iPlots: Interactive Graphics for R

Martin Theus, Simon Urbanek
University of Augsburg, Department of Computeroriented Statistics and Data Analysis
Universitätsstr. 14, 86135 Augsburg, Germany
{martin.theus, simon.urbanek}@math.uni-augsburg.de

1 Introduction

After more than 10 years of ongoing software development in interactive statistical graphics software at the Department of Computeroriented Statistics and Data Analysis at Augsburg University, a whole suite of software packages emerged. Some of the most recent and still supported packages include KLIMT [4] by Simon Urbanek, MANET [1] by Heike Hofmann and Mondrian [3] by Martin Theus. What all these packages have in common are the interactive features, which are the building blocks for a graphical data analysis. Although interactive statistical graphics plays a major role in a data analysis process, it usually needs to be complemented with classical statistical methods or other data mining tools. None of the interactive tools mentioned above ever intended to incorporate these methods, although it would be desirable. There are some exceptions to this rule, e.g. MANET offers Biplots and Mondrian can setup loglinear models. Nonetheless, the big drawback when implementing these algorithms is of course the amount of testing needed in order to "get them right". Looking at KLIMT, we see a different option. KLIMT does not implement algorithms to build CARTs, but relies on R. It either imports the output of R via a ASCII file, or directly talks to R. Since R's communication skills for various systems are still pretty limited, this option is not very satisfactory and platform dependent.

When bringing R's method richness to interactive software is too cumbersome, why not bringing interactive graphics features to R? The basic graphics system in R dates back to times, when many of the readers where still in kindergarten, and implements the ink on paper model. This was perfectly justified at that time, but does not allow for interactive features, which we know from modern system. Thus we were forced to develop the interactive features outside of R. For a maximum platform independence JAVA was the premier choice to work with. The iPlots package is designed as a library for R which adds interactive graphics to R.

First versions of iPlots implement interactive barcharts, histograms and scatterplots. All iPlots can be manipulated from within R and can be enhanced with static elements like lines and points.

2 Design Choices

When integrating interactive graphics into R it is obviously desirable not to introduce a new way of handling objects and devices, but to stick as close to R syntax and semantics as possible. iPlots come as an R library and are invoked simply by the command `library(iplots)`. Plots in iPlots use the same names as their static counterparts, except for the leading '`i`'. E.g. to bring up an interactive histogram one simply types `ihist(data)`, where `data` is any arbitrary R vector. The most important plot options which specify colors, labels and scale information are supported transparently by iPlots.

The concept of linked highlighting, i.e. cases in a plot can be selected and light up in all other plots, as well as color brushing, i.e. all points belonging to a specific group are painted in the same color, need a concept of a data set, which allows to link cases across variables. The corresponding R construct of a dataframe would be the solution of choice. Unfortunately it is not possible to associate a variable from a dataframe with this dataframe once it was passed to a function. This would force the user to explicitly add the dataframe name (as known from the `data=...` option in model functions) when calling an iplot. To be less restrictive and more compatible, we chose to go a different way and introduced the iSets. iSets are data structures, which hold all data belonging to a relation. Within an iSet alls cases with corresponding row-ids are linked. The handling of iSets is adopted from the way devices are handled in R. When the first iPlot is generated, a corresponding iSet is created. As long as the user only uses data from the same relation – which is assumed when the data vectors are all of the same length – all data is collected in the initially created iSet and no new iSet is created. Whenever the user plots an iPlot with data from a different relation, i.e. the length of the vector is different from the number of rows in the current

