in this work. The choice of R is primarily based on the fact that it has many open source packages for data analysis,

and so it seems sensible to create visualisations within the same software that analyses are likely to be performed. The Comprehensive R Archive Network (CRAN) repository alone hosts 8393 packages (as of May 13th, 2016) implementing a broad range of computational tasks. R has a base graphics system, which is programmable, and outputs to either the computer screen or a file. Basic plots in R are not interactive. Clicking on a graphics device typically has no effect. In Windows, right-clicking a graphics device gives options to copy, save or print the contents of the device.

Work by Duncan Murdoch saw the introduction of `getGraphicsEvent` and related functions to the grDevices package in R version 2.1.0, released in April 2005. These functions provide a simple way to listen for user input (by mouse or keyboard) to a graphics device. In the decade since, there seem to have been very few published uses of these functions; some examples are the Association Navigator (Buja et al., 2010), an interactive graphical tool for exploring large correlation tables, and the sudoku package (Brahm et al., 2014) which provides a sudoku puzzle solver and generator. The `getGraphicsEvent` suite of functions was chosen to provide interactivity for the graphics in this thesis as it minimised the external software dependencies, allowing development efforts to focus on the graphics themselves.

The condvis package uses `getGraphicsEvent` and related functions for interactive graphics, and this can be seen in all of the interactive examples in this thesis. Software alternatives for interactive graphics which were explored but not used for development are discussed in Section 8.4. The current version of condvis implements a Shiny version of the interactive conditional expectation plot. See markajoc.github.io/condvis/example-fev.html for the Shiny version of the conditional expectation plot of the FEV data in Section 1.2.

8.3 Basic interactive graphics in R

The basic process of producing and using an interactive graphic in R with `getGraphicsEvent` can be summarised in six steps.

1. Open a graphics device enabled for interactivity.
2. Call a plotting function.
3. Define event handling functions: `onMouseDown`, `onMouseUp`, `onMouseMove`, `onKeybd`. These functions handle the user input, and subsequently carry out any necessary plotting functions.
4. Set event handling functions for device: `setGraphicsEventHandlers`.
5. Call listening function `getGraphicsEvent`.
6. Provide user input with mouse or keyboard.

A template for producing interactive graphics in R is provided in Code Snippet 8.1.

```
x11() #1
plot.new() #2

mouseclick <- function (buttons, x, y) {...} #3
mouserelease <- function (buttons, x, y) {...}
mousemove <- function (buttons, x, y) {...}
keybdclick <- function (key) {...}

setGraphicsEventHandlers( #4
  onMouseDown = mouseclick,
  onMouseUp = mouserelease,
  onMouseMove = mousemove,
  onKeybd = keybdclick)

getGraphicsEvent() #5
```

Code Snippet 8.1: Template for producing interactive graphics in R. The numbers correspond to the list in Section 8.3. Note that this is not viable code, rather an abstract template that could be filled in to produce an interactive graphic.

8.3.1 Examples

This section presents three examples of the type of interactive graphics found in the condvis package. It is advisable to close other graphics devices before proceeding with each example.

Example: Interactive histogram

For an introductory demonstration of interactive graphics using `getGraphicsEvent()`, we first consider an interactive histogram; a histogram whose bars we can highlight by clicking with the mouse. The first thing we do is open a graphics device enabled for interactivity and plot a histogram, saving the output to a variable (Code Snippet 8.2). We can also open a suitable device using the internal `opendev` function from condvis. Next, we define an event handling function to define what happens

```
if (identical(version$os, "linux-gnu"))
  x11(type = "Xlib") else x11()
o <- hist(mtcars$mpg)
```

Code Snippet 8.2: Open a graphics device and draw histogram. For Figure 8.1.

when a mouse button is clicked. This function does three things:

1. If a previous click has been recorded, the previously highlighted bar is erased with a white bar.

2. The nearest bar to the mouse click is located.

3. The newly selected bar is highlighted.

```
mouseclick <- function (object){
  index <- NULL
  function (buttons, x, y){
    if (!is.null(index)){
      rect(xleft = object$breaks[index], xright = object$breaks[index + 1],
        ybottom = 0, ytop = object$counts[index], col = "white")
    }
    index <<- which.min(abs(grconvertX(x, "ndc", "user") - object$mids))
    rect(xleft = object$breaks[index], xright = object$breaks[index + 1],
      ybottom = 0, ytop = object$counts[index], col = "gray")
  }
}
```

Code Snippet 8.3: Define an event handler for mouse clicks. For Figure 8.1.

Finally, we set the event handlers on the device and start listening for events. The result is an interactive histogram as in Figure 8.1.

```
setGraphicsEventHandlers(onMouseDown = mouseclick(o))
getGraphicsEvent()
```

Code Snippet 8.4: Set the event handler, and listen for events. For Figure 8.1.

Example: Linked histogram and scatterplot

This example shows the capability to link graphics together, where interaction with one graphic can propagate changes to the other(s) in a master/slave style. We produce an interactive histogram as in the previous example, but now we use the histogram to highlight cases in a separate scatterplot. We produce the two plots as in Code Snippet 8.5. We can then use the code in Code Snippet 8.7 to set the event handler and start listening for events. This results in an interactive histogram that controls the highlighting of cases in another scatterplot (Figure 8.2). We can stop the interactive session by closing the graphics device or pressing the 'Esc' key at the console.

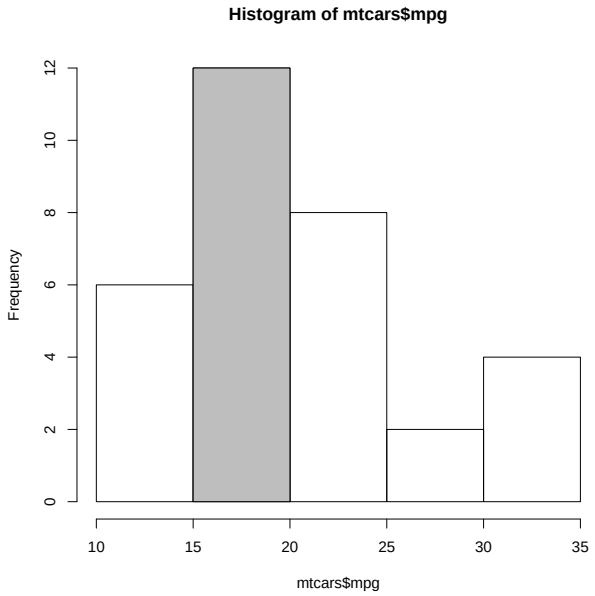


Figure 8.1: Interactive histogram. From Code Snippets 8.2, 8.3, and 8.4.

```
## Open a graphics device for the slave plot

if (identical(version$os, "linux-gnu"))
  x11(type = "Xlib") else x11()

dev1 <- dev.cur()
plot(mtcars$qsec, mtcars$wt)

## Open a suitable device for interactivity. This will be the master plot, we
## interact with it and changes propagate to the slave plot

if (identical(version$os, "linux-gnu"))
  x11(type = "Xlib") else x11()

dev2 <- dev.cur()
o <- hist(mtcars$mpg)
```

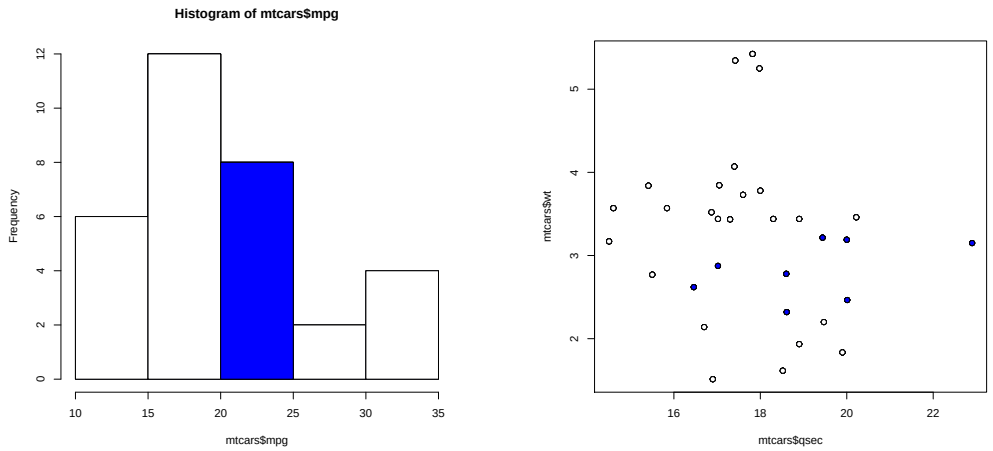
Code Snippet 8.5: Open a device and draw the slave plot. Open a second device and draw the interactive master plot. For Figure 8.2.

```
mouseclick <- function (object){
  index <- NULL
  function (buttons, x, y){
    dev.set(dev2)
    if (!is.null(index)){
      rect(xleft = object$breaks[index], xright = object$breaks[index + 1L],
        ybottom = 0, ytop = object$counts[index], col = "white")
    }
    index <<- which.min(abs(grconvertX(x, "ndc", "user") - object$mids))
    rect(xleft = object$breaks[index], xright = object$breaks[index + 1L],
      ybottom = 0, ytop = object$counts[index], col = "blue")
    dev.set(dev1)
    colour <- c("white", "blue")[(findInterval(mtcars$mpg, o$breaks[index:(
      index + 1L)]) == 1L) + 1L]
    points(mtcars$qsec, mtcars$wt, pch = 21, bg = colour)
  }
}
```

Code Snippet 8.6: Define event handler to take mouse clicks on the histogram, highlight the relevant bar of the histogram, and then highlight the selected data on the scatterplot. For Figure 8.2.

```
setGraphicsEventHandlers(onMouseDown = mouseclick(o))
getGraphicsEvent()
```

Code Snippet 8.7: Set the event handler, and listen for events. For Figure 8.2.



(a) (b)

Figure 8.2: Interactive histogram (a) that controls the highlighting in scatterplot (b). From Code Snippets 8.5, 8.6 and 8.7.

Example: Two-way linked graphics

This example shows two-way linked graphics, where changes in one interactive graphic can be propagated to arbitrarily many other graphics, which themselves can be interactive graphics with the same capability.

We produce two scatterplots in separate graphics devices, and propagate clicks from one to the other and vice versa. We start by defining a plot function, which returns an object to retain information for updating the plot. We then define an update method which accepts plotting coordinates and draws a point (see Code Snippet 8.8).

We then define the event handling functions in Code Snippet 8.9. The mouse click handler sets the appropriate device coordinate system, records the pointer coordinates, and updates both graphics devices. Note that we have actually defined a function that returns a mouse click event handling function. This time, we add a keyboard event handling function to terminate the interactive session by pressing the 'q' key.

