Creating a complex figure using create.multiplot()

Jeff Green

July 22, 2014

Contents

1.0 Introduction

When visualizing complex data sets, it is often effective to combine multiple plots into a single figure. This tutorial will illustrate three examples of how to create a complex figure from several smaller plots using the create.multiplot function available in the "BL.plotting.general" R package. This function allows for customization of the layout and plot sizes. The first step is to install and load the package "BL.plotting.general" into R, along with its dependencies. Refer to the end of this document for session information in which this tutorial was written and tested.

```
> library(BL.plotting.general);
```

2.0 Example 1: Simple Figure

We will start by making a figure out of a single bar plot and three heat maps using the HairEyeColor data set supplied by the R datasets package.

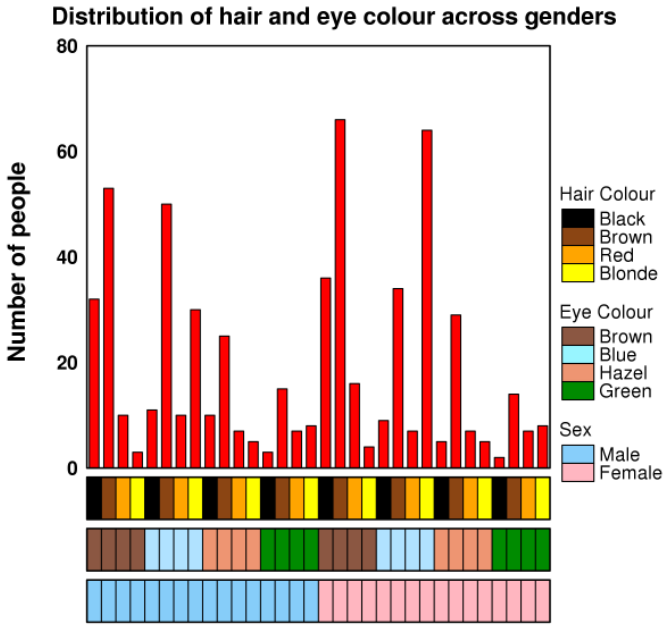

Figure 1: Finished Example 1 Plot

2.1 Setting up the data

Begin by formatting the data. The create.barplot function requires its input to be a data frame, and the create.heatmap function requires its input to be a matrix.

```
> # put the array into a dataframe that will be used in the barplot
> haireye <- as.data.frame(HairEyeColor);
>
> # put the Hair column of the dataframe into a matrix for the heat map
> Hair <- as.matrix(
>       as.numeric(haireye[, 1])
>       );
>
> # put the Eye column of the dataframe into a matrix for the heat map
> Eye <- as.matrix(
>       as.numeric(haireye[, 2])
>       );
>
> # put the Sex column of the dataframe into a matrix for the heat map
> Sex <- as.matrix(
>       as.numeric(haireye[, 3])
>       );
```

2.2 Creating the bar plot

To create a bar plot out of the formatted data is simple. The code below will create a bar plot showing the number of males and females with a particular hair and eye colour. Proper labels and titles will be added at the end using the create.multiplot function.

```
> # create the barplot used for the frequency of each type of person
> hair.eye.colour.barplot <- create.barplot(
```

```
>         formula = Freq ~ c(1:32),
>         data = haireye,
>         col = 'red',
>         # set the limits that the plot will display to 0 and 80 to make it look nicer
>         ylim = c(0.00,80)
>         );
```