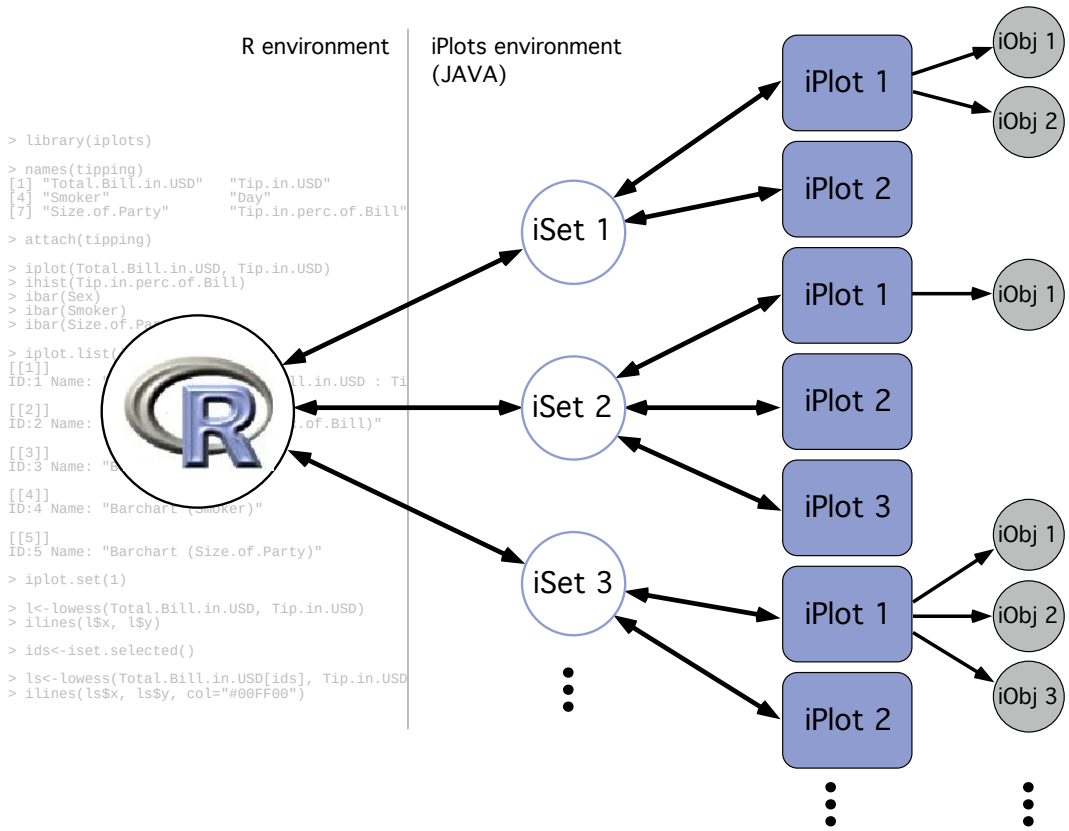


Figure 1. iSets are at the core of the iPlots data handling

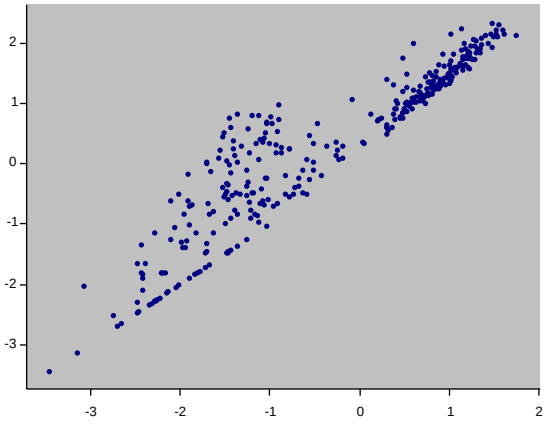
iSet, a new iSet is created. To avoid mistakes, a warning dialog prompts the user to confirm the creation of the new iSet. One can switch back and forth between iSets with next and previous functions as well as by directly specifying the iSet's id. The same scheme is used to manage iPlots within iSets.

Figure 1 shows an exemplary environment of an iPlot session. In this session data from three different relations is held in three iSets. To each of the iSets iPlots are attached. Furthermore, iObj like lines or points can be attached to an iPlot. The next Section gives a brief example of how iPlots can be used in a data analysis session in R.

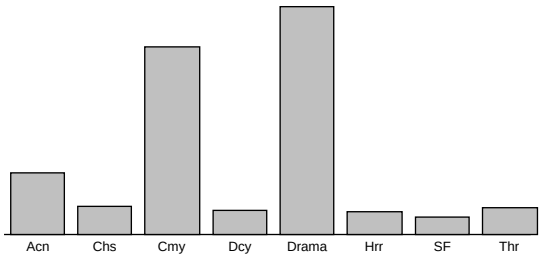
3 Sample Session: Movie Data

In this rather short section we want to show how iPlots can be used. We want to investigate the movies-dataset from Simonoff's Summer 2000 Chance article [2]. The data is available at. http://pages.stern.nyu.edu/ jsimonof/movies/. The data we use contains both the 1998 and 99 movies. Simonoff's final model distinguishes between movies with 10 or less opening screens and more than 10 opening screens. The most important variable in both parts of the model is the logged revenue of the 1st weekend. He also includes the logged number of opening screens into both parts. For the part with more than 10 opening screens the genre of the movie and the number of oscar nominees are included. To investigate this model the following session could be used:

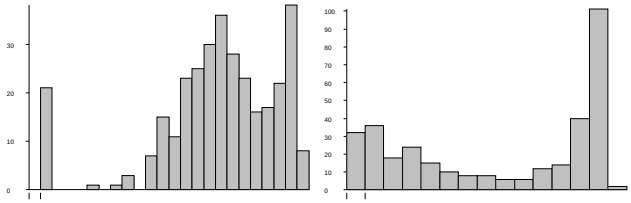
```
> movies <- read.table("F.txt", sep="\t",
+ header=T, quote='"')
> library(iplots)
> attach(movies)
> iplot(Log.1st.weekend, Log.domestic)
ID:1 Name: "Scatterplot (Log.1st.weekend : ...
```