Finally, in Code Snippet 8.10, we open two graphics devices and set graphics handlers on *each* device. We start the interactive session as per usual with a call to `getGraphicsEvent`. On some platforms, the first click on an inactive device may only switch focus to that device without drawing a point.

```
## Function to create plot and return object storing
## device number, ‘usr‘ and ‘mar‘.

myplot <- function (pch){
  plot(0, 0, col = NULL, pch = pch)
  structure(list(device = dev.cur(), usr = par()$usr,
    mar = par()$mar, pch = pch), class = "myplot")
}

## Update method for plot object: switches to the
## correct device, sets ‘usr‘ and ‘mar‘, then plots
## a point given by ‘x‘ and ‘y‘

update.myplot <- function (object, x, y, col){
  dev.set(object$device)
  par(usr = object$usr)
  par(mar = object$mar)
  points(x, y, pch = object$pch, col = col)
}
```

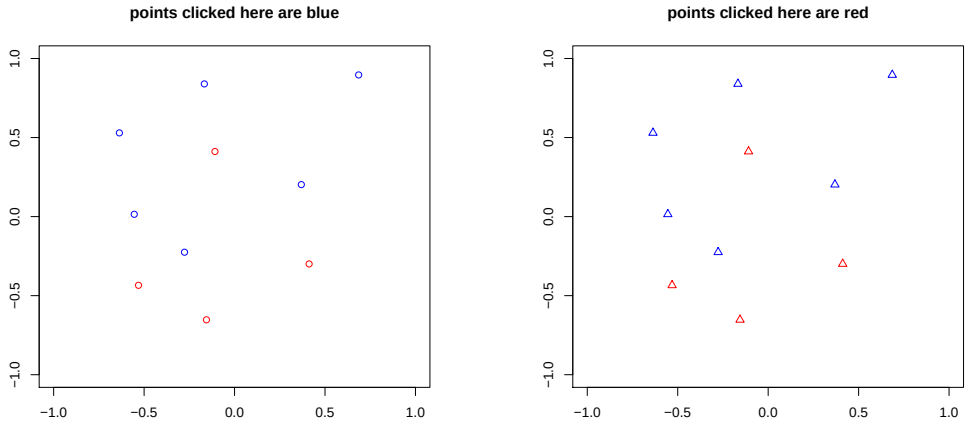
Code Snippet 8.8: Function to create a scatterplot, returning an object with information relevant to updating the plot. Also, the update method for the plot object. For Figure 8.3.

```
## Function to create an event handler function to react to mouse clicks, which
## takes the xy coordinates of the mouse click, interprets them to the correct
## user coordinates, and draws a point to both devices via ‘update‘.

mouseclick <- function (object, col){
  function (buttons, x, y){
    dev.set(object$device)
    par(usr = object$usr)
    par(mar = object$mar)
    xnew <- grconvertX(x, "ndc", "user")
    ynew <- grconvertY(y, "ndc", "user")
    update(o1, x = xnew, y = ynew, col = col)
    update(o2, x = xnew, y = ynew, col = col)
  }
}

## Function to create an event handler function to react to keystrokes; just
## providing a way to end the interactive session by pressing ‘q’.

keystroke <- function (){
  function (key){
    if (identical(key, "q")){
      cat("\nInteractive session ended.\n")
      return(invisible(1))
    }
  }
}
```

Code Snippet 8.9: Event handling functions for mouse clicks and keyboard strokes. For Figure 8.3.

```
## Open a graphics device enabled for interactivity. Create plot, save
## resulting object and set up the event listener.

if (identical(version$os, "linux-gnu"))
  x11(type = "Xlib") else x11()
o1 <- myplot(1)
title(main = "points clicked here are blue")
setGraphicsEventHandlers(onMouseDown = mouseclick(o1, "blue"), onKeybd =
  keystroke())

## Open a second suitable graphics device. Create another plot, save resulting
## object, and set up another event listener.

if (identical(version$os, "linux-gnu"))
  x11(type = "Xlib") else x11()
o2 <- myplot(2)
title(main = "points clicked here are red")
setGraphicsEventHandlers(onMouseDown = mouseclick(o2, "red"), onKeybd =
  keystroke())

## Listen for mouse clicks.

getGraphicsEvent()

## Click around the plot area on EITHER device, the point is plotted on each
## device at the same coordinates with colour to show where the click
## originated.
```

Code Snippet 8.10: Create two plots, setting up an event handler for each device. Listen for events. For Figure 8.3.



(a) (b)

Figure 8.3: Points clicked in scatterplot (a) are also drawn in (b) and vice versa. From Code Snippets 8.8, 8.9, and 8.10.

8.3.2 Event handling functions

The `getGraphicsEvent` function takes four types of event handling functions as input. Each one is a function written by the user, taking specific named inputs:

- `onMouseDown` takes three inputs: `buttons`, `x` and `y`.
- `onMouseUp` takes three inputs: `buttons`, `x` and `y`.
- `onMouseMove` takes three inputs: `buttons`, `x` and `y`.
- `onKeybd` takes one input: `key`.

On X11 devices, mouse motion and button events are only captured correctly on relatively recent versions of R (see PR 16700). On the Mac OS, there are some keystrokes which are given different names, such as the return key.

The interactive session is terminated if one of the event handlers returns a non-`NULL` value, which can happen unintentionally. As a consequence, it is useful to end each function definition with something like `points(NULL)`, to ensure that the interactive session does not terminate prematurely.

8.3.3 Coordinate systems

There are two important coordinate systems for interactive graphics in R.

- `"user"` – user coordinates
- `"ndc"` – normalized device coordinates

The functions `grconvertX()` and `grconvertY()` are available from grDevices to convert between these different coordinate systems. These function calls depend on the current `par` settings, which is important when updating plots.

8.3.4 Updating plots

Graphics code in R is optimised for drawing high quality vector graphics, and so is not particularly fast in drawing to the screen, and does not provide ways to efficiently update parts of an existing graphic. Updating and redrawing are integral parts of any interactive graphics system. We are faced with two options when updating a graphic in R:

- We can update the plot using a full redraw. This is the simplest option in maintaining coordinate systems and keeping a clean display, but it can be rather slow. Or,
- we can do a partial erase and redraw. This is more difficult, but can produce visibly smoother results. This involves figuring out what parts of the graphic need to be erased, blanking them with plotting commands, and then redrawing the new elements.

In condvis, we opt for the latter option whenever possible, as it produces the smoothest appearance when updating graphics. In a few cases, the full redraw option is used because the required information for selective updating is buried in the internal code of another package.

Neither of these options particularly overcomes the problem that R sends vector drawing commands to the graphics device, all of which are refreshed if the device is resized. This has many benefits, such as resizing axes or maintaining aspect ratios of text, but it makes for a lot of drawing if a device is resized in the middle of an interactive session. See Section 8.3.6 for further discussion on this topic.

8.3.5 Linked graphics

Separate graphics devices

When this work started, closing an inactive graphics device while listening for events on another device with `getGraphicsEvent` caused R to crash. This bug has since been fixed (PR 16438). For this reason, the linked interactive graphics for condvis were first developed to be all located on one graphics device. Another difficulty in working with separate devices is the way in which graphics devices are identified in R. Devices are numbered as they are opened, and the number decreases by 1 if a previously opened device is closed. This makes it difficult to consistently and robustly identify the correct device with which we want to link.

Splitting one graphics device

We can split an R graphics device using `par()$mfrow` or `layout()`, but then we plot to these sub-panels in order, and cannot access them arbitrarily. The `split.screen` function allows more flexible splitting, and subsequent selection of specific screens.

However, when switching between screens defined by `split.screen`, the most important `"user"` coordinate system is not guaranteed to be preserved. Indeed, the documentation specifically states "The behavior associated with returning to a screen to add to an existing plot is unpredictable and may result in problems that are not readily visible." This is dealt with in condvis by saving the current `par("usr")` and `par("mar")` values in each plot object.

8.3.6 Persistent problems

This section discusses some problems with the software implementation of conditional visualisation found in the condvis package. The problems discussed here are those with no apparent solution currently available, and so the problems either currently persist or have a sub-optimal workaround.

Resizing graphics device

R draws vector graphics to the graphics device, and *remembers* everything it has drawn to the device. As a result, when a window is resized, everything is redrawn. If we have been interacting with a device for a long time, the redraw can involve a very large number of drawing operations. It would be preferable to have control of what is redrawn upon resizing operations, and allow the device to *forget* certain graphics objects which are no longer needed. In many cases, it would actually be preferable to have a fixed image as a background, and only plot the interactive elements on top of that background; a layered approach. It is possible to partially achieve this using a PNG output as the background, but it is not very practical, especially on resizing the window.

Flickering display

The X11 device interface in R does not support `dev.hold` and `dev.flush`; functions which allow plotting commands to be buffered and then applied to the device in quick succession. As a result, time between erasing and drawing new objects can be large enough so as to be perceptible and often irritating. A reasonable solution is to place the erasing code as close as possible to the code for redrawing. However, if the code for redrawing is causing the delay, there is not much that can be done without `dev.flush`.

Concurrent interactive sessions

The current implementation of `getGraphicsEvent` does not allow for a second call to the function until the previous interactive session has been terminated. As such, once we create an interactive graphic with `getGraphicsEvent`, it is difficult to add further related graphics to the interactive session, and it is impossible to create

new interactive graphics from within an interactive session. Consequently, the interactive sessions in condvis are standalone, and cannot easily be combined or run concurrently.

8.4 Alternative software

Web browsers and Shiny

Shiny is a web application framework for R, developed by RStudio. The shiny package (Chang et al., 2015) allows R users who are not necessarily experienced in web programming to produce interactive web applications to present their analyses, which have been carried out in R. R code describing the application to be made can be stored in two script files – one for loading the data and performing the analysis, and one to control the display and user interface – then the shiny function may be run in a standard R session, producing the HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript (not to be confused with Java) code required to form the web application, which may subsequently be viewed in a web browser. The interactive application then has the ability to send information back to the R session and run further calculations in R. It is now possible to run Shiny code on a web server which runs R, and hence deploy the whole application online, available to users without any R distribution installed. The main reason that Shiny has not been used as the main development platform is that Shiny does not give direct provision for a workflow involving plot interactions which alter the plots themselves, or at least, did not do so when the main development of condvis took place.

Another example of using web programming to produce interactive graphics is the ggvis package (Chang and Wickham, 2015), building on concepts from ggplot2 (Wickham, 2009), ggplot2 being based on a grammar of graphics (Wilkinson et al., 2006),

Java and Processing

Java is a programming language, which has become a global standard for developing and delivering computer applications. Java is well suited to creating high quality visualisations and graphical user interfaces, which can be used across a broad range of operating systems and computer architectures. Processing (https://processing.org/) is a development environment which allows users with little Java experience to create both animated and interactive computer visualisations with relative ease in Java.

We can perform data analysis in R and visualise the analysis using a Java application. An obvious problem arises when we want to interact with the graphic and change the original analysis. This requires a two-way connection between R and Java.

The Rserve (Urbanek, 2013b) package allows R to run as a server, serving requests from external programs such as Java applications. The rJava (Urbanek, 2013a) package provides an Application Programming Interface (API) to Java programs from R. The iplots (Urbanek and Wichtrey, 2013) package produces a small selection of interactive graphics using the rJava package. KLIMT (Urbanek and Unwin, 2002) is a Java program for interactively visualising tree models. CASSAT (Unwin et al., 2003) is a Java program for producing interactive parallel coordinates plots.

The main reason Java has not been used in this project is that it does not allow us to easily take advantage of the vast number of plots already coded in the many R packages available.

OpenGL

The rgl (Adler and Murdoch, 2016) package uses OpenGL (2015) to create interactive 3-D graphics from R, such as rotatable surfaces. The main drawback (as with Java) is that rgl cannot simply make R graphics interactive, it is creating new graphics with the OpenGL engine.

Tcl/Tk

Tcl/Tk (https://tcl.tk/) is a combination of a scripting language and a graphical user interface toolkit, which can be used to make interactive graphics using the tcltk package, distributed with R as standard. Examples of packages which use this platform are loon (Waddell and Oldford, 2015) for interactive, linked histograms, scatterplots, star glyphs and parallel coordinates plots, and RnavGraph (Waddell and Oldford, 2014) for exploring high-dimensional data using graphs as the navigational infrastructure.

