Untargeted LC-MS Metabolomics Filtering Example

Introduction

We demonstrate here how to carry out the filtering procedures described in “Filtering procedures for untargeted LC-MS metabolomics data” by Schiffman, et al. We use the CRC dataset mentioned in the manuscript for demonstration. To implement the functions within the filtering pipeline, users need…