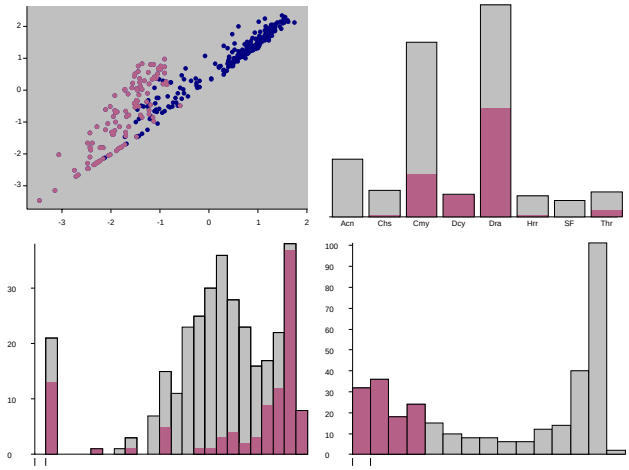
```
> ibar(Genre)
ID:2 Name: "Barchart (Genre)"
```




```
# How much of the total domestic gross was
# made after the 1st weekend?
> ihist((Total.domestic.gross - First.weekend)
+ / Total.domestic.gross)
ID:3 Name: "Histogram ((Total.domestic.gross ...
> ihist(Log.opening.screens, breaks=seq(0,4,.25))
ID:4 Name: "Histogram (Log.opening.screens.2)"
```



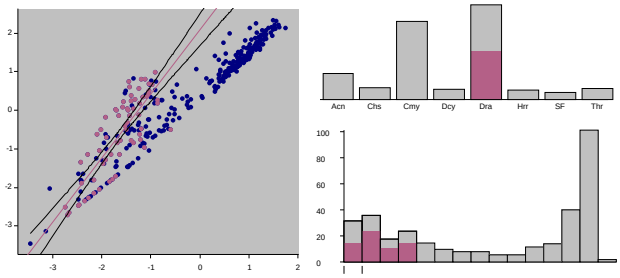
These four iPlots set up so far allow for a closer investigation of Simonoff's proposed model. The next Figure shows an example of the linked highlighting. All movies with less than 10 opening screens have been selected in the corresponding histogram.



There are a few immediate findings. The selected group contributes almost exclusively to the group of movies, which made their total domestic gross after the first weekend, and also accounts for more than half of the movies, which made their total gross only at the opening weekend. Furthermore these movies all have a rather small total domestic gross. There is a strong interaction between the selected group and the genre of the movie.

The following code was used to set up the next Figure. A regression line is added to the plot along with the 90% confidence bounds. The regression line is calculated for the currently selected points only, and changes whenever the selection changes, until the user exits the loop with the Break menu command in iPlots.

```
iplot.set(1)
r.line <- iabline(0, col="marked", visible=FALSE)
cb1 <- ilines(0, col="black", visible=FALSE)
cb2 <- ilines(0, col="black", visible=FALSE)
x.sorted <- sort(Log.1st.weekend)
while (!is.null(ievent.wait())) {
  if (iset.sel.changed()) {
    s <- iset.selected()
    if (length(s)>0) {
      y <- Log.domestic[s]
      x <- Log.1st.weekend[s]
      m <- lm(y ~ x)
      iobj.opt(r.line, reg=m, visible=TRUE)
      ci<-predict(m, data.frame(x=x.sorted),
              level=0.9, interval="confidence")
      iobj.opt(cb1, x.sorted, ci[,2], visible=TRUE)
      iobj.opt(cb2, x.sorted, ci[,3], visible=TRUE)
    } else {
      iobj.opt(list(r.line, cb1, cb2), visible=FALSE)
    }
  }
}
iobj.rm(list(r.line, cb1, cb2))
```



In this Figue only movies with less than 10 opening screens and genre "drama" are selected, showing a clear deviation from the rest of the points in the scatterplot.

4 Function Reference

4.1 Plots

This section gives a short listing of all functions related to iPlots implemented so far.

- Histogram:
  `ihist(x, breaks, col, bwidth, anchor, right, main, xlab, ylab, xlim, ylim)`
- Scatter plot:
  `iplot(x, y, col, main, xlab, ylab, xlim, ylim, log)`
- Bar chart:
  `ibar(data, col)`
  Note: Unlike the `barplot` R function, this iPlot requires the entire factor as the `data` parameter, because the highlighting uses linking at the case level.