8.5 Chapter summary

This chapter described the type of interactive graphics used to implement the conditional visualisation techniques described in Chapters 3 4, and 5. The graphics were developed in R to allow for easy integration with existing code for model fitting and data visualisation. Certain sacrifices are made, but the result is a set of interactive graphical tools that almost exclusively use base R functions. The only non-standard dependency is the XQuartz device for the Mac OS.

Chapter 9

Conclusion and outlook

9.1 Summary

This thesis presents a new approach to conditional visualisation with a focus on statistical models, alongside a software implementation in R in the condvis package. The idea is to visualise one low-dimensional section at a time and display nearby observed data on the section according to a distance measure. By incorporating observed data into the visualisation, it becomes possible to decide whether the observed data support the fitted model, or if the section represents an extrapolation in data space. This approach is implemented in two ways; by manually choosing sections with the aid of data summary graphics, or by preselecting the sections to visualise and then moving through them one by one.

The methods have been demonstrated in applications involving additive spline models, support vector machines, Bayesian regression, random forests, gradient boosted trees, neural networks, and ensembles of models.

The manual interactive approach works quite well for models with less than ten predictors, and is especially useful for querying a fitted model at arbitrary points in the predictor space. The current section can be changed quickly and easily, so predictor interactions can be explored in an intuitive way.

For models with many more predictors, it becomes very difficult to choose suitable sections to visualise by hand, and so it makes sense to automatically choose sections according to some criteria. Examples of such sections would be observations which are fit poorly by the model, or centres from a clustering analysis of the data.

9.2 Further work

The research presented in this thesis represents early steps in the direction of taking a hands-on approach to understanding complicated statistical models in high-dimensional data space. In keeping with this, there are many shortcomings of the approach, potential improvements left to explore, and newly opened avenues of research to go down. This section gives some recommendations for future work which

could be worth investigating.

9.2.1 Dissimilarity function

The similarity weight used to make plots conditional as in Section 3.4 is based on a dissimilarity function. The dissimilarity function used in this thesis is rather simple, and for good reason; it is intended to give a model-agnostic way to choose data points which can sensibly be compared with each other, or compared with a particular slice through a fitted model.

An improvement could be made in the treatment of categorical predictors. This could be as simple as using a different scaling constant for each predictor, giving different weights to mismatches on different predictors. A more flexible coefficient of dissimilarity could also be used (Gower, 1971).

The dissimilarity measure used for continuous predictors in this thesis takes no account of covariance between predictors. It might be desirable in some situations to use Mahalanobis distance to construct a dissimilarity measure.

There may be situations where it would be sensible to take account of the response in formulating a dissimilarity measure. By fitting a model, we could use the gradient of the fitted model as a guide to alter our distance measures to count observations as closer together along directions of small gradient, and count observations as further apart along directions of large fitted model gradient. For another example, we could fit a tree model to the data. We could then assign zero dissimilarity to observations in the same partition, a moderate dissimilarity to observations in adjoining partitions, and larger values elsewhere. Or, we could use any other dissimilarity measure on the observations, treating observations as though they reside at the centroid of their respective partitioned regions from the tree model.

Whatever the future holds for dissimilarity measures in conditional visualisation, it will certainly become necessary at some point to write some fast code implementing a fixed radius near neighbours search algorithm (see, for example, Bentley (1975)). While the brute force approach of calculating all distances works well for data that fit comfortably in RAM, for millions or billions of observations, some little tricks would become useful. For example, the distance can be calculated on a subset of columns, discarding any observations exceeding the chosen fixed radius. This process can be continued in a cascading fashion, reducing the number of observations under consideration at each iteration, until all columns have been checked. The basic setup of the fixed radius near neighbours search lends itself to distributed and parallel computing, which bodes well for addressing the inevitable need to scale the method to much larger data sets.

9.2.2 Cognostics

When continuous conditioning variables are involved, there are infinitely many sections which can be visualised. *Cognostics* (Tukey and Tukey, 1982) refers to the

process of using a computer to screen large numbers of computer graphics, with the goal of selecting a subset of graphics which are of interest for human examination, according to some criteria. This method could be applied to the problem of visualising a useful subset of the infinitely many sections available (see Anand and Talbot (2016) for a related discussion on the choice of shingles for trellis graphics).

In some sense, we can see this as the inverse of the approach using interactive conditional expectation plots as in Chapter 4. There, we choose a section and then investigate it for interesting properties. With cognostics, we hope to find the parts of the predictor space where sections will demonstrate some specific property of the data or fitted model. This would be closely related to the conditional tour, discussed in Chapter 5.

9.2.3 Model comparison

Several models can be visualised on a two-dimensional section without difficulty (see Sections 7.3 and 6.5). This can be achieved by plotting a curve for each model, giving each curve a different colour or line style. As such, visual model comparison is possible on two-dimensional sections. Model comparison is far more difficult for three-dimensional sections. The section visualisation is itself a two-dimensional projection of three-dimensional space, and so the regression surface of one model could easily obscure that of another model, hindering any kind of visual comparison. One simple approach to the problem would be to produce a different section visualisation for each model, and arrange them in a grid as with trellis graphics. Another approach would be to visualise the differences between models, but this would only allow comparison of two models. Section 7 considers a broader view of model comparison, looking at ensembles of models.

9.2.4 Flexible conditioning

The whole approach to conditional visualisation presented in this thesis has been quite strict in its definition by fixing predictor values. This implies sections which are orthogonal to the axes in predictor space, which makes for straightforward interpretation of the conditioning. Of course, we could be far more general in considering conditioning as a restriction to any valid subregion of the predictor space. This would then necessitate an integration of the fitted model over the subregion as in Nason et al. (2004) and Friedman (2001), but might prove useful in cases involving highly structured data.

9.2.5 Special data structures

Time series

The work in this thesis has made no special provisions for the treatment of time variables. In some ways, time variables may be treated just like any other numeric

predictors, but if a time variable was used in the distance function when seeking observations near a section, unexpected results might be obtained. For example, the data are scaled before calculating distances. It may not make much sense to standardise a variable which gives the number of seconds since the epoch.

Using time as a conditioning predictor could very well be an interesting way to approach the visualisation of model of a process which evolves through time. This is getting closer to the goal of the visualisations from Gapminder (gapminder.org), where time-series data is visualised interactively.

Nested predictors

No explicit consideration has been given to nested predictors or hierarchical models in this thesis. The main problem is that a nested predictor structure provides the possibility of producing a much larger number of extrapolations or meaningless predictor vectors than a model without hierarchy of predictors. For example, consider two predictors for `country` and `city`. The city predictor is nested within the country predictor. It would seem to be meaningless to consider a combination of `country = "Germany"` and `city = "Madrid"`, and we do not need distance measures to figure this out!

One possible way to address this structure would be to provide a special kind of condition selector graphic. To choose a section, the user would be required to start by selecting factor levels of the predictors at the top of the nesting (e.g., country), and then only factor levels which match the nesting structure are offered for selection at lower and lower levels of the hierarchy (e.g., cities in that country, boroughs in that city, and so on).

It might be interesting to try displaying observed data on such sections which match the current section at various levels of the hierarchy, so that group effects and individual effects might be given some kind of context.

9.2.6 Path for conditional tour

The conditional tour in Chapter 5 is presented as an automated alternative to interactively choosing sections to visualise as in Chapter 4. This automated approach to choosing sections alleviates certain problems such as high-dimensionality of predictor spaces, but brings its own difficulties in trying to evaluate the 'quality' of a section with one or two criteria, such as proximity to observed data. It would certainly be worthwhile trying to take an interactive graphical approach to creating the path for the conditional tour, allowing the user to tweak an otherwise automated process. This could also make better use of the diagnostic plots described in Section 5.5, which are currently only used for *post hoc* assessment of the path.

9.2.7 Missing data

The visualisations presented in this thesis give no consideration to missing data, and the implementations in condvis remove incomplete observations before producing any graphics. This is acceptable for exploring the basic concepts of conditional visualisation but is not very practical in general. It would be desirable to allow incomplete observations to be included in conditional visualisation techniques, beginning with some of the many existing approaches (see discussion in Chapter 3 of Cook and Swayne (2007)).

9.2.8 Generalised conditional visualisation

Chapter 7 introduced the idea that conditional visualisation, as presented in this thesis, could be abstracted to applying the similarity weight concept to any visualisation with individual observations as input. This is surely worth exploring further, and will hopefully be reflected in further iterations of the condvis package, by separating the code for conditioning a plot (condition selectors, or conditional tour) from the code for visualising a section in data space.

9.2.9 'Big data'

This thesis makes no specific mention of 'big data' (big, in the sense of being too big to handle in RAM). Likewise, the software implementation in condvis has not been designed for such large datasets. At the same time, we have shown most of the techniques necessary for scaling the methodology in this thesis already. These consist of; using subsets of important predictors to alleviate large p (see Section 6.5), making condition selector plots robust to large n by using summary graphics, the fact that a conditional plot is a drill-down graphic involving substantially less than n observations. The fixed-radius near neighbour search which forms a central part of the process for calculating similarity weight is a good candidate for parallel computation. For these reasons, it seems there is good potential for scaling the methodology for conditional visualisation presented in this thesis to much larger datasets.

9.3 Conclusion

This thesis presents a new approach to conditional visualisation for data and fitted models. The new method involves visualising a single section at a time, displaying both fitted models on the section and observed data which are near the section according to a distance measure. Sections may be chosen either by hand with interactive graphics, or programmatically according to an algorithm. The new method allows the interpretation of a broad range of regression and classification models from linear models to 'black-box' models. Finally, by enriching the visualisation with observed data, the user can decide if the observed data support the fitted model and related assumptions throughout the predictor space.

List of Acronyms

API Application Programming Interface

CART Classification and Regression Trees

CERES Combining Conditional Expectations and Residuals

CRAN Comprehensive R Archive Network

CSS Cascading Style Sheets

FEV Forced Expiratory Volume

HTML HyperText Markup Language

ICE Individual Conditional Expectation

MCMC Markov Chain Monte Carlo

PDP Partial Dependence Plot

RAM Random Access Memory

RGB Red, Green, Blue

Appendix A

Simulating trading data

This appendix describes how the data set referred to in Section 6.7 is generated.

A.1 Trading strategy

We describe a simplistic strategy to trade a financial product (see Figure A.1 for an outline). The instructions to open a trade come from a supply/demand indicator described below. The decision to close a trade is governed entirely by profit/loss limits. The strategy only holds one position at a time, and every trade is the same size. When the strategy is holding no position, it has three options: buy, short, or do nothing. When the strategy has an open position it can do one of two things: close the position, or do nothing.

We define a supply/demand indicator as

$$\mathrm{BA} = \log\left(\frac{\text{bid volume}}{\text{ask volume}}\right)$$

The raw input to the strategy is the history of BA up to the current time period (taking time periods to be seconds). The parameters are

- window: the time window in seconds for the calculation of EMA(BA), the exponential moving average of BA.
- BAlim: the critical values of EMA(BA) at which we will open a trade. We just use one value, buying when EMA(BA) exceeds +BAlim and shorting when EMA(BA) is lower than −BAlim, to keep this a one-dimensional input.
- PL: the take-profit and stop-loss targets (in percent). As with BAlim, we specify one number and make the limits symmetric to keep this a one-dimensional input.

A.2 Simulation

We have a three-dimensional input space over which we might optimise this trading strategy. We choose a grid of points in the input space and simulate the strategy's

	Trading strategy
	Given: EMA(BA), BAlim, and PL
1	**if** EMA(BA) $>$ +BAlim
2	buy and hold until take-profit or stop-loss
3	**else if** EMA(BA) $<$ −BAlim
4	short and close at take-profit or stop-loss
5	**else** do nothing

Figure A.1: Outline of trading decision process. EMA(BA) is the exponential moving average of the BA indicator. ±BAlim are the critical values that EMA(BA) must cross to initiate a trade. ±PL are the take-profit and stop-loss in percent.

performance on a sample trading day using some simple assumptions. As output, we calculate a Sharpe ratio for each simulated trading day.

