Visualization of large multivariate datasets with the `tabplot` package

Martijn Tennekes and Edwin de Jonge

June 15, 2011

Abstract

The tableplot is a powerful visualization method to explore and analyse large multivariate datasets. In this vignette, the implementation of tableplots in R is described.

1 Introduction

The tableplot is a visualization method that is used to explore and analyse large datasets. By using tableplots, data analysts are able to observe the relationships between the variables, discover strange data patterns, and check the occurrence and selectivity of missing values.

An example of a tableplot applied to the diamonds dataset of the ggplot2 package is illustrated in Figure **??**. Each column represents a variable. The whole data set is ordered according to one or more columns (in this case, carat), and then grouped into row bins. Algorithm **??** describes the creation of a tableplot into detail.

2 Getting started with the `tableplot` function

To illustrate the `tabplot` package, we will use the diamonds dataset that is provided in the `ggplot2` package. This dataset contains information about 53,940 diamonds. There are 7 continuous variables and 3 categorical. In order to illustrate the visualization of missing values, we add several NA's.

```
> require(ggplot2)
> data(diamonds)
> is.na(diamonds$price) <- diamonds$cut == "Ideal"
> is.na(diamonds$cut) <- (runif(nrow(diamonds)) > 0.8)
```

A tableplot is simply created by the function `tableplot`:

```
> tableplot(diamonds)
```

Algorithm 1 Create tableplot

Input: Tabular dataset t, column i_s of which the distribution is of interest.

$t' \leftarrow$ sort t according to the values of column i_s.
Divide t' into n row bins according to the order of t'.
for each column i **do**
    if i is numeric **then**
        $m_{ib} \leftarrow$ mean value per bin b
        $c_{ib} \leftarrow$ fraction of missing values per bin b
    end if
    if i is categorical **then**
        $f_{ijb} \leftarrow$ frequency of each category j (including missing values) per bin b
    end if
end for
for each column i **do**
    if i is numeric **then**
        Plot a bar chart of the mean values $\{m_{i1}, m_{i2}, \ldots, m_{in}\}$. A logarithmic scale can be used. The fraction of missing values $\{c_{i1}, c_{i2}, \ldots, c_{in}\}$ determines the lightness of the bar colour. The lighter the colour, the more missing values occur in bin b. If all values are missing, a light red bar of full length is drawn.
    end if
    if i is categorical **then**
        Plot a stacked bar chart according to the frequencies $\{f_{i1b}, f_{i2b}, \ldots\}$ for each bin b. Each category is shown is a distinct colour. If there are missing values, they are depicted by a red colour.
    end if
end for

The result is depicted in Figure **??**. By default, all variables of the dataset are depicted. With the argument `colNames`, we can specify which variables are plotted.

Further, the dataset is by default sorted according to the values of the first variable. With the argument `sortCol`, we can specify on which variables the data is sorted.

```
> tableplot(diamonds, colNames = c("carat", "price", "cut", "color",
+     "clarity"), sortCol = "price")
```

The result is illustrated in Figure **??**.

Setting an appropriate number of row bins (argument `nbins`) is important, like in a histogram. A good number of row bins is a trade of between good polished but meaningless data, and detaild, but noisy data. In practice, we found out that the default number of 100 usually is a good starting

point.

The percentages near the vertical axis indicate which subset of the data in terms of units (row) is depicted. The range from 0% to 100% in Figure **??** means that all units of the data are plotted.

Suppose we want to focus our attention to the 5% most expensive diamonds. This is possible by setting the `from` argument to 0, and the `to` argument to 5:

```
> tableplot(diamonds, colNames = c("carat", "price", "cut", "color",
+     "clarity"), sortCol = "price", from = 0, to = 5)
```

Observe that in the obtained tableplot in Figure **??**, the number of row bins is still 100, so that the number of units per row bin is now 27 instead of 540. Therefore, much more detail can be observed in this tableplot.

The vertical axis contains two sets of tick marks. The set of small tick marks correspond with the row bins. The set of large tick marks correspond with the percentages. The number of tick marks between `from` and `to` is determined by R's function `pretty`.

3 Customizing the tableplot

3.1 Continuous variables

For each bin of a continuous variable, the mean value is calculated (see Algorithm **??**). When the distribution of these mean values is exponential, it is useful to apply a logarithmic transformation (see **?**). The argument `scales` is default set to the auto-detection mode `"auto"`, and can also set to linear mode `"lin"` or logarithmic mode `"log"`.