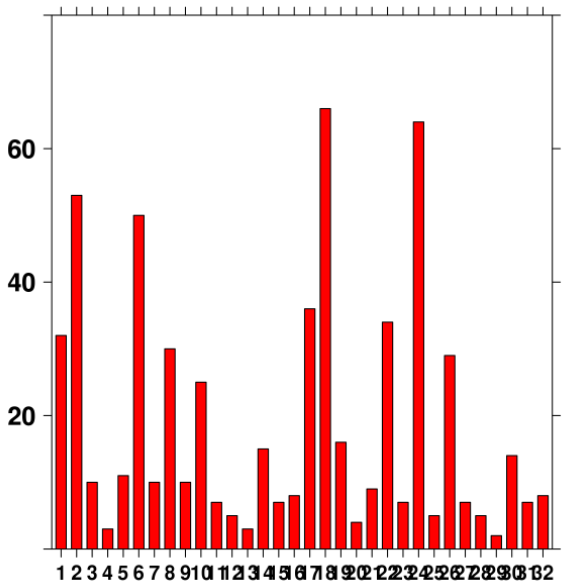

Figure 2: Hair and Eye Colour Bar Plot

2.3 Creating the first covariate bar

To create the first covariate bar we will use the create.heatmap fuction. We need to specify colours we want and allow only those colours to be used (there is only going to be one of each of the colours in the scale). This is done by specifying total.col; this paramater sets the number of diffrent colours that are shown. You must specify the number of actual colours plus one to account for white, which is shown if there is no value. Also, we remove the ticks on the y-axis and the colour key as they are not required in the final multiplot.

```
> hair.heatmap <- create.heatmap(
>         x = Hair,
>         clustering.method = 'none',
>         scale.data = FALSE,
>         colour.scheme = c("black", "chocolate4", "orange", "yellow"),
>         # only show each of those colours once (i.e, four values, four colours)
>         total.col = 5,
>         grid.col = TRUE,
>         print.colour.key = FALSE,
>         # remove y-axis ticks
>         yaxis.tck = 0,
>         height = 1
>         );
```

Figure 3: Hair Colour Covariate

2.4 Creating the second covariate bar

To create the second covariate bar, we use eye colour as the data, and choose a different colour scheme to differentiate between the heat maps.

```
> eye.heatmap <- create.heatmap(
>         x = Eye,
>         clustering.method = 'none',
>         scale.data = FALSE,
>         colour.scheme = c("lightsalmon4", "lightskyblue1", "lightsalmon2", "green4"),
>         total.col = 5,
>         grid.col = TRUE,
>         print.colour.key = FALSE,
>         yaxis.tck = 0,
>         height = 1
>         );
```

Figure 4: Eye Colour Covariate

2.5 Creating the third covariate bar

To create the third covariate bar, we use Sex as the data and again choose a different colour scheme.

```
> sex.heatmap <- create.heatmap(
>         x = Sex,
>         clustering.method = 'none',
>         scale.data = FALSE,
>         colour.scheme = force.colour.scheme(c('Male','Female'),scheme = 'sex'),
>         total.col = 3,
>         grid.col = TRUE,
>         print.colour.key = FALSE,
>         yaxis.tck = 0,
>         height = 1
>         );
```

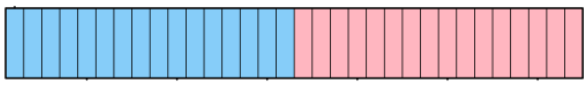

Figure 5: Sex Covariate

2.6 Creating the legend

A legend must be created to differentiate the meaning of the heat maps. Our legend will have three sections: one for Hair Colour, Eye Colour and Sex.

```
> # create the legend that will be used to show the values of the covariates
> legends <- legend.grob(list(
>
>       # create the legend for the hair colour
>       legend = list(
>               colours = c("black", "chocolate4", "orange", "yellow"),
>               title = "Hair Colour",
>               labels = c("Black", "Brown", "Red", "Blonde"),
>               size = 3,
>               title.cex = 2,
>               label.cex = 2
>               ),
>
>       # create the legend for the eye colour
>       legend = list(
>               colours = c("lightsalmon4", "lightskyblue1", "lightsalmon2", "green4"),
>               title = "Eye Colour",
>               labels = c("Brown", "Blue", "Hazel", "Green"),
>               size = 3,
>               title.cex = 2,
>               label.cex = 2
>               ),
>
>       # create the legend for the sex
>       legend = list(
>               colours = force.colour.scheme(c('Male','Female'),scheme = 'sex'),
>               title = "Sex",
>               labels = c("Male", "Female"),
>               size = 3,
>               title.cex = 2,
>               label.cex = 2
>               )
>       ),
>       title.just = "left",
>       title.fontface = "plain")
```