Assumptions

We make the following assumptions in simulating the performance of the trading strategy:

- that we can trade within the same second as the trading decision.
- that we can buy or short the product in small amounts (e.g. volumes around 2 - 5) at the interpolated traded prices at any second during the day. We could be more conservative in this and assume less favourable traded prices. We could also attach a random failure rate to trade execution to model the uncertainty of orders being filled.
- that our take-profit and stop-loss orders are executed at the first interpolated traded price outside these limits. For the price history we have here, this assumption is reasonable, but in general it is not.
- a trading friction of 0.01% of traded value on every trade from opening to closing (including shorts)

Glossary of financial terms

close to close a trade means to sell an asset that you have bought, or to buy back an asset that you have sold short.

edge an apparent statistical advantage making trades more likely to be profitable.

friction losses associated with trading, such as transaction fees and taxes; given in percent of traded value.

open to open a trade means to buy an asset or sell it short.

Sharpe ratio The Sharpe (1994) ratio is the excess return divided by the standard deviation of returns. In this case, we consider a zero benchmark, and so the raw returns are used for excess returns.

short to short an asset is to borrow an asset, and sell it, with the intention of buying it back at a later time in order to return it to the original owner.

stop-loss a threshold, usually specified in terms of percent of the initial traded price, where we will close a trade in order to stop further losses. A stop-loss of 1% means that if a product we buy decreases in price by 1% or more, we will sell it to avoid losing any more money.

take-profit a threshold, usually specified in terms of percent of the initial traded price, where we will close a trade in order to take profits. A take-profit of 1% means that if a product we buy increases in price by 1% or more, we will sell it.

Appendix B

R scripts

This appendix provides R scripts for the examples presented in Chapter 6 and elsewhere in the thesis. The scripts and resulting workspaces may be downloaded from http://tinyurl.com/condvisthesis.

B.1 Simulated data: Interaction between predictors (Section 4.6)

```
## Simulated data based on Goldstein et al. 2015.
## Mark O'Connell, August 2016.
set.seed(746182481)

X <- matrix(runif(2000 * 3, -1, 1), ncol = 3)
colnames(X) <- c("X1", "X2", "X3")

## Create indicators for the piecewise function.

gamma1 <- X[, 3] <= -0.5
gamma2 <- X[, 3] > -0.5 & X[, 3] <= 0
gamma3 <- X[, 3] > 0 & X[, 3] <= 0.5
gamma4 <- X[, 3] > 0.5

## Create the response and add Gaussian noise.

error <- rnorm(mean = 0, sd = 1, n = nrow(X))
Y <- 0.5 * X[, 1] + gamma1 * (-5 * X[, 2]) + gamma2 * (abs(5 * X[, 2])) +
  gamma3 * (-abs(5 * X[, 2])) + gamma4 * ((5 * X[, 2])) + error

dat <- data.frame(cbind(Y, X))

## Marginal plot of Y versus X2.

plot(dat$X2, dat$Y, xlab = "X2", ylab = "Y", cex = 0.5)

## Fit a gradient boosted tree model.
```

```
library(gbm)
model <- gbm(Y ~ ., data = dat, distribution = "gaussian", n.trees = 400,
  interaction.depth = 3, shrinkage = 0.2)

## Produce some ICE plots.

library(ICEbox)
ICEobject <- ice(model, dat, predictor = "X2", predictfcn = function(object,
  newdata, ...) predict(object, newdata, n.trees = object$n.trees, ...),
  num_grid_pts = 50)

plot(ICEobject, frac_to_plot = 0.2, cex = 0.3)

plot(ICEobject, frac_to_plot = 0.2, cex = 0.3, centered = TRUE)

## Condition the ICE plot on X3 like trellis graphics.

o <- ice(model, dat, predictor = "X2", predictfcn = function(object,
  newdata, ...) predict(object, newdata, n.trees = object$n.trees, ...),
  num_grid_pts = 50, indices_to_build = gamma1)
plot(o, cex = 0.3, main = "X3 < -0.5", ylim = c(-6, 6))

o <- ice(model, dat, predictor = "X2", predictfcn = function(object,
  newdata, ...) predict(object, newdata, n.trees = object$n.trees, ...),
  num_grid_pts = 50, indices_to_build = gamma2)
plot(o, cex = 0.3, main = "-0.5 < X3 <= 0", ylim = c(-6, 6))

o <- ice(model, dat, predictor = "X2", predictfcn = function(object,
  newdata, ...) predict(object, newdata, n.trees = object$n.trees, ...),
  num_grid_pts = 50, indices_to_build = gamma3)
plot(o, cex = 0.3, main = "0 < X3 <= 0.5", ylim = c(-6, 6))

o <- ice(model, dat, predictor = "X2", predictfcn = function(object,
  newdata, ...) predict(object, newdata, n.trees = object$n.trees, ...),
  num_grid_pts = 50, indices_to_build = gamma4)
plot(o, cex = 0.3, main = "0.5 < X3", ylim = c(-6, 6))

save.image("simulated/interaction-workspace.rda")
```

B.2 Simulated data: Correlated predictors (Section 5.7)

```
## Simulated data with correlated predictors.
## Mark O'Connell, August 2016.
set.seed(746182481)

library(mvtnorm)

n <- 500

x1 <- runif(n)
x23 <- rmvnorm(n, sigma = matrix(c(1, 0.6, 0.6, 1), ncol = 2))
```

```
x2 <- x23[, 1]
x3 <- x23[, 2]
y <- sin(10 * x1) + x2 + x3

d <- data.frame(y, x1, x2, x3)

## Scatterplot matrix

pairs(d)

## Trellis plot

coplot(y ~ x1 | x2 + x3)

library(e1071)

model <- svm(y ~ ., data = d)

set.seed(746182481)
path <- makepath(x = d[, c("x2", "x3")], ncentroids = 15)

plot(d$x2, d$x3, asp = 1, col = "gray", xlab = "x2", ylab = "x3")
points(path$centers, pch = 16)
points(path$path, type = "l")
points(path$path[14, , drop = FALSE], pch = "+", cex = 4, col = "red")
text(path$centers[c(1, nrow(path$centers)), , drop = FALSE], labels = c("start",
  "end"), pos = 2)

save.image("simulated/correlated-workspace.rda")
```

B.3 Power plant example (Section 6.2)

```
## Power plant data
## Mark O'Connell, August 2016
set.seed(746182481)

library(condvis)
data(powerplant)

## Scale the response, mainly for the neural network.

powerplant$PE <- condvis:::scale2unit(powerplant$PE)

## Fit an additive spline model.

library(mgcv)
model.gam <- mgcv::gam(PE ~ s(AT) + s(V) + s(AP) + s(RH), data = powerplant)

## Fit a support vector machine with radial kernel.

library(e1071)
```

```
model.svm <- svm(PE ~ ., data = powerplant, epsilon = 0.4, gamma = 0.25)

## Fit a neural network.

library(nnet)
model.nnet <- nnet(PE ~ ., data = powerplant, size = 20, decay = 0.5, maxit =
  300)

models <- list(gam = model.gam, svm = model.svm, nnet = model.nnet)

save.image(file = "powerplant-workspace.rda")
```

B.4 Wine example (Section 6.3)

```
## Wine data
## Mark O'Connell, July 2016
set.seed(746182481)
library(condvis)

data(wine)

## Include 'caret' package for cross-validation.

library(caret)

## Train a support vector machine.

control.svm <- trainControl(method = "cv", number = 5)
tunegrid.svm <- expand.grid(sigma = c(0.01, 0.05, 0.1, 0.5), C = c(0.5, 1, 10,
  100))
cv.svm <- train(Class ~ Alcohol + Malic + Ash + Magnesium + Phenols +
  Flavanoids, data = wine, method = "svmRadial", trControl = control.svm,
  tuneGrid = tunegrid.svm)

final.svm <- structure(list(model = cv.svm$finalModel), class = "ksvmpred")

## S4 objects have a rigid approach to methods, so make an S3 wrapper to pass
## to 'ceplot'.

predict.ksvmpred <- function(object, newdata, ...)
{
  if (missing(newdata))
    predict(object$model)
  else predict(object$model, newdata = newdata[, colnames(object$model@xmatrix[[
    1]])])
}

## Train a random forest.

control.rf <- trainControl(method = "cv", number = 5)
tunegrid.rf <- expand.grid(mtry = 2:6)
```

```
cv.rf <- train(Class ~ Alcohol + Malic + Ash + Magnesium + Phenols +
  Flavanoids, data = wine, method = "rf", trControl = control.rf, tuneGrid =
  tunegrid.rf)

final.rf <- cv.rf$finalModel

## Train a gradient boosted model.

library(gbm)
cv.gbm <- gbm(Class ~ Alcohol + Malic + Ash + Magnesium + Phenols +
  Flavanoids, data = wine, cv.folds = 5, train.fraction = 0.8, distribution =
  "multinomial", verbose = TRUE, n.trees = 400, shrinkage = 0.02,
  n.minobsinnode = 5, interaction.depth = 3)

final.gbm <- gbm(Class ~ Alcohol + Malic + Ash + Magnesium + Phenols +
  Flavanoids, data = wine, cv.folds = 5, train.fraction = 0.8, distribution =
  "multinomial", verbose = TRUE, n.trees = 1.2 * which.min(cv.gbm$cv.error),
  shrinkage = 0.02, n.minobsinnode = 5, interaction.depth = 3)

save.image("wine-workspace.rda")
```

B.5 Credit card defaults example (Section 6.4)

```
## Analysis of credit card default data from Taiwan, from the UCI repository.
## Mark O'Connell, July 2016.
set.seed(746182481)

load("creditcard.rda")

## Rename the response and delete the ID variable.

colnames(creditcard)[ncol(creditcard)] <- "default"
creditcard$ID <- NULL

nr <- nrow(creditcard)

## Hold out 8,000 cases for testing, and train on the remainder.

testindex <- sample(1:nr, 8000)
test <- creditcard[testindex, ]
train <- creditcard[-testindex, ]

## Train a gradient boosted tree model. Set `shrinkage`, `interaction.depth` and
## `n.minobsinnode` to reasonable values and cross-validate for best number of
## trees.

library(gbm)

model.cv <- gbm(formula = default ~ ., distribution = "adaboost", data = train,
  n.trees = 800, interaction.depth = 4, n.minobsinnode = 10, shrinkage = 0.03,
  bag.fraction = 0.7, train.fraction = 0.7, cv.folds = 5, verbose = TRUE)
```

```
## Train a single model with ‘n.trees‘ chosen by minimum cross validation error.

model <- gbm(formula = default ~ ., distribution = "adaboost", data = train,
  n.trees = which.min(model.cv$cv.error), interaction.depth = 4, n.minobsinnode
  = 10, shrinkage = 0.03, bag.fraction = 0.7, verbose = TRUE)

## Calculate training logloss

p <- predict(model, n.trees = model$n.trees, type = "response")
p <- pmax(pmin(p, 1 - 10e-15), 10e-15)
y <- train[, "default"]
logloss.contrib.train <- y * log(p) + (1 - y) * log(1 - p)
logloss.train <- -mean(logloss.contrib.train)

## Calculate test logloss.

p <- predict(model, newdata = test, n.trees = model$n.trees, type = "response")
p <- pmax(pmin(p, 1 - 10e-15), 10e-15)
y <- test[, "default"]
logloss.contrib.test <- y * log(p) + (1 - y) * log(1 - p)
logloss.test <- -mean(logloss.contrib.test)

## Calculate the variable importance according to the gbm.

varimp <- rownames(summary(model, plot = FALSE))

save.image("creditcard-workspace.rda")
```

