Resampling-based inference using the `mosaic` package

Daniel Kaplan[*]
Macalester College
St. Paul, MN

Nicholas J. Horton[†]
Smith College
Northampton, MA

Randall Pruim[‡]
Calvin College
Grand Rapids, MI

March 25, 2012

Contents

1 Introduction

This document is intended to describe ways of dealing with functions that take advantage of the features of the `mosaic` package. One of the goals of the `mosaic` package is to provide elementary commands that can be easily strung together by novices without having to master the esoteric aspects of programming.

Among other things, the `mosaic` package simplifies the following:

1. Creating functions from simple mathematical formulas like $f(x) = A\log(Bx)$
2. Creating functions describing the fit of a model
3. Plotting functions in `lattice` graphics plots

[*]dtkaplan@gmail.com
[†]nhorton@smith.edu
[‡]rpruim@calvin.edu

More information about the package and this initiative can be found at the Project MOSAIC website: www.mosaic-web.org.

2 Background and setup

2.1 R and RStudio

R is an open-source statistical environment that has been used at a number of institutions to teach introductory statistics. Among other advantages, R makes it easy to demonstrate the concepts of statistical inference through randomization while providing a sensible path for beginners to progress to advanced and professional statistics. RStudio (http://www.rstudio.org) is an open-source integrated development environment for R which facilitates use of the system.

2.2 Setup

The mosaic package is available over the Internet and can be installed into R using the standard features of the system (this needs only be done once).

```
install.packages("mosaic")
```

Once installed, the package must be loaded so that it is available (this must be done within each R session). In addition to loading the package, we set the number of digits to display by default as well as the number of simulations to undertake.

```
require(mosaic)
options(digits = 3)
```

3 Creating Functions from Formulas

R provides a mechanism for creating functions of arbitrary complexity. The mosaic package provides an alternative for creating relatively simple functions that can be described with formulas.

As an example, suppose we want to define the function (family)

$$f(x) = A\log(Bx)$$

where x is our primary input to f and A and B are parameters that we way wish to adjust. That standard way of doing this in R using `function()`:

```
f <- function(x, A = 1, B = 1) A * log(B *
    x)
```

Alternatively, we can use the mosaic function `makeFun()` to achieve an equivalent function by a different means.

```
g <- makeFun(A * log(B * x) ~ x, A = 1, B = 1)
```

```
x <- 1:3; A <- 1:3; B=1:3
grid <- expand.grid(x,A,B)
functionComparison <- data.frame(x=grid[,1], A=grid[,2], B=grid[,3])
functionComparison$f <- with(functionComparison, f(x,A,B))
functionComparison$g <- with(functionComparison, g(x,A,B))
head(functionComparison)
```

```
  x A B     f     g
1 1 1 1 0.000 0.000
2 2 1 1 0.693 0.693
3 3 1 1 1.099 1.099
4 1 2 1 0.000 0.000
5 2 2 1 1.386 1.386
6 3 2 1 2.197 2.197
```

```
tally( with(functionComparison, f==g) )
```

```

 TRUE FALSE Total
   27     0    27
```

The mosaic package includes a number of utilities for working with functions created this way, including differentiation and integration operators. These are described in a separate vignette on Calculus with mosaic.

4 Creating Functions from Models

For statistics, functions often come from data rather than from formulas. While `lm()` and `predict()` can be used to fit values using a linear model fit, both the input and output formats are awkward. To fit new values, `predict()` requires that the inputs be encapsulated in a data frame. And the output of `predict()` is not a function, it is merely returns the values for a particular set of predictors. `makeFun()` can create from a linear model a function that describes the model fit. This function can then be evaluated on inputs or plotted (see the next section) like any other function. This interface is much more natural to students and others unfamiliar with all of R's data structures.

```
data(ChickWeight)
head(ChickWeight)
```

```
  weight Time Chick Diet
1     42    0     1    1
2     51    2     1    1
3     59    4     1    1
4     64    6     1    1
5     76    8     1    1
6     93   10     1    1
```