2.7 Creating the final figure

Now that we have all the neccessary plots and legends required to represent this data, we can combine them using the create.multiplot function. We arrange it so the bar plot will be the largest and at the top, and the three heat maps will appear below explaining what each of the bars represents (Eye colour, Hair colour and Sex). The order is achieved through listing the plots in the order in which they should appear (from bottom to top) in the plot.objects parameter. Furthermore, in this plot we will use labels only on the barplot, leaving others without labels and tick marks.

```
> create.multiplot(
>         # save the four plots that will be used in the multiplot from bottom to top
>         plot.objects = list(sex.heatmap, eye.heatmap, hair.heatmap, hair.eye.colour.barplot),
>         filename = "ex1_final_plot.png",
>         main = "Distribution of hair and eye colour across genders",
>         # set the labels on the side of the multiplot
>         # (tabs are needed for labels to be in proper location)
```

```
>         ylab.label = c("\t", "Number of people", "\t", "\t"),
>         main.key.padding = 2,
>         ylab.cex = 1.25,
>         main.cex = 1.25,
>         yaxis.cex = 1,
>         panel.heights = c(1, 0.15, 0.15, 0.15),
>         # set the spacing between the plots so there is very little space between heat graphs
>         yspacing = c(-1, -1, -1),
>         xaxis.lab = NULL,
>         # remove axes tick marks
>         xaxis.alternating = 0,
>         yaxis.alternating = 0,
>         # set the yaxis labels (only needed on the bar plot)
>         yaxis.lab = list(NULL, NULL, NULL, seq(0, 100, 20)),
>         # put the legend we created on the right side of the multiplot
>         legend = list(right = list(fun = legends)),
>         print.new.legend = TRUE,
>         );
```

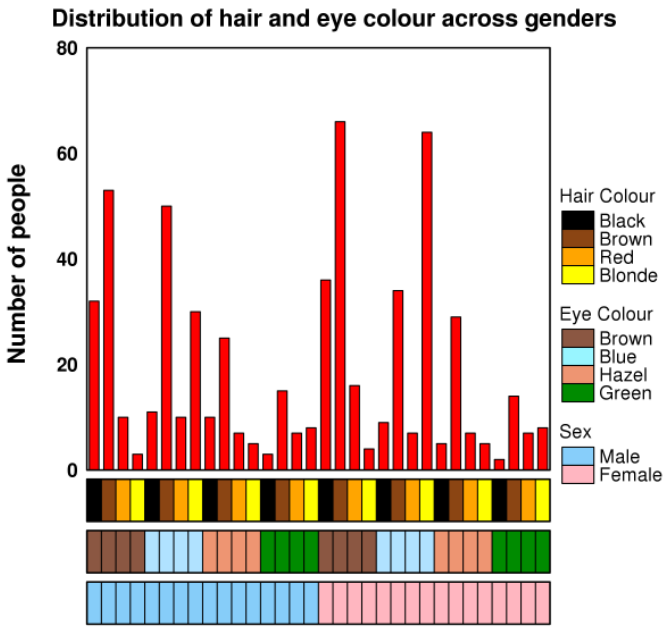

Figure 6: Finished Example 1 Plot

3.0 Example 2: Complex Layout

In this example we make a plot to convey gene expression using a more complex layout. This plot consists of two bar plots and a heatmap. The heatmap takes up most of the figure and represents gene expression level changes in different samples. The top bar plot will represent the amount of sample used, and the side bar plot represents the importance of the gene. (Note that this example is not based upon real data.)