B.6 Blog comments example (Section 6.5)

```
## Code to examine blog comments data from UCI repository.
## Mark O'Connell, October 2016.
library("xgboost")

set.seed(746182481)

load("blogtrain.rda")
load("blogtest.rda")

## Remove predictors that have only one unique value.

removevars <- which(vapply(blogdata, function(x) length(unique(x)), integer(1L))
  <= 1)
train <- blogdata[, -removevars]

## Create binary target, and remove 'comments' column.

target <- as.integer(train$comments > 0)
train$comments <- NULL
train$target <- target
```

```
## Train a classifier to predict probability of comments.

classifier.params <- list(
  booster = "gbtree"
, objective = "binary:logistic"
, eta = 0.03
, gamma = 0.1
, max_depth = 6
, min_child_weight = 1
, colsample_bytree = 0.7
, eval_metric = "logloss")

## Formula to create model matrix.

f <- formula(paste0("~ 0+", paste(setdiff(colnames(train), c("comments",
  "target")), collapse = "+")))

## Function to train an XGBoost model, stopping when further iterations increase
## the loss on the validation set.

trainXGB <- function(params, formula, data, trainindex, nrounds = 1500){
  x.train <- xgb.DMatrix(model.matrix(formula, data = data[trainindex, ]),
    label = target[trainindex])
  x.validate <- xgb.DMatrix(model.matrix(formula, data = data[-trainindex, ]),
    label = target[-trainindex])
  suppressWarnings(
    structure(list(model = xgb.train(nrounds = nrounds, params = params, data =
      x.train, watchlist = list("validate" = x.validate, "train" = x.train),
      print.every.n = 20, nthread = 4, early.stop.round = 200), formula =
      formula), class = "xgbpred")
    )
  }

## Split the data into `nfolds`, and train the same number of models, holding
## out one fold at a time as the validation set. The number of boosting
## iterations for each model is chosen by the performance on the validation set.
## This populates the ensemble of classifiers.

classifiers <- list()
nfolds <- 5
j <- 1
folds <- sample(rep(1:nfolds, length.out = nrow(train)))
for (i in 1:nfolds){
  fitindex <- which(folds != i)
  classifiers[[j]] <- trainXGB(classifier.params, f, train, fitindex)
  cat("\nFinished training classifier", j, "\n")
  j <- j + 1
}

names(classifiers) <- paste("xgb", 1:(j - 1), sep = "-")
```

```
classifiers.ensemble <- structure(classifiers, class = "ensemble")

## Method to make predictions from an XGBoost model.

predict.xgbpred <- function(object, newdata, ...){
  newdata <- if (missing(newdata))
    train
  else newdata
  mm <- model.matrix(f, data = newdata)
  newx <- xgb.DMatrix(mm)
  predict(object$model, newx)
}

## Method to make predictions from an ensemble.

predict.ensemble <- function(object, newdata, ...){
  newdata <- if (missing(newdata))
    train
  else newdata
  preds <- lapply(object, predict, newdata = newdata)
  out <- rowMeans(data.frame(preds))
  out
}

## Calculate the logloss on the 'unseen' test data.

logloss <- function(y, p){
  - mean(y * log(p) + (1 - y) * log(1 - p))
}

test.logloss <- logloss(test$comments > 0, predict(classifiers.ensemble,
  newdata = test))

## Extract variable importance from one of the classifiers in the ensemble.

varimp <- xgb.importance(feature_names = setdiff(colnames(train), "comments"),
  model = classifiers[[1]]$model)

ensemble <- classifiers
ensemble[[length(ensemble) + 1]] <- classifiers.ensemble
names(ensemble)[length(ensemble)] <- "ensemble"

save.image("blog-workspace.rda")
```

B.7 Prostate data example (Section 6.6)

```
## Analysis of prostate data
## Mark O'Connell, August 2016.

## Include `ElemStatLearn` for data, and `rstan` to do MCMC
```

```
library(ElemStatLearn)
library(rstan)

## Load data, scale and create design matrix

data(prostate)
trainindex <- which(prostate$train)
prostate$train <- NULL

prostate.scaled <- scale(prostate[, setdiff(colnames(prostate), c("train",
  "lpsa"))])

lpsa <- scale(prostate$lpsa)

f <- ~ 0 + lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45

x <- scale(model.matrix(f, data = prostate[trainindex, ]), center = attr(
  prostate.scaled, "scaled:center"), scale = attr(prostate.scaled,
  "scaled:scale"))

## Create list of data for stan

standata <- list(
  N = nrow(x),
  K = ncol(x),
  y = lpsa[trainindex],
  x = x,
  lambda = 0.0001
  )

## Model specification, taken from Stan manual 2.6.0

stanmodelcode <-
"
  data {
    int<lower=0> N;
    int<lower=1> K;
    vector[N] y;
    matrix[N,K] x;
    real<lower=0> lambda;
  }
  parameters {
    vector[K] beta;
  }
  transformed parameters {
    real<lower=0> squared_error;
    squared_error <- dot_self(y - x * beta);
  }
  model {
    increment_log_prob(- squared_error);
    increment_log_prob(- lambda * dot_self(beta));
```

```
  }
  generated quantities {
    real<lower=0> sigma_squared;
    sigma_squared <- squared_error / (N - 1);
  }
"

## Compile the model and run sampling (slow)

stanfit <- stan(model_code = stanmodelcode, data = standata)

## Extract posterior samples of parameters

beta.postsample <- extract(stanfit)$beta

## Create an S3 object and an accompanying predict method

stanobj <- structure(beta.postsample, class = c("stanpred", "custompred"))

predict.stanpred <- function (object, newdata, interval = "none", ...){
  newx <- scale(model.matrix(f, data = newdata), center = attr(prostate.scaled,
    "scaled:center"), scale = attr(prostate.scaled, "scaled:scale"))
  y <- newx %*% t(object)
  if (identical(interval, "none")){
    out <- apply(y, 1, median) + mean(prostate$lpsa)
  } else if (identical(interval, "confidence")){
    out <- data.frame(fit = apply(y, 1, median) + mean(prostate$lpsa), lwr =
    apply(y, 1, quantile, 0.025) + mean(prostate$lpsa), upr = apply(y, 1,
    quantile, 0.975) + mean(prostate$lpsa))
  }
  out
}

## Set shrinkage parameter to 15, and take samples

standata$lambda <- 15
stansample_lambda_2 <- sampling(stanfit@stanmodel, data = standata)
beta.postsample_2 <- extract(stansample_lambda_2)$beta
stanobj_2 <- structure(beta.postsample_2, class = c("stanpred", "custompred"))

## Set shrinkage parameter to 50, and take samples

standata$lambda <- 50
stansample_lambda_3 <- sampling(stanfit@stanmodel, data = standata)
beta.postsample_3 <- extract(stansample_lambda_3)$beta
stanobj_3 <- structure(beta.postsample_3, class = c("stanpred", "custompred"))

models <- list(lambda_0.5 = stanobj, lambda_15 = stanobj_2, lambda_50 =
  stanobj_3)

save.image("prostate-workspace.rda")
```

B.8 Trading strategy example (Section 6.7)

B.8.1 Simulate trading

```
## Code to create simulated trading data.
## Mark O'Connell, October 2016.
load("cleandata.rda")

## EMAcpp for exponential moving average, based on
## http://www.eckner.com/papers/ts\_alg.pdf

Rcpp::cppFunction("
NumericVector EMAcpp(NumericVector x, IntegerVector time, double tau){
  int n = x.size();
  NumericVector out(n);
  out[0] = x[0];
  for(int i = 1; i < n; ++i){
    double w = exp(-(time[i] - time[i - 1]) / tau);
    out[i] = out[i - 1] * w + x[i] * (1 - w);
  }
  return(out);
}
")

## tradecpp for simulating trades

Rcpp::cppFunction('
NumericVector tradecpp(NumericVector price, int index, String action, double
  takeprofit, double stoploss, double tradingcost){
  index = index - 1;
  int i = index;
  double r = 999;
  if (action == "buy"){
    double profittarget = (1 + (takeprofit + tradingcost) / 100) * price[index];
    double losstarget = (1 - (stoploss + tradingcost) / 100) * price[index];
    while (i < price.size()){
      if (price[i] > profittarget){
        r = (price[i] / price[index] - 1) * 100 - tradingcost;
        break;
      }
      if (price[i] < losstarget){
        r = (price[i] / price[index] - 1) * 100 - tradingcost;
        break;
      }
      i = i + 1;
    }
  }
  if (action == "sell"){
    double profittarget = (1 - (takeprofit + tradingcost) / 100) * price[index];
    double losstarget = (1 + (stoploss + tradingcost) / 100) * price[index];
    while (i < price.size()){
      if (price[i] < profittarget){
```

```
        r = (price[index] / price[i] - 1) * 100 - tradingcost;
        break;
      }
      if (price[i] > losstarget){
        r = (price[index] / price[i] - 1) * 100 - tradingcost;
        break;
      }
      i = i + 1;
    }
  }
  NumericVector out(3);
  out[0] = r;
  out[1] = index + 1;
  out[2] = i + 1;
  return(out);
}
')

a <- Sys.time()

## Create range of values for EMA window for moving average, in seconds

window <- c(5, 10, 30, seq(60, 600, 60), seq(720, 3600, 120))

## Create range of values for profit/loss limit, in percent

PL <- c(0.05, 0.1, 0.15)

## Create range of values for critical values of EMA(BA) to initiate trades

BAlim <- c(0.05, 0.1, 0.15, 0.2)

## Create empty arrays for what we want to store

totalreturn <- array(dim = c(length(window), length(PL), length(BAlim)))
meanreturn <- totalreturn
sdreturn <- totalreturn
totaltrades <- totalreturn

## Create bid-ask size indicator, as log of bid size over ask size

BA <-log(quotes.interp$bsize / quotes.interp$asize)

for(k in seq_along(window)){

  ## Calculate exponential moving average of BA

  BAema <- EMAcpp(BA, as.integer(quotes.interp$time), window[k])
  for(k2 in seq_along(BAlim)){

    ## convert BAema to sell-stay-buy recommendation according to BAlim
```

```
    recommend <- c("sell", "stay", "buy")[findInterval(BAema, c(-BAlim[k2],
      BAlim[k2])) + 1]
    for(k1 in seq_along(PL)){
      mytrades <- data.frame()
      i <- window[k]
      j <- 1

      ## Advance time second by second, and place a trade if recommended

      while (i < nrow(trades.interp)){
        if (recommend[i] != "stay"){

          ## Place trade and evaluate outcome

          temp <- tradecpp(trades.interp$price, i, action = recommend[i], PL[k1]
            , PL[k1], 0.01)

          ## In the C++ code, 999 is assigned to trades that do not close

          if (temp[1] == 999) temp[1] <- NA
          mytrades <- rbind(mytrades, temp)
          j <- j + 1

          ## Reset time counter to the second after the trade closed

          i <- temp[3] + 1
        } else i <- i + 1
      }
      mytrades <- as.data.frame(mytrades)
      colnames(mytrades) <- c("return", "openindex", "closeindex")

      ## Assign everything we want to keep (there is some redundancy here)

      totalreturn[k, k1, k2] <- sum(mytrades$return, na.rm = TRUE)
      totaltrades[k, k1, k2] <- sum(!is.na(mytrades$return))
      meanreturn[k, k1, k2] <- mean(mytrades$return, na.rm = TRUE)
      sdreturn[k, k1, k2] <- sd(mytrades$return, na.rm = TRUE)
    }
  }
}
Sys.time() - a

## Calculate a rough Sharpe ratio

sharpe <- sqrt(totaltrades) * meanreturn / sdreturn

## Plot a trellis arrangement of totalreturn or sharpe

dev.new(height = 6, width = 8)
par(mfrow = c(length(PL), length(BAlim)))
```

```
par(mar = c(2, 2, 2, 2))
for (i in seq_along(PL)){
  for (j in seq_along(BAlim)){
    plot(window, sharpe[, i, j], ylim = c(0, 3), type = "o", col = "blue",
      main = paste("BAlim =", BAlim[j], ", PL =", PL[i]))
    abline(h = 0, col = "lightgray")
  }
}

save(window, PL, BAlim, totalreturn, totaltrades, meanreturn, sdreturn, sharpe,
  file = "simout.rda")
```