4.2 Manipulation of plots and data sets

iPlots support two methods for accessing individual plots. First interface was defined to resemble the functions used in conjunction with graphics devices. The functions `iplot.set`, `iplot.cur`, `iplot.list`, `iplot.next`, `iplot.prev` and `iplot.off` provide means to select, retrieve or close the current iPlot. This corresponds to the `dev.`*xx* commands for R graphics devices. All plot-specific commands apply to the current iPlot, unless the `plot` parameter is set.

Second means of accessing individual plots without changing the current plot is to use the plot object returned by plot creating commands. This object can be passed via `plot` parameter to most iPlot functions, for example:

```
p<-iplot(x,y); ibar(z);
ilines(lm(y~x), plot=p)
```

Some other iPlot and iSet manipulating functions include:

- `iplot.data(id=NULL)`
  Returns the content of the plot data variables.
- `iplot.opt(..., plot = iplot.cur())` Sets options or parameters of the plot. The possible options can include those specified in the initial plot invocation.
- `iset.new(name = NULL, data = NULL)`
  Creates a new iSet and makes it current. The iSet is assigned a name of the form "data.#", where # is an unique ID.
- `iset.set(which = iset.next())`
  Change the current iSet.
- `iset.cur()`
  Returns the current iSet.
- `iset.list()`
  Lists all iSets.
- `iset.next(which = iset.cur())`
  Returns the next iSet in the list. The list is circular.
- `iset.prev(which = iset.cur())`
  Returns the previous iSet in the list.

4.3 Selection and color brushing

- `iset.select(what, mode="replace")`
  Selects cases specified by `what`, which can be either a logical vector used as a mask or a numeric vector used as list of IDs. `mode` specifies which logical operation should be applied relative to the existing selection.
- `iset.selected()`
  Returns a vector of selected cases as IDs.
- `iset.col(color=NULL, what="all")`
  Brushes specified cases with `color`. `what` can be either a list of IDs, a logical vector or the string `"all"`.

4.4 iObjects Toolkit

Every iPlot can have an arbitrary number of additional graphical objects attached, such as lines, rectangles, polygons or labels. Those objects, called iObjects, appear in the plot graphics and are programmable as of shape, position and color.

- `ilines(x, y=NULL, col=NULL, fill=NULL, ...)`
  Takes given coordinates and joins the corresponding points with line segments. The resulting iObject is a polygon or a polyline.
- `iabline(a, b=0, ...)`
  Creates a line with the specified intercept and slope.
- `itext(x, y=NULL, labels = seq(along = x), ...)`
  Adds given text labels as an iObject to the plot at specified coordinates.
- `ipoints(x, y=NULL, col=NULL, ...)`
  Creates a new iObject which draws a sequence of points at the specified coordinates.
- `iobj.list(plot = iplot.cur())`
  Lists all iObjects of the plot.
- `iobj.cur(plot = iplot.cur())`
  Returns the ID of the current iObject of the plot.
- `iobj.rm(obj = iobj.cur())`
  Removes the iObject `obj` from the associated plot.
- `iobj.set(obj)`
  Make `obj` the current iObject.
- `iobj.opt(...,obj=iobj.cur())`
  Sets options and properties of the given iObject. The parameters are object-dependent.

4.5 Event handling

iPlots offer some basic functions to make interaction between iPlots and R more flexible. These functions can be used to build animations and small interaction loops in R.

- `ievent.wait(...)`
  This function waits until an iPlots event occurs. The result is `NULL` for the *break* event.
- `iset.sel.changed(iset=iset.cur(),...)`
  Returns `TRUE` if the selection of the specified iSet changed since the last call of this function.

5 Summary & Outlook

iPlots are a R library which brings interactive graphics into the R environment. iPlots offer a variety of customization options and can be modified and enhanced directly through the R interface. In the first release iPlots offer only few plots like histogram, barchart and scatterplot. The current development is focused on adding more plots like parallel coordinate plots, mosaic plots and maps.

Further effort is made to integrate iPlots and R in an unified R interface based on Java. This combination will enhance the user experience by providing flexibility and common tools, such as window management, menus and help for all four parts of the system: console, editor, R graphics and iPlots. Finally such a tight integration will allow for much improved event handling with callback support, without the need for `ievent.wait` loops.

References

[1] H. Hofmann. Simpson on Board the Titanic? Interactive methods for dealing with multivariate categorical data. *Statistical Computing & Statistical Graphics Newsletter*, 9(2):16–19, 1998.

[2] J. Simonoff and I. Sparrow. Predicting movie grosses: Winners and losers, blockbusters and sleepers. *CHANCE Magazine*, 13(3), Summer 2002.

[3] M. Theus. *Mondrian - Interactive Statistical Graphics in JAVA*. http://www.theusRus.de/Mondrian, 2001.

[4] S. Urbanek and A. R. Unwin. Making Trees Interactive - KLIMT. In *Proc. of the 33th Symposium of the Interface of Computing Science and Statistics*, 2001.