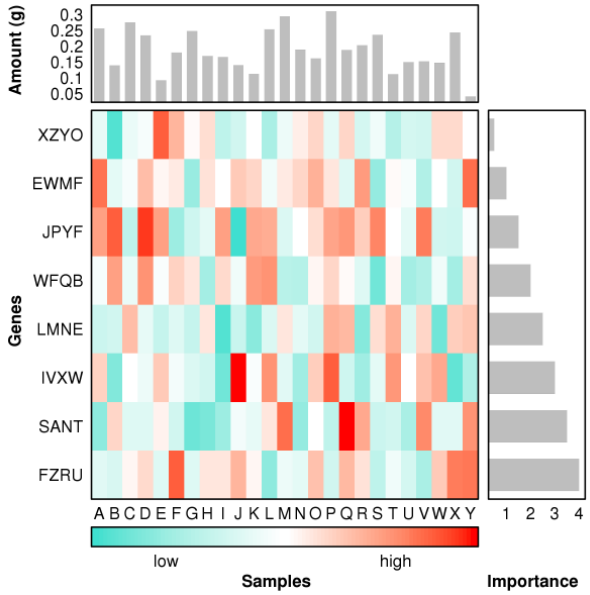

Figure 7: Finished Example 2 Plot

3.1 Setting up the data

We will generate data for this example.

```
> set.seed(12345);

> heatmap.data <- data.frame(
>   a = rnorm(n = 25, mean = 0, sd = 0.75),
>   b = rnorm(n = 25, mean = 0, sd = 0.75),
>   c = rnorm(n = 25, mean = 0, sd = 0.75),
>   d = rnorm(n = 25, mean = 0, sd = 0.75),
>   e = rnorm(n = 25, mean = 0, sd = 0.75),
>   f = rnorm(n = 25, mean = 0, sd = 0.75),
>   g = rnorm(n = 25, mean = 0, sd = 0.75),
>   h = rnorm(n = 25, mean = 0, sd = 0.75)
>   );
>
> colorkey.data <- data.frame(
>   x <- seq(-50,50,1)
>   );
>
> top.barplot.data <- data.frame(
>   x = rnorm(n = 25, mean = 2, sd = 0.75),
>   y = seq(1,25,1)
```

```
>   );
>
> side.barplot.data <- data.frame(
>   x = rnorm(n = 8, mean = 0, sd = 0.75),
>   y = seq(1,8,1)
>   );
```

3.2 Creating the main heatmap

Here we create the heatmap that will be used to display the expression level for each sample and gene.

```
> # create the main heatmap
> gene.expression.heatmap <- create.heatmap(
>     x = heatmap.data,
>     xaxis.tck = 0,
>     yaxis.tck = 0,
>     colourkey.cex = 1,
>     clustering.method = 'none',
>     axes.lwd = 1,
>     ylab.label = 'y',
>     xlab.label = 'x',
>     yaxis.fontface = 1,
>     xaxis.fontface = 1,
>     xlab.cex = 1,
>     ylab.cex = 1,
>     main.cex = 1,
>     colour.scheme = c('red','white','turquoise')
>     );
```

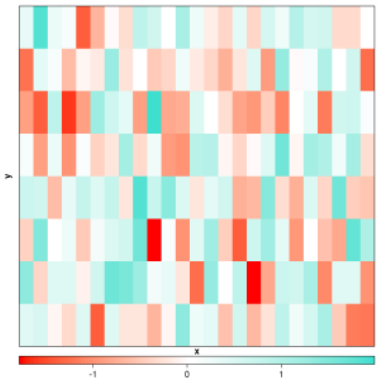

Figure 8: Main Heat Map

3.3 Creating the colourkey

Here we create the colourkey for the main heat map. It will display the same colours as the heat map, as well as indicate what the colours represent.

```
> key <- create.heatmap(
>   x = colorkey.data,
>   clustering.method = 'none',
>   scale.data = FALSE,
>   # set the same colours as are in the heatmap
```

```
>   colour.scheme = c('turquoise','white','red'),
>   print.colour.key = FALSE,
>   yaxis.tck = 0,
>   xat = c(10,90),
>   xaxis.lab = c('low', 'high'),
>   xaxis.rot = 0,
>   xaxis.cex = 2,
>   height = 1
>   );
```

Figure 9: Main Heat Map Colourkey

3.4 Creating the top bar plot

Here we create the bar plot that will appear at the top of the plot representing the amount of sample. The bar plot will look very simple, but details will be added later when the create.multiplot function is called.

```
> sample.barplot <- create.barplot(
>     formula = x~y,
>     data = top.barplot.data,
>     lwd = 0
>     );
```

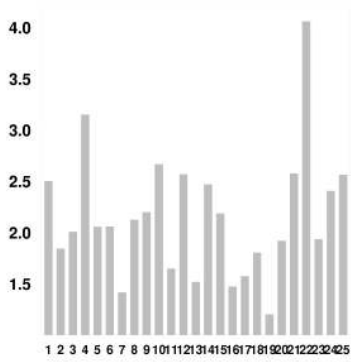

Figure 10: Amount of Sample Bar Plot

3.5 Creating the side bar plot

Here we will create the bar plot that will appear on the right side of the plot representing the importance of the gene. The bar plot will look very simple, but details will be added later when the create.multiplot function is called.

```
> importance.barplot <- create.barplot(
>     formula = x~y,
>     data = side.barplot.data,
```

```
>     lwd = 0,
>     sample.order = 'decreasing',
>     plot.horizontal = TRUE
>     );
```

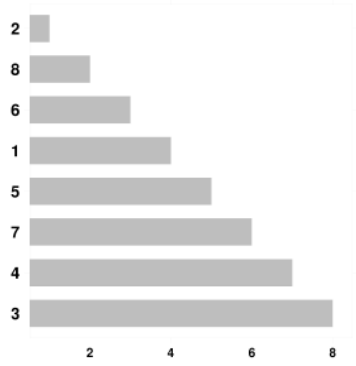

Figure 11: Importance Bar Plot

3.6 Creating the final figure

Here is where the create.multiplot function is used to combine the plots. First we need to design the layout; in this case we are choosing 3 rows of 2 columns and skipping the area used for the bottom right plot. In addition, we will assemble the plots such that the bottom row contains just the colourkey, the middle row contains the main heat map and the side bar plot, and the top row contains the top barplot. Below is a figure describing the layout; blue represents an area that a plot will appear there, red represents an area that is skipped, and yellow represents an unused area.