B.8.2 Model the simulation

```
## Code to model simulated trading data.
## Mark O'Connell, October 2016.
load("simout.rda")

dat <- data.frame(
    sharpe = as.vector(sharpe),
    window = rep(window, length(PL) * length(BAlim)),
    PL = rep(rep(PL, each = length(window)), length(BAlim)),
    BAlim = rep(BAlim, each = length(window) * length(PL))
)

dat <- dat[dat$sharpe < 5, ]

library(kernlab)

## Fit a Gaussian process

gp <- gausspr(sharpe ~ ., data = dat, kpar = list(sigma = 3))

## Construct an S3 wrapper for the Gaussian process model object

gp_wrapper <- structure(list(model = gp), class = "gpwrap")

## Create a predict method for the S3 wrapper

predict.gpwrap <- function(object, newdata, ...){
  if (missing(newdata))
    predict(object$model)
  else predict(object$model, newdata = newdata)
}

save.image("trading-workspace.rda")
```

B.9 Ensemble visualisation (Chapter 7)

B.9.1 Blog data model ensemble logloss calculation

```
## Code to examine model ensemble trained on blog data.
## Mark O'Connell, October 2016.

load("blog-workspace.rda")
library("xgboost")

## Function to calculate contributions to logarithmic loss.

logloss <- function (y, p){
  p <- pmax(pmin(p, 1 - 10e-15), 10e-15)
  -(y * log(p) + (1 - y) * log(1 - p))
}

## Calculate the logloss contributions for each observation and each model.

E_blog <- data.frame(lapply(classifiers, function(x) {logloss(train$target,
  predict(x, newdata = train))}))

save.image("ensemble-xgb-workspace.rda")
```

B.9.2 Power plant data model ensemble

```
## Code to examine models fit to power plant data.
## Mark O'Connell, October 2016.
load("powerplant-workspace.rda")

library(PairViz); library("mgcv"); library("e1071"); library("nnet")

## Calculate the residuals for each model in the ensemble.

E <- data.frame(lapply(models, function(x) powerplant$PE - predict(x, newdata =
  powerplant)))

## On each panel, show where zero is.

panelfun <- function (x, y, ...){
  abline(h = 0, v = 0, col = "gray")
  points(x, y)
}

## Scatterplot matrix of residuals.

pairs(E, panel = panelfun, cex = 0.1)

## Parallel coordinates plot of residuals.

pcp(E, scale = FALSE)
```

```
## Calculate similarity weights for observations relative to the first
## observation.

weights <- similarityweight(x = powerplant[1, -5], data =powerplant[, -5],
  threshold = 0.4)

## Get the plotting colours and ordering from the similarity weight.

colour <- condvis:::weightcolor("black", weights)
order <- attr(colour, "order")

## On each panel, plot observations with the colour and ordering from the
## similarity weights, so we only see observations near the first observation
## in predictor space.

panelfun1 <- function(x, y, ...){
  abline(h = 0, v = 0, col = "gray")
  points(x[order], y[order], col = colour[order], ...)
  points(x[1], y[1], col = "red", pch = "+", cex = 3)
}

## Conditional scatterplot matrix.

pairs(E, panel = panelfun1, cex = 0.4)

## Conditional parallel coordinates plot.

pcp(E, scale = FALSE, col = NA)
for (i in order){
  points(1:ncol(E), E[i, ], type = "l", col = colour[i])
}
points(1:ncol(E), E[1, ], type = "l", col = "red", lwd = 2)
```

B.9.3 Blog data model ensemble plots

```
## Code to examine model ensemble trained on blog data.
## Mark O'Connell, October 2016.

library("PairViz")

load("ensemble-xgb-workspace.rda")

## Scatterplot matrix of logloss contributions for each observation.

pairs(E_blog, cex = 0.1, cex.axis = 1.2)

## Parallel coordinates plot of logloss contributions for each observation.

pcp(E_blog, scale = FALSE)

## Make an index of the worst fit observations, by worst mean logloss
## contribution across the models.
```

```
worstindex <- order(rowMeans(E_blog), decreasing = TRUE)

## Get similarity weights for observations relative to the worst fit
## observation, considering only the dimensions of the 20 most important
## predictors.

weights <- similarityweight(x = train[worstindex[1], varimp$Feature[1:20]],
  data = train[, varimp$Feature[1:20]], threshold = 0.4)

## Use the similarity weights to make plotting colours, and a plotting order.

colour <- condvis:::weightcolor("black", weights)
order <- attr(colour, "order")

## On each panel, plot the points in order with their colour determined from
## their similarity weight.

panelfun <- function(x, y, ...){
  points(x[order], y[order], col = colour[order], ...)
  points(x[worstindex[1L]], y[worstindex[1L]], col = "red", pch = "+", cex = 2)
}

## Conditional scatterplot matrix showing points near the worst fit observation.

pairs(E_blog, panel = panelfun, cex = 0.2)

## Conditional parallel coordinates plot showing points near the worst fit
## observation.

pcp(E_blog, scale = FALSE, col = NA)
for (i in order){
  points(1:ncol(E_blog), E_blog[i, ], type = "l", col = colour[i])
}
points(1:ncol(E_blog), E_blog[worstindex[1], ], type = "l", col = "red", lwd =
  2)
```

Appendix C

Package documentation

This appendix contains the condvis documentation for

- `ceplot`,
- `condtour`
- `similarityweight`.

Excerpt from condvis documentation

October 5, 2016

`ceplot` *Interactive conditional expectation plot*

Description

Creates an interactive conditional expectation plot, which consists of two main parts. One part is a single plot depicting a section through a fitted model surface, or conditional expectation. The other part shows small data summaries which give the current condition, which can be altered by clicking with the mouse.

Usage

```
ceplot(data, model, response = NULL, sectionvars = NULL,
  conditionvars = NULL, threshold = NULL, lambda = NULL,
  distance = c("euclidean", "maxnorm"), type = c("default", "separate",
  "shiny"), view3d = FALSE, Corder = "default", selectortype = "minimal",
  conf = FALSE, probs = FALSE, col = "black", pch = NULL,
  residuals = FALSE, xsplotpar = NULL, modelpar = NULL,
  xcplotpar = NULL)
```

Arguments

`data`	A dataframe containing the data to plot
`model`	A model object, or list of model objects
`response`	Character name of response in `data`
`sectionvars`	Character name of variable(s) from `data` on which to take a section, can be of length 1 or 2.
`conditionvars`	Character names of conditioning variables from `data`. These are the predictors which we can set to single values in order to produce a section. Can be a list of vectors of length 1 or 2. Can be a character vector, which is then paired up using `arrangeC`. If `NULL`, an attempt will be made to extract all variable names which are not `response` or `sectionvars` from `model`, and these will be arranged using `arrangeC`.
`threshold`	This is a threshold distance. Points further than `threshold` away from the current section will not be visible. Passed to `similarityweight`.

lambda	A constant to multiply by number of factor mismatches in constructing a general dissimilarity measure. If left NULL, behaves as though lambda is set greater than threshold, and so only observations whose factor levels match the current section are visible. Passed to similarityweight.
distance	A character vector describing the type of distance measure to use, either "euclidean" (default) or "maxnorm".
type	This specifies the type of interactive plot. "default" places everything on one device. "separate" places condition selectors on one device and the section on another. (These two options require XQuartz on OS X). "shiny" produces a Shiny application.
view3d	Logical; if TRUE plots a three-dimensional regression surface if possible.
Corder	Character name for method of ordering conditioning variables. See arrangeC.
selectortype	Type of condition selector plots to use. Must be "minimal" if type is "default". If type is "separate", can be "pcp" (see plotxc.pcp or "full" (see plotxc.full).
conf	Logical; if TRUE plots confidence bounds (or equivalent) for models which provide this.
probs	Logical; if TRUE, shows predicted class probabilities instead of just predicted classes. Only available if S specifies two numeric predictors and the model's predict method provides this.
col	Colour for observed data.
pch	Plot symbols for observed data.
residuals	Logical; if TRUE, plots a residual versus predictor plot instead of the usual scale of raw response.
xsplotpar	Plotting parameters for section visualisation as a list, passed to plotxs. Can specify xlim, ylim.
modelpar	Plotting parameters for models as a list, passed to plotxs. Not used.
xcplotpar	Plotting parameters for condition selector plots as a list, passed to plotxc. Can specify col for highlighting current section, cex, and trim (see plotxc).

See Also

condtour, similarityweight

Examples

```
## Not run:
## Example 1: Multivariate regression, xs one continuous predictor

mtcars$cyl <- as.factor(mtcars$cyl)

library(mgcv)
model1 <- list(
    quadratic = lm(mpg ~ cyl + hp + wt + I(wt^2), data = mtcars),
    additive = mgcv::gam(mpg ~ cyl + hp + s(wt), data = mtcars))

conditionvars1 <- list(c("cyl", "hp"))

ceplot(data = mtcars, model = model1, response = "mpg", sectionvars = "wt",
  conditionvars = conditionvars1, threshold = 0.3, conf = T)
```

```
## Example 2: Binary classification, xs one categorical predictor

mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$am <- as.factor(mtcars$am)

library(e1071)
model2 <- list(
  svm = svm(am ~ mpg + wt + cyl, data = mtcars, family = "binomial"),
  glm = glm(am ~ mpg + wt + cyl, data = mtcars, family = "binomial"))

ceplot(data = mtcars, model = model2, sectionvars = "wt", threshold = 1,
  type = "shiny")

## Example 3: Multivariate regression, xs both continuous

mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$gear <- as.factor(mtcars$gear)

library(e1071)
model3 <- list(svm(mpg ~ wt + qsec + cyl + hp + gear,
  data = mtcars, family = "binomial"))

conditionvars3 <- list(c("cyl","gear"), "hp")

ceplot(data = mtcars, model = model3, sectionvars = c("wt", "qsec"),
  threshold = 1, conditionvars = conditionvars3)

ceplot(data = mtcars, model = model3, sectionvars = c("wt", "qsec"),
    threshold = 1, type = "separate", view3d = T)

## Example 4: Multi-class classification, xs both categorical

mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$vs <- as.factor(mtcars$vs)
mtcars$am <- as.factor(mtcars$am)
mtcars$gear <- as.factor(mtcars$gear)
mtcars$carb <- as.factor(mtcars$carb)

library(e1071)
model4 <- list(svm(carb ~ ., data = mtcars, family = "binomial"))

ceplot(data = mtcars, model = model4, sectionvars = c("cyl", "gear"),
  threshold = 3)

## Example 5: Multi-class classification, xs both continuous

data(wine)
wine$Class <- as.factor(wine$Class)
library(e1071)

model5 <- list(svm(Class ~ ., data = wine, probability = TRUE))

ceplot(data = wine, model = model5, sectionvars = c("Hue", "Flavanoids"),
  threshold = 3, probs = TRUE)

ceplot(data = wine, model = model5, sectionvars = c("Hue", "Flavanoids"),
  threshold = 3, type = "separate")
```

```
ceplot(data = wine, model = model5, sectionvars = c("Hue", "Flavanoids"),
  threshold = 3, type = "separate", selectortype = "pcp")

## Example 6: Multi-class classification, xs with one categorical predictor,
##            and one continuous predictor.

mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$carb <- as.factor(mtcars$carb)

library(e1071)
model6 <- list(svm(cyl ~ carb + wt + hp, data = mtcars, family = "binomial"))

ceplot(data = mtcars, model = model6, threshold = 1, sectionvars = c("carb",
  "wt"), conditionvars = "hp")

## End(Not run)
```

---

condtour *Conditional tour; a tour through sections in data space*

Description

Whereas ceplot allows the user to interactively choose sections to visualise, condtour allows the user to pre-select all sections to visualise, order them, and cycle through them one by one. ']' key advances the tour, and '[' key goes back. Can adjust threshold for the current section visualisation with ',' and '.' keys.