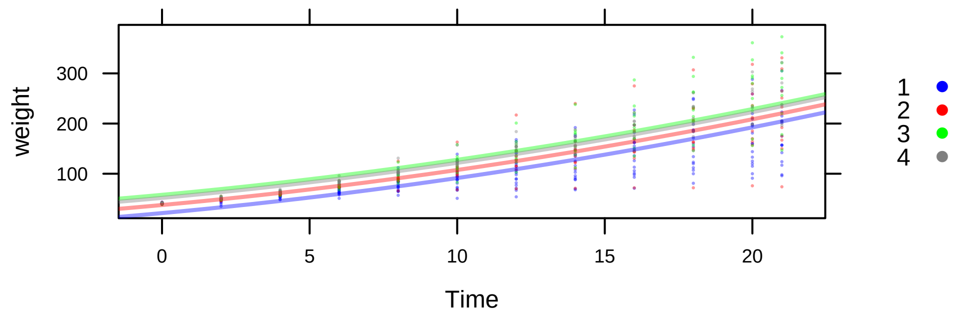
```
model <- lm(weight ~ Time + I(Time^2) + Diet,
    ChickWeight)
Wt <- makeFun(model)
Wt(Time = 5, Diet = "2")
```

```
   1
68.7


Wt(Time = 1:10, Diet = "2")

    1     2     3     4     5     6     7     8     9    10
 43.2  49.1  55.3  61.8  68.7  75.8  83.3  91.0  99.1 107.5


Wt(Time = 5, Diet = "2", interval = "confidence")

   fit  lwr  upr
1 68.7 61.7 75.6
```

5 Plotting Functions

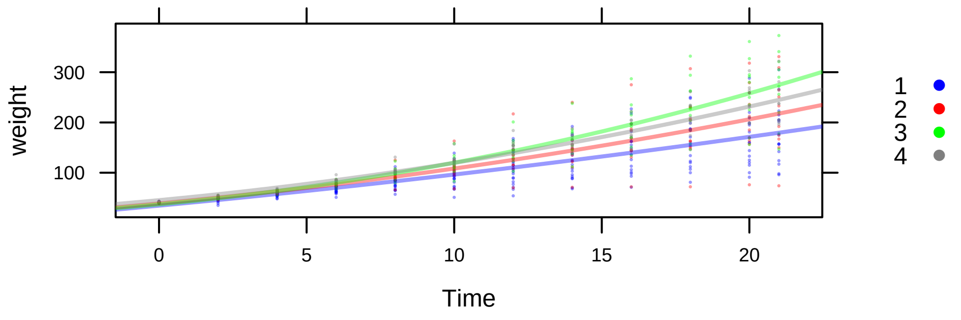
The function `plotFun()` simplifies adding functions to graphical displays.

```
xyplot(weight ~ Time, data = ChickWeight,
    groups = Diet, auto.key = list(space = "right"),
    alpha = 0.4, cex = 0.2, par.settings = list(superpose.symbol = list(col = c("blue",
        "red", "green", "gray50"), pch = 16)))
plotFun(Wt(Time, Diet = "1") ~ Time, col = "blue",
    lwd = 2, add = TRUE, alpha = 0.4)
plotFun(Wt(Time, Diet = "2") ~ Time, col = "red",
    lwd = 2, add = TRUE, alpha = 0.4)
plotFun(Wt(Time, Diet = "3") ~ Time, col = "green",
    lwd = 2, add = TRUE, alpha = 0.4)
plotFun(Wt(Time, Diet = "4") ~ Time, col = "gray50",
    lwd = 2, add = TRUE, alpha = 0.4)
```



Adding a simple interaction term gives a different fit that predicts larger differences later in the observation period.

```
model2 <- lm(weight ~ Time + I(Time^2) * Diet,
    ChickWeight)
Wt2 <- makeFun(model2)
xyplot(weight ~ Time, data = ChickWeight,
    groups = Diet, auto.key = list(space = "right"),
    alpha = 0.4, cex = 0.2, par.settings = list(superpose.symbol = list(col = c("blue",
        "red", "green", "gray50"), pch = 16)))
plotFun(Wt2(Time, Diet = "1") ~ Time, col = "blue",
    lwd = 2, add = TRUE, alpha = 0.4)
plotFun(Wt2(Time, Diet = "2") ~ Time, col = "red",
    lwd = 2, add = TRUE, alpha = 0.4)
plotFun(Wt2(Time, Diet = "3") ~ Time, col = "green",
    lwd = 2, add = TRUE, alpha = 0.4)
plotFun(Wt2(Time, Diet = "4") ~ Time, col = "gray50",
    lwd = 2, add = TRUE, alpha = 0.4)
```



6 Acknowledgments

Project MOSAIC is supported by the US National Science Foundation (DUE-0920350). More information about the package and this initiative can be found at the Project MOSAIC website: www.mosaic-web.org.

7 References

- G. W. Cobb, The introductory statistics course: a Ptolemaic curriculum?, *Technology Innovations in Statistics Education*, 2007, 1(1).
- B. Efron & R. J. Tibshirani, *An introduction to the bootstrap*, 1993, Chapman & Hall, New York.
- T Hesterberg, D. S. Moore, S. Monaghan, A. Clipson & R. Epstein. *Bootstrap methods and permutation tests (2nd edition)*, (2005), W.H. Freeman, New York.
- D. Kaplan, *Statistical modeling: A fresh approach*, http://www.macalester.edu/~kaplan/ISM.
- T. Speed, Simulation, *IMS Bulletin*, 2011, 40(3):18.