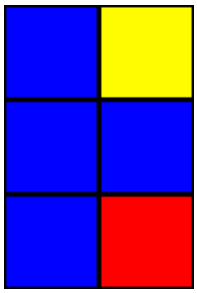

Figure 12: Plot Layout

```
> create.multiplot(
>   # use the four plots we created earlier as the objects for the multiplot
>   filename = "ex2_final_plot.png",
>   plot.objects = list(key, gene.expression.heatmap, importance.barplot, sample.barplot),
>   panel.heights = c(0.25, 1, 0.05),
>   panel.widths = c(1, 0.25),
>   # this is where we will specify how the plots layout will look
>   # plot.layout specifys the number of rows and columns (3 rows, 2 columns)
>   # layout.skip specifys which of the plots in the specification will be skipped
>   # the skip specification starts at the bottom left, moves to the right and then up
>   plot.layout = c(2, 3),
```

```
>   layout.skip = c(FALSE, TRUE, FALSE, FALSE, FALSE, FALSE),
>   xaxis.alternating = 0,
>   yaxis.alternating = 0,
>   xaxis.cex = 1,
>   yaxis.cex = 1,
>   xlab.cex = 1,
>   ylab.cex = 1,
>   # format labels (tabs are needed for proper spacing between labels)
>   xlab.label = c('\t', 'Samples', '\t', '    Importance'),
>   ylab.label = c( 'Amount (g)', '\t', '\t', 'Genes', '\t', '\t'),
>   ylab.padding = 6,
>   xlab.to.xaxis.padding = 0,
>   xaxis.lab = list(
>     c("",'low',"", "",'high', ""),
>     LETTERS[1:25],
>     seq(0,5,1),
>     NULL
>     ),
>   yaxis.lab = list(
>     NULL,
>     replicate(8, paste(sample(LETTERS, 4, replace = TRUE), collapse = "")),
>     NULL,
>     seq(0,4,0.05)
>     ),
>   xspacing = -0.5,
>   yspacing = c(0,-1),
>   xaxis.fontface = 1,
>   yaxis.fontface = 1,
>   resolution = 300
>   );
```

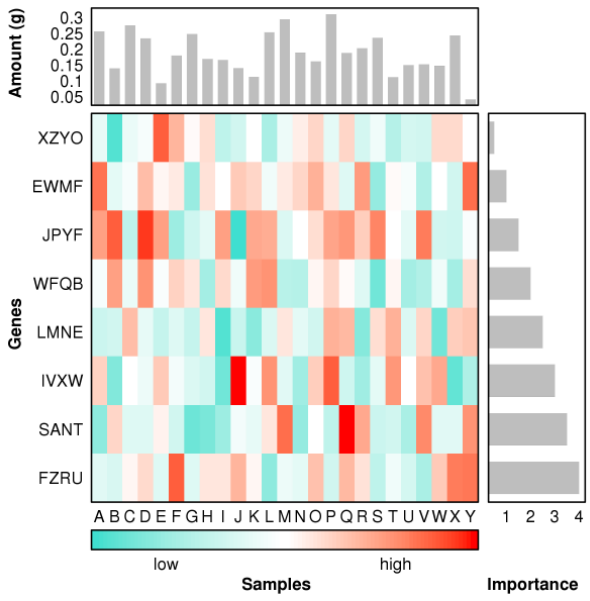

Figure 13: Finished Example 2 Plot

4.0 Example 3: Using Features Effectively

In this example we show how to take advantage of some of the customization features in create.multiplot(). The main difference between this example and last is the use of colour and other features to make the figure more aesthetically pleasing. The dataset for this example is the null distribution of the accuracy, specificity and sensitivity of samples with feature sizes of 30, 100, 300, 500 and 1000 compared to the biomarker (Sig). (Note that the data used in this example is generated.)