Usage

```
condtour(data, model, path, response = NULL, sectionvars = NULL,
  conditionvars = NULL, threshold = NULL, lambda = NULL,
  distance = c("euclidean", "maxnorm"), view3d = FALSE,
  Corder = "default", conf = FALSE, col = "black", pch = NULL,
  xsplotpar = NULL, modelpar = NULL, xcplotpar = NULL)
```

Arguments

data	A dataframe.
model	A fitted model object, or a list of such objects.
path	A dataframe, describing the sections to take. Basically a dataframe with its colnames being conditionvars.
response	Character name of response variable in data.
sectionvars	Character name(s) of variables in data on which to take sections.
conditionvars	Character name(s) of variables in data on which to condition.
threshold	Threshold distance. Observed data which are a distance greater than threshold from the current section are not visible. Passed to similarityweight.
lambda	A constant to multiply by number of factor mismatches in constructing a general dissimilarity measure. If left NULL, behaves as though lambda is set greater than threshold, and so only observations whose factor levels match the current section are visible. Passed to similarityweight.

distance	The type of distance measure to use, either "euclidean" (default) or "maxnorm".
view3d	Logical; if TRUE, plots a three-dimensional regression surface when possible.
Corder	Character name for method of ordering conditioning variables. See arrangeC.
conf	Logical; if TRUE, plots confidence bounds or equivalent when possible.
col	Colour for observed data points.
pch	Plot symbols for observed data points.
xsplotpar	Plotting parameters for section visualisation as a list, passed to plotxs. Not used.
modelpar	Plotting parameters for models as a list, passed to plotxs. Not used.
xcplotpar	Plotting parameters for condition selector plots as a list, passed to plotxc. Can specify cex.axis, cex.lab , tck, col for highlighting current section, cex.

Value

Produces a set of interactive plots. One device displays the current section. A second device shows the the current section in the space of the conditioning predictors given by conditionvars. A third device shows some simple diagnostic plots; one to show approximately how much data are visible on each section, and another to show what proportion of data are *visited* by the tour.

See Also

ceplot, similarityweight

Examples

```
## Not run:

data(powerplant)
library(e1071)
model <- svm(PE ~ ., data = powerplant)
path <- makepath(powerplant[-5], 25)
condtour(data = powerplant, model = model, path = path$path,
  sectionvars = "AT")

data(wine)
wine$Class <- as.factor(wine$Class)
library(e1071)
model5 <- list(svm(Class ~ ., data = wine))
conditionvars1 <- setdiff(colnames(wine), c("Class", "Hue", "Flavanoids"))
path <- makepath(wine[, conditionvars1], 50)
condtour(data = wine, model = model5, path = path$path, sectionvars = c("Hue"
  , "Flavanoids"), threshold = 3)


## End(Not run)
```

`similarityweight` *Calculate the similarity weight for a set of observations*

Description

Calculate the similarity weight for a set of observations, based on their distance from some arbitary points in data space. Observations which are very similar to the point under consideration are given weight 1, while observations which are dissimilar to the point are given weight zero.

Usage

```
similarityweight(x, data, threshold = NULL, distance = NULL,
  lambda = NULL)
```

Arguments

`x`	A dataframe describing arbitrary points in the space of the data (i.e., with same `colnames` as `data`).
`data`	A dataframe representing observed data.
`threshold`	Threshold distance outside which observations will be assigned similarity weight zero. This is numeric and should be > 0. Defaults to 1.
`distance`	The type of distance measure to be used, currently just two types of Minkowski distance: `"euclidean"` (default), and `"maxnorm"`.
`lambda`	A constant to multiply by the number of categorical mismatches, before adding to the Minkowski distance, to give a general dissimilarity measure. If left `NULL`, behaves as though `lambda` is set larger than `threshold`, meaning that one factor mismatch guarantees zero weight.

Details

Similarity weight is assigned to observations based on their distance from a given point. The distance is calculated as Minkowski distance between the numeric elements for the observations whose categorical elements match, with the option to use a more general dissimilarity measure comprising Minkowski distance and a mismatch count.

Value

A numeric vector or matrix, with values from 0 to 1. The similarity weights for the observations in `data` arranged in rows for each row in `x`.

See Also

`dist1`

Examples

```
## Say we want to find observations similar to the first observation.
## The first observation is identical to itself, so it gets weight 1. The
## second observation is similar, so it gets some weight. The rest are more
## different, and so get zero weight.
```

```
data(mtcars)
similarityweight(x = mtcars[1, ], data = mtcars)

## By increasing the threshold, we can find observations which are more
## approximately similar to the first row. Note that the second observation
## now has weight 1, so we lose some ability to discern how similar
## observations are by increasing the threshold.

similarityweight(x = mtcars[1, ], data = mtcars, threshold = 5)

## Can provide a number of points to 'x'. Here we see that the Mazda RX4 Wag
## is more similar to the Merc 280 than the Mazda RX4 is.

similarityweight(mtcars[1:2, ], mtcars, threshold = 3)
```

Bibliography

Adler, D. and Murdoch, D. (2016), *rgl: 3D Visualization Using OpenGL*, r package version 0.95.1441.

Aeberhard, S., Coomans, D., and de Vel, O. (1992), "Comparison of Classifiers in High Dimensional Settings (92-02)," Tech. rep., Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland.

Anand, A. and Talbot, J. (2016), "Automatic Selection of Partitioning Variables for Small Multiple Displays," *IEEE Transactions on Visualization and Computer Graphics*, 22, 669–677.

Asimov, D. (1985), "The Grand Tour: A Tool for Viewing Multidimensional Data," *SIAM Journal on Scientific & Statistical Computing*, 6, 128–143.

Atkinson, A. C. (1982), "Regression Diagnostics, Transformations and Constructed Variables," *Journal of the Royal Statistical Society. Series B (Methodological)*, 44, pp. 1–36.

Bean, L. H. (1929), "A Simplified Method of Graphic Curvilinear Correlation," *Journal of the American Statistical Association*, 24, 386–397.

— (1930), "Application of a Simplified Method of Correlation to Problems in Acreage and Yield Variations," *Journal of the American Statistical Association*, 25, 428–439.

Becker, R. A. and Cleveland, W. S. (1987), "Brushing Scatterplots," *Technometrics*, 29, pp. 127–142.

Becker, R. A., Cleveland, W. S., and Shyu, M.-J. (1996), "The Visual Design and Control of Trellis Display," *Journal of Computational and Graphical Statistics*, 5, pp. 123–155.

Bentley, J. L. (1975), "A Survey of Techniques for Fixed-radius Near Neighbour Searching," Tech. Rep. SLAC-186, Stanford Linear Accelerator Center.

Brahm, D., Snow, G., Seeliger, C., and Bengtsson, H. (2014), *sudoku: Sudoku Puzzle Generator and Solver*, r package version 2.6.

Breheny, P. and Burchett, W. (2016), *visreg: Visualization of Regression Models*, r package version 2.3-0.

Breiman, L. (2001), "Random Forests," *Machine Learning*, 45, 5–32.

Breiman, L., Friedman, J., Stone, C., and Olshen, R. (1984), *Classification and Regression Trees*, The Wadsworth and Brooks-Cole Statistics-Probability series, Taylor & Francis.

Buja, A., Krieger, A., and George, E. (2010), "A Tool for Mining Large Correlation Tables: The Association Navigator," *report to the Simons Foundation Autism Research Initiative*.

Buja, A., McDonald, J., Michalak, J., and Stuetzle, W. (1991), "Interactive data visualization using focusing and linking," in *Proceedings of IEEE Conference on Visualization '91*, pp. 156–163, 419.

Buza, K. (2014), "Feedback Prediction for Blogs," in *Data Analysis, Machine Learning and Knowledge Discovery*, eds. Spiliopoulou, M., Schmidt-Thieme, L., and Janning, R., Cham: Springer International Publishing, pp. 145–152.

Chang, W., Cheng, J., Allaire, J., Xie, Y., and McPherson, J. (2015), *shiny: Web Application Framework for R*, r package version 0.12.1.

Chang, W. and Wickham, H. (2015), *ggvis: Interactive Grammar of Graphics*, r package version 0.4.2.

Chen, H., Loeppky, J. L., Sacks, J., and Welch, W. J. (2016a), "Analysis Methods for Computer Experiments: How to Assess and What Counts?" *Statistical Science*, 31, 40–60.

Chen, T. and Guestrin, C. (2016), "XGBoost: A Scalable Tree Boosting System." in *22nd SIGKDD Conference on Knowledge Discovery and Data Mining.*

Chen, T., He, T., and Benesty, M. (2016b), *xgboost: Extreme Gradient Boosting*, r package version 0.4-4.

Cleveland, W. S. (1993), *Visualizing Data*, New Jersey: Summit Press.

Cook, D., Buja, A., Cabrera, J., and Hurley, C. (1995), "Grand Tour and Projection Pursuit," *Journal of Computational and Graphical Statistics*, 4, pp. 155–172.

Cook, D. and Swayne, D. (2007), *Interactive and Dynamic Graphics for Data Analysis*, Springer-Verlag New York.

Cook, R. D. (1993), "Exploring Partial Residual Plots," *Technometrics*, 35, pp. 351–362.

— (1998), *Regression Graphics: Ideas for Studying Regressions Through Graphics*, A Wiley-Interscience publication, Wiley.

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

Dietterich, T. G. (2000), "Ensemble Methods in Machine Learning," Tech. rep., First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science.

Earle, D. and Hurley, C. B. (2015), "Advances in Dendrogram Seriation for Application to Visualization," *Journal of Computational and Graphical Statistics*, 24, 1–25.

Edelsbrunner, H., Kirkpatrick, D., and Seidel, R. (1983), "On the shape of a set of points in the plane," *IEEE Transactions on Information Theory*, 29, 551–559.

Emerson, J. W., Green, W. A., Schloerke, B., Crowley, J., Cook, D., Hofmann, H., and Wickham, H. (2013), "The Generalized Pairs Plot," *Journal of Computational and Graphical Statistics*, 22, 79–91.

Ezekiel, M. (1924), "A Method of Handling Curvilinear Correlation for any Number of Variables," *Journal of the American Statistical Association*, 19, pp. 431–453.

Ezekiel, M. and Fox, K. (1959), *Methods of Correlation and Regression Analysis: Linear and Curvilinear*, Wiley.