Observe that the x-axes of the variables depth and table in Figure **??** are broken. The x-axis of a variable i is broken if either

$$0 < max(m_{i1}, m_{i2}, \ldots, m_{in}) \qquad \text{AND}$$
$$\texttt{bias\_brokenX} \cdot max(m_{i1}, m_{i2}, \ldots, m_{in}) < min(m_{i1}, m_{i2}, \ldots, m_{in})$$

OR

$$0 > min(m_{i1}, m_{i2}, \ldots, m_{in}) \qquad \text{AND}$$
$$\texttt{bias\_brokenX} \cdot min(m_{i1}, m_{i2}, \ldots, m_{in}) > max(m_{i1}, m_{i2}, \ldots, m_{in}),$$

where `bias_brokenX` is a bias parameter that should be a number between 0 and 1. If `bias_brokenX=1` then the above conditions are always false, which implies that the x-axes are never broken. On the other hand, if `bias_brokenX=0` then the x-axes are always broken. By default, `bias_brokenX =0.8`, which mean that an x-axis is broken if (in case of a variable with positive values) the minimum value is at least 0.8 times the maximum value. In the diamonds dataset, this applies to the variables depth and table.

3.2 Categorical variables

The color palettes of categorical variables can be customized with the argument `pals`. Several qualitative palettes from literature (**?**, **?**) are implemented. They are stored in the list `tabplotPalettes` and can be shown by

```
> tableplot_showPalettes()
```

The default palette is a combination of Set1 and Set2. It has the advantage each category has a unique color for varibles with up to 16 categories.

Suppose we want a to use the default palette for the variable `cut`, but starting with the seventh color, pink. Further we want the color blind friendly palette for the variable `color`, but without the first color (black), and a custom palette, say a rainbow palette, for the variable `clarity`:

```
> tableplot(diamonds, pals = list(7, "col_blind_friendly(2)", rainbow(8)))
```

3.3 The `tabplot` object

The function `tableplot` returns a `tabplot` object, that can be used to make minor changes to the tableplot, for instance the order of columns or the color palettes. Of course, these changes can also be made by generating a new tableplot, such as in the examples above. However, if it takes considerable time to generate a tableplot, then it is practical to make minor changes immediately.

The output of the `tableplot` function can be assigned to a variable. The graphical output can be omitted by setting the argument `plot` to FALSE.

```
> tab <- tableplot(diamonds, plot = FALSE)
```

The `tabplot` object is a list that contains all information to depict a tableplot. The generic functions `summary` and `plot` can be applied to the `tabplot` object.

```
> summary(tab)
```

```
     general            variable1               variable2
variables:10      name       :carat        name      :cut
objects  :53940   type       :numeric      type      :categorical
bins     :100     sort       :decreasing   sort      :NA
from     :0%      scale_init :auto         categories:6
to       :100%    scale_final:lin
     variable3               variable4                variable5
name      :color         name      :clarity       name       :depth
type      :categorical   type      :categorical   type       :numeric
sort      :NA            sort      :NA            sort       :NA
```

```
categories:7             categories:8             scale_init :auto
                                                  scale_final:lin
      variable6                variable7          variable8
name       :table     name       :price     name       :x
type       :numeric   type       :numeric   type       :numeric
sort       :NA        sort       :NA        sort       :NA
scale_init :auto      scale_init :auto      scale_init :auto
scale_final:lin       scale_final:lin       scale_final:lin
      variable9                variable10
name       :y         name       :z
type       :numeric   type       :numeric
sort       :NA        sort       :NA
scale_init :auto      scale_init :auto
scale_final:lin       scale_final:lin
```

```
> plot(tab)
```

The function `changeTabplot` is used to make minor changes to a `tabplot` object. Suppose we want the columns in the order of **??**, and we want to change all color palettes to default starting with the first color.

```
> tab2 <- changeTabplot(tab, colNames = c("carat", "price", "cut",
+     "color", "clarity"), pals = list(1))
> plot(tab2)
```

4 Graphical User Interface `tableGUI`

The function `tableGUI` starts a Graphical User Interface (GUI).

```
> tableGUI()
```

5 Final remarks

Conclusions

A First appendix on the `tabplot` package

B Second appendix