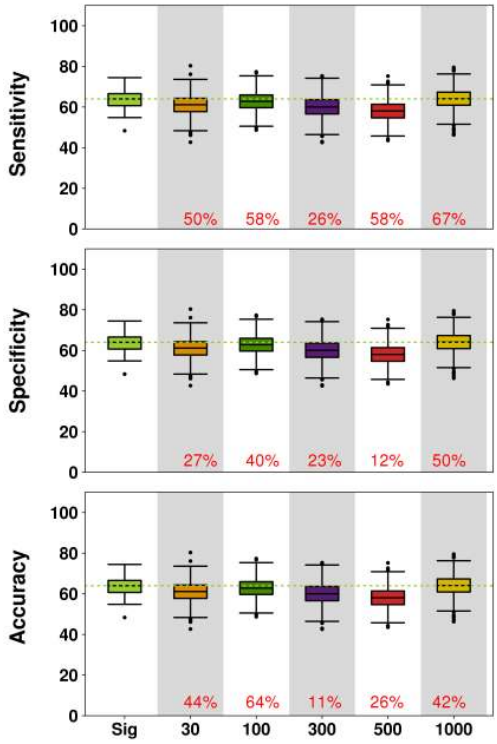

Figure 14: Finished Example 3 Plot

4.1 Set up the data

Here we set up the data used throughout this example.

```
> col.scheme <- c(default.colours(11));
>
> set.seed(12345);
> data.acc <- data.frame(
>  "30" = rnorm(1000, 0.65, 0.05),
>  "100" = rnorm(1000, 0.68, 0.05),
>  "300" = rnorm(1000, 0.60, 0.05),
>  "500" = rnorm(1000, 0.63, 0.05),
>  "1000" = rnorm(1000, 0.65, 0.05)
>  );
>
> data.sens <- data.frame(
>  "30" = rnorm(1000, 0.63, 0.05),
>  "100" = rnorm(1000, 0.64, 0.05),
>  "300" = rnorm(1000, 0.60, 0.05),
>  "500" = rnorm(1000, 0.64, 0.05),
```

```
>  "1000" = rnorm(1000, 0.65, 0.05)
>  );
>
> data.spec <- data.frame(
>  "30" = rnorm(1000, 0.61, 0.05),
>  "100" = rnorm(1000, 0.63, 0.05),
>  "300" = rnorm(1000, 0.60, 0.05),
>  "500" = rnorm(1000, 0.58, 0.05),
>  "1000" = rnorm(1000, 0.64, 0.05)
>  );
>
> colnames(data.acc) <- c('30','100','300','500','1000');
> colnames(data.sens) <- c('30','100','300','500','1000');
> colnames(data.spec) <- c('30','100','300','500','1000');
>
> performance.null <- list(
>    Accuracy = data.acc,
>   Sensitivity = data.sens,
>    Specificity = data.spec
>    );
>
> performance.marker <- data.frame(
>   Sensitivity = rnorm(100, 0.63, 0.05),
>   Specificity = rnorm(100, 0.64, 0.05),
>   Accuracy = rnorm(100, 0.65, 0.05),
>   N.probes = rep(c(30, 100, 300, 500, 1000), 20)
>   );
>
> med.perc <- as.data.frame(
>         matrix(
>                 ncol = 3,
>                 nrow = 5,
>                 dimnames = list(
>                         c('30', '100', '300', '500', '1000'),
>                         names(performance.null)
>                         )
>                 )
>         );
```