Fox, J. (1987), "Effect Displays for Generalized Linear Models," in *Sociological Methodology 1987 (Volume 17)*, ed. Clogg, C. C., Washington, D. C.: American Sociological Association, pp. 347–361.

— (2003), "Effect Displays in R for Generalised Linear Models," *Journal of Statistical Software*, 8 (15), pp. 1 – 27.

Fox, J., Weisberg, S., Friendly, M., Hong, J., Andersen, R., Firth, D., and Taylor, S. (2016), *effects: Effect Displays for Linear, Generalized Linear, and Other Models*, r package version 3.3.1.

Friedman, J. H. (2001), "Greedy Function Approximation: A Gradient Boosting Machine," *The Annals of Statistics*, 29, pp. 1189–1232.

Friendly, M. (2013), "The Generalized Ridge Trace Plot: Visualizing Bias and Precision," *Journal of Computational and Graphical Statistics*, 22, 50–68.

Furnas, G. W. and Buja, A. (1994), "Prosection views: Dimensional inference through sections and projections," *Journal of Computational and Graphical Statistics*, 3, 323–385.

Geisser, S. (1993), *Predictive Inference*, Chapman & Hall/CRC Monographs on Statistics & Applied Probability, Taylor & Francis.

Goldstein, A., Kapelner, A., and Bleich, J. (2016), *ICEbox: Individual Conditional Expectation Plot Toolbox*, r package version 1.1.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015), "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation," *Journal of Computational and Graphical Statistics*, 24, pp 44–65.

Gower, J. C. (1971), "A General Coefficient of Similarity and Some of Its Properties," *Biometrics*, 27, 857–871.

Grolemund, G. and Wickham, H. (2015), "Visualizing Complex Data With Embedded Plots," *Journal of Computational and Graphical Statistics*, 24, 26–43.

— (2016), *R for Data Science*, O'Reilly Media.

Hamming, R. W. (1950), "Error detecting and error correcting codes," *Bell System technical journal*, 29, 147–160.

Hastie, T. (2013), *gam: Generalized Additive Models*, r package version 1.09.1.

Hastie, T. and Tibshirani, R. (1990), *Generalized Additive Models*, Chapman & Hall/CRC Monographs on Statistics & Applied Probability, Taylor & Francis.

Hastie, T., Tibshirani, R., and Friedman, J. (2009), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, Springer Series in Statistics, Springer.

Henderson, H. V. and Velleman, P. F. (1981), "Building Multiple Regression Models Interactively," *Biometrics*, 37, pp. 391–411.

Hoerl, A. E. and Kennard, R. W. (1970), "Ridge Regression: Applications to Nonorthogonal Problems," *Technometrics*, 12, pp. 69–82.

Hurley, C. B. and Earle, D. (2013), *DendSer: Dendrogram seriation: ordering for visualisation*, r package version 1.0.1.

Hyndman, R. J., Einbeck, J., and Wand, M. (2013), *hdrcde: Highest density regions and conditional density estimation*, r package version 3.1.

Inselberg, A. and Dimsdale, B. (1990), "Parallel Coordinates: A Tool for Visualizing Multi-dimensional Geometry," in *Proceedings of the 1st Conference on Visualization '90*, Los Alamitos, CA, USA: IEEE Computer Society Press, VIS '90, pp. 361–378.

Johnson, B. W. and McCulloch, R. E. (1987), "Added-Variable Plots in Linear Regression," *Technometrics*, 29, pp. 427–433.

Kahn, M. (2005), "An Exhalent Problem for Teaching Statistics," *Journal of Statistics Education*, 13.

Karatzoglou, A., Smola, A., and Hornik, K. (2016), *kernlab: Kernel-Based Machine Learning Lab*, r package version 0.9-24.

Karatzoglou, A., Smola, A., Hornik, K., and Zeileis, A. (2004), "kernlab – An S4 Package for Kernel Methods in R," *Journal of Statistical Software*, 11, 1–20.

Kaufman, L. and Rousseeuw, P. (2009), *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley Series in Probability and Statistics, Wiley.

Kuhn, M. (2015), *caret: Classification and Regression Training*, r package version 6.0-47.

Landwehr, J. M. and Pregibon, D. (1993), "Comment on "Improved Added Variable and Partial Residual Plots for the Detection of Influential Observations in Generalized Linear Models" by O'Hara Hines, R. J. and Carter, E. M." *Applied Statistics*, 42, pp. 16–20.

Larsen, W. A. and McCleary, S. J. (1972), "The Use of Partial Residual Plots in Regression Analysis," *Technometrics*, 14, pp. 781–790.

Liaw, A. and Wiener, M. (2015), *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*, r package version 4.6-12.

MacQueen, J. B. (1967), "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Berkeley, Calif.: University of California Press, pp. 281–297.

Mallows, C. L. (1986), "Augmented Partial Residuals," *Technometrics*, 28, pp. 313–319.

McCulloch, W. S. and Pitts, W. (1943), "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, 5, 115–133.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2014), *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, r package version 1.6-4.

Mosteller, F. and Tukey, J. (1977), *Data Analysis and Regression: A Second Course in Statistics*, Addison-Wesley series in behavioral science, Addison-Wesley Publishing Company.

Nason, M., Emerson, S., and LeBlanc, M. (2004), "CARTscans: A Tool for Visualizing Complex Models," *Journal of Computational and Graphical Statistics*, 13, pp. 807–825.

Neuwirth, E. (2014), *RColorBrewer: ColorBrewer Palettes*, r package version 1.1-2.

O'Connell, M. (2016), *condvis: Conditional Visualization for Statistical Models*, r package version 0.3-3.

O'Connell, M., Hurley, C. B., and Domijan, K. (2016), "Conditional Visualization for Statistical Models: An Introduction to the condvis Package in R," *Journal of Statistical Software*, to appear.

O'Hara Hines, R. J. and Carter, E. M. (1993), "Improved Added Variable and Partial Residual Plots for the Detection of Influential Observations in Generalized Linear Models," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 42, pp. 3–20.

OpenGL (2015), "Open GL website," `https://www.opengl.org/`, accessed: 6th May, 2015.

Parmanto, B., Munro, P. W., and Doyle, H. R. (1996), "Improving Committee Diagnosis with Resampling Techniques," *Advances in neural information processing systems*, 882–888.

R Core Team (2015), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.

Ridgeway, G. (2013), *gbm: Generalized Boosted Regression Models*, r package version 2.1.

Ripley, B. (2016), *nnet: Feed-Forward Neural Networks and Multinomial Log-Linear Models*, r package version 7.3-12.

Rockafellar, R. (2015), *Convex Analysis*, Princeton Landmarks in Mathematics and Physics, Princeton University Press.

Rokach, L. (2010), "Ensemble-based classifiers," *Artificial Intelligence Review*, 33, 1–39.

Rosner, B. (2010), *Fundamentals of Biostatistics*, Cengage Learning.

Sarkar, D. (2008), *Lattice: Multivariate Data Visualization with R*, New York: Springer, iSBN 978-0-387-75968-5.

Sharpe, W. F. (1994), "The Sharpe Ratio," *The Journal of Portfolio Management*, 21, 49–58.

Smola, A. and Vapnik, V. (1997), "Support vector regression machines," *Advances in Neural Information Processing Systems*, 9, 155–161.

Stamey, T. A., Kabalin, J. N., McNeal, J. E., Johnstone, I. M., Freiha, F., Redwine, E. A., and Yang, N. (1989), "Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate II radical prostatectomy treated patients." *The Journal of Urology*, 141, 1076–1083.

Stan Development Team (2015), *Stan: A C++ Library for Probability and Sampling v2.10.0*.

— (2016), *rstan: R interface to Stan*, r package version 2.12.1.

Stuetzle, W. (1987), "Plot Windows," *Journal of the American Statistical Association*, 82, pp. 466–475.

— (1991), "Odds Plots: A graphical aid for finding associations between views of a data set." in *Computing and Graphics in Statistics*, Springer, pp. 207 – 217.

Tüfekci, P. (2014), "Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods," *International Journal of Electrical Power & Energy Systems*, 60, pp. 126–140.

Tukey, J. and Tukey, P. (1985), "Computer Graphics and Exploratory Data Analysis: An Introduction," in *Proceedings of the Sixth Annual Conference and Exposition: Computer Graphics '85*, Fairfax, VA: National Computer Graphics Association.

Tukey, J. W. and Tukey, P. A. (1982), "Some graphics for studying four-dimensional data." in *Computer Science and Statistics: Proceedings of the 14th Symposium on the Interface*, ed. K.W. Heiner, R.S. Sacher, J.W. Wilkinson, Springer, pp. 60 – 66.

Unwin, A., Volinsky, C., and Winkler, S. (2003), "Parallel Coordinates for Exploratory Modelling Analysis," *Computational Statistics & Data Analysis*, 43, pp. 553–564.

Urbanek, S. (2013a), *rJava: Low-level R to Java interface*, r package version 0.9-6.

— (2013b), *Rserve: Binary R server*, r package version 1.7-3.

Urbanek, S. and Unwin, A. R. (2002), "Making Trees Interactive with KLIMT," *Statistical Computing and Graphics Newsletter*, 13, 13 – 16.

Urbanek, S. and Wichtrey, T. (2013), *iplots: iPlots - interactive graphics for R*, r package version 1.1-7.

Venables, W. N. and Ripley, B. D. (2002), *Modern Applied Statistics with S*, New York: Springer, 4th ed., iSBN 0-387-95457-0.

Waddell, A. R. and Oldford, R. W. (2014), *RnavGraph: Using Graphs as a Navigational Infrastructure*, r package version 0.1.8.

— (2015), *loon: Interactive Statistical Visualization*, r package version 1.0.1.

Warnes, G. R., Bolker, B., Bonebakker, L., Gentleman, R., Liaw, W. H. A., Lumley, T., Maechler, M., Magnusson, A., Moeller, S., Schwartz, M., and Venables, B. (2015), *gplots: Various R Programming Tools for Plotting Data*, r package version 2.17.0.

Welsch, R. E. (1982), "Discussion of Atkinson (1982): Regression Diagnostics, Transformations and Constructed Variables," *Journal of the Royal Statistical Society. Series B (Methodological)*.

Wickham, H. (2009), *ggplot2: elegant graphics for data analysis*, Springer New York.

Wickham, H., Cook, D., and Hofmann, H. (2015), "Visualizing statistical models: Removing the blindfold," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 8, 203–225.

Widrow, B. and Hoff, M. E. (1960), "Adaptive Switching Circuits," in *IRE WESCON Convention Record*, vol. 4, pp. 96–104.

Wilkinson, L. and Anand, A. (2012), *scagnostics: Compute scagnostics - scatterplot diagnostics*, r package version 0.2-4.

Wilkinson, L., Anand, A., and Grossman, R. L. (2005), "Graph-Theoretic Scagnostics." in *INFOVIS*, vol. 5, p. 21.

Wilkinson, L., Wills, D., Rope, D., Norton, A., and Dubbs, R. (2006), *The Grammar of Graphics*, Statistics and Computing, Springer New York.

Wilkinson, L. and Wills, G. (2008), "Scagnostics Distributions," *Journal of Computational and Graphical Statistics*, 17, pp. 473–491.

Williams, C. K. I. and Rasmussen, C. E. (1995), "Gaussian Processes for Regression," *Advances in Neural Information Processing*, 8.

Wood, F. S. (1973), "The Use of Individual Effects and Residuals in Fitting Equations to Data," *Technometrics*, 15, pp. 677–695.

Wood, S. (2015), *mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML Smoothness Estimation*, r package version 1.8-9.

Yeh, I.-C. and Lien, C. (2009), "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert Systems with Applications*, 36, 2473 – 2480.