4.2 Create the tables for each sample

Here we will loop over all the performance columns and put them into one table for each sample.

```
> for (category in names(performance.null)) {
>        # obtain the current performance.null
>        this.performance   <- performance.null[[category]];
>        median.perf.marker <- median(performance.marker[,category]);
>
>        # loop over each size
>        for (size in colnames(this.performance)) {
>
>                # calculate how many of the signature falls below the median
>                this.num    <- sum(this.performance[,size] >= median.perf.marker);
>                this.perc   <- (this.num / nrow(this.performance)) * 100;
>
>                # save the result to data-frame
```

```
>                 med.perc[size, category] <- this.perc;
>
>                 }
>
>         # reformat the table for violin plot calculation
>         this.table <- data.frame(
>                 Value = c(
>                         performance.marker[,category],
>                         performance.null[[category]][,  '30'],
>                         performance.null[[category]][, '100'],
>                         performance.null[[category]][, '300'],
>                         performance.null[[category]][, '500'],
>                         performance.null[[category]][,'1000']
>                         ),
>                 FeatureSize = c(
>                         rep('Sig',  nrow(performance.marker)),
>                         rep('30',   nrow(performance.null[[category]])),
>                         rep('100',  nrow(performance.null[[category]])),
>                         rep('300',  nrow(performance.null[[category]])),
>                         rep('500',  nrow(performance.null[[category]])),
>                         rep('1000', nrow(performance.null[[category]]))
>                         ),
>                 stringsAsFactors = FALSE
>                 );
>
>       # create a dotted line across each of the box plot at the median of the biomarker
>       abline.obj <- list(
>                         value = median.perf.marker,
>                         col = if ('Sensitivity' == category) { col.scheme[6] }
>                                 else if ('Specificity' == category) { col.scheme[7] }
>                                 else if ('Accuracy' == category)    { col.scheme[11] },
>                         lwd = 3,
>                         type = 'dotted'
>                         );
>
>       # input the feature size into the table
>       this.table$FeatureSize <- factor(
>                 x = this.table$FeatureSize,
>                 levels = unique(this.table$FeatureSize)
>                 );
>
>         # introduce a checkpoint here for between columns measure
>         above.med.values <- nchar(paste(sprintf("%.0f", med.perc[,category]), '%', sep = ''));
>
>         # get different boxplot.col for different categories
>         if ('Sensitivity' == category)      { boxplot.col <- c(col.scheme[6], col.scheme[1:5]); }
>         else if ('Specificity' == category) { boxplot.col <- c(col.scheme[7], col.scheme[1:5]); }
>         else if ('Accuracy' == category)    { boxplot.col <- c(col.scheme[11], col.scheme[1:5]); }
>         }
```

4.3 Create the box plots

Here we will create the box plots that will be used to show the distribtuion of the performance, and its comparison to the biomarker.

```
> for (category in names(performance.null)) {
```

```
>         # plot boxplot plot of sensitivity, specificity, accuracy distribution
>         this.boxplot <- create.boxplot(
>                 formula = Value ~ FeatureSize,
>                 data = this.table,
>                 filename = NULL,
>                 ylimits = c(0,1.1),
>                 yat = seq(0,1,0.2),
>                 yaxis.lab = seq(0,100,20),
>                 ylab.label = category,
>                 xlab.label = "Feature Size",
>                 width = 8,
>                 fill = boxplot.col,
>                 lwd = 3,
>                 add.rectangle = TRUE,
>                 xleft.rectangle = c(1.5, 3.5, 5.5),
>                 xright.rectangle = c(2.5, 4.5, 6.5),
>                 ybottom.rectangle = 0,
>                 ytop.rectangle = 1.1,
>                 col.rectangle = 'gray1',
>                 alpha.rectangle = 0.15,
>                 # create legend that will contain the percentages
>                 legend = list(
>                     inside = list(
>                             fun = draw.key,
>                             args = list(
>                                     key = list(
>                                             text = list(
>                                                     lab = paste(
>                                                           sprintf("%.0f", med.perc[,category]),
>                                                           '%', sep = ''),
>                                                     col = 'red'
>                                                       ),
>                                             cex = 2,
>                                             columns = 5,
>                                             between.columns =
>                                                     if (sum(above.med.values == 2) > 2) {6}
>                                                     else if (sum(above.med.values == 2) < 2) {4}
>                                                     else if (sum(above.med.values == 2) == 2) {5}
>                                             )
>                                     ),
>                             x = 0.23,
>                             y = if ('Sensitivity' == category) {0.71}
>                                     else if ('Specificity' == category) {0.37}
>                                     else if ('Accuracy' == category) {0.03},
>                             corner = c(0,1),
>                             draw = FALSE
>                               )
>                     )
>                 );
>
>         if ('Sensitivity' == category) {
>                 abline.obj.sens <- abline.obj;
>                 sens.boxplot <- this.boxplot + layer(
>                         panel.abline(
>                                 h = abline.obj.sens$value,
>                                 lty = abline.obj.sens$type,
```

```
>                                 lwd = abline.obj.sens$lwd,
>                                 col = abline.obj.sens$col
>                                 )
>                         );
>                 }
>         else if ('Specificity' == category) {
>                 abline.obj.spec <- abline.obj;
>                 spec.boxplot <- this.boxplot + layer(
>                         panel.abline(
>                                 h = abline.obj.spec$value,
>                                 lty = abline.obj.spec$type,
>                                 lwd = abline.obj.spec$lwd,
>                                 col = abline.obj.spec$col
>                                 )
>                         );
>                 }
>         else if ('Accuracy' == category) {
>                 abline.obj.accu <- abline.obj;
>                 accu.boxplot <- this.boxplot + layer(
>                         panel.abline(
>                                 h = abline.obj.accu$value,
>                                 lty = abline.obj.accu$type,
>                                 lwd = abline.obj.accu$lwd,
>                                 col = abline.obj.accu$col
>                                 )
>                         );
>                 }
>         }
```

4.4 Creating the final figure

Here we combine the three boxplots from above to effectively show the data.

```
> PerformanceMultiPlot <- create.multiplot(
>         # these are the plots that will be displayed (ordered bottom to top)
>         plot.objects = list(accu.boxplot, spec.boxplot, sens.boxplot),
>
>         filename = 'ex3_final_plot.png',
>         # this is the layout of the plot (this says 3 columns and 1 row)
>         plot.layout = c(1,3),
>
>         yat = seq(0,1,0.2),
>         yaxis.lab = seq(0,100,20),
>         ylab.label = list('Sensitivity', 'Specificity', 'Accuracy'),
>         ylab.cex = 2.5,
>         height = 15,
>         width = 10,
>         yspacing = 2,
>         merge.legends = TRUE ,
>         resolution = 300
>         );
```

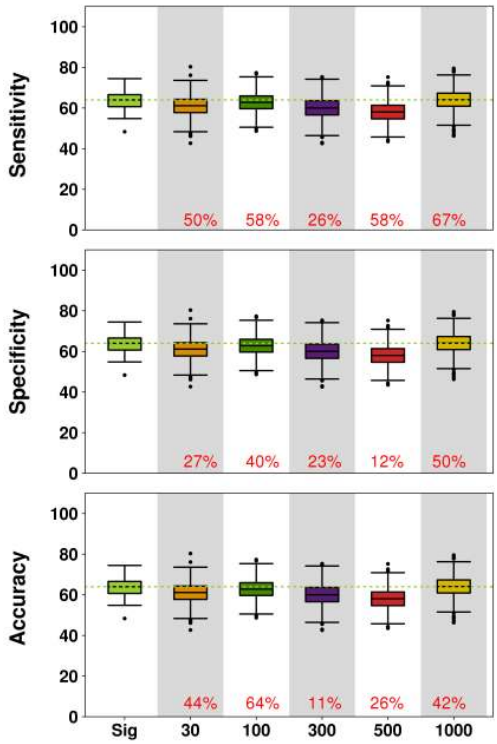

Figure 15: Finished Example 3 Plot

5.0 Troubleshooting

This section briefly outlines some issues that beginners may have with this function.

1. To begin, the way that the plots are displayed according to the layout you specify may be confusing. The first plot you specify in plot.objects is going to be drawn at the bottom left most area for a plot and the ordering will then fill the plots to the right. After reaching the rightmost plot, the plots in the row above will begin to be drawn.

2. In addition, a common mistake that is made is that the amount of panel widths/heights must be a multiple of the number of plots. The warning will look like:

```
1: In widths.x[pos.widths[[nm]]] <- widths.settings[[nm]] * widths.defaults[[nm]]$x :
number of items to replace is not a multiple of replacement length
2: In widths.x[pos.widths[["panel"]]] <- widths.settings[["panel"]] *  :
number of items to replace is not a multiple of replacement length
```

This may cause the plots to not look as you intend, but is easily fixed by ensuring that your panel.widths and panel.heights parameters are multiples of the number of plots that can be plotted. For example, if your layout has 2 rows and 3 columns, the panel.heights parameter can have 2 inputs, and the panel.widths parameter can have 3 inputs.

6.0 Session info

```
R version 3.0.2 (2013-09-25)
Platform: x86_64-unknown-linux-gnu (64-bit)
```

```
locale:
 [1] LC_CTYPE=en_CA.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_CA.UTF-8        LC_COLLATE=en_CA.UTF-8
 [5] LC_MONETARY=en_CA.UTF-8    LC_MESSAGES=en_CA.UTF-8
 [7] LC_PAPER=en_CA.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_CA.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] grid      stats     graphics  grDevices datasets  utils     methods
[8] base

other attached packages:
 [1] BoutrosLab.plotting.general_5.0.16   BoutrosLab.dist.overload_1.0.1
 [3] MASS_7.3-29                         BoutrosLab.statistics.general_2.1.1
 [5] BoutrosLab.utilities_1.9.5          hexbin_1.26.3
 [7] cluster_1.14.4                      latticeExtra_0.6-26
 [9] RColorBrewer_1.0-5                  lattice_0.20-25

loaded via a namespace (and not attached):
[1] tools_3.0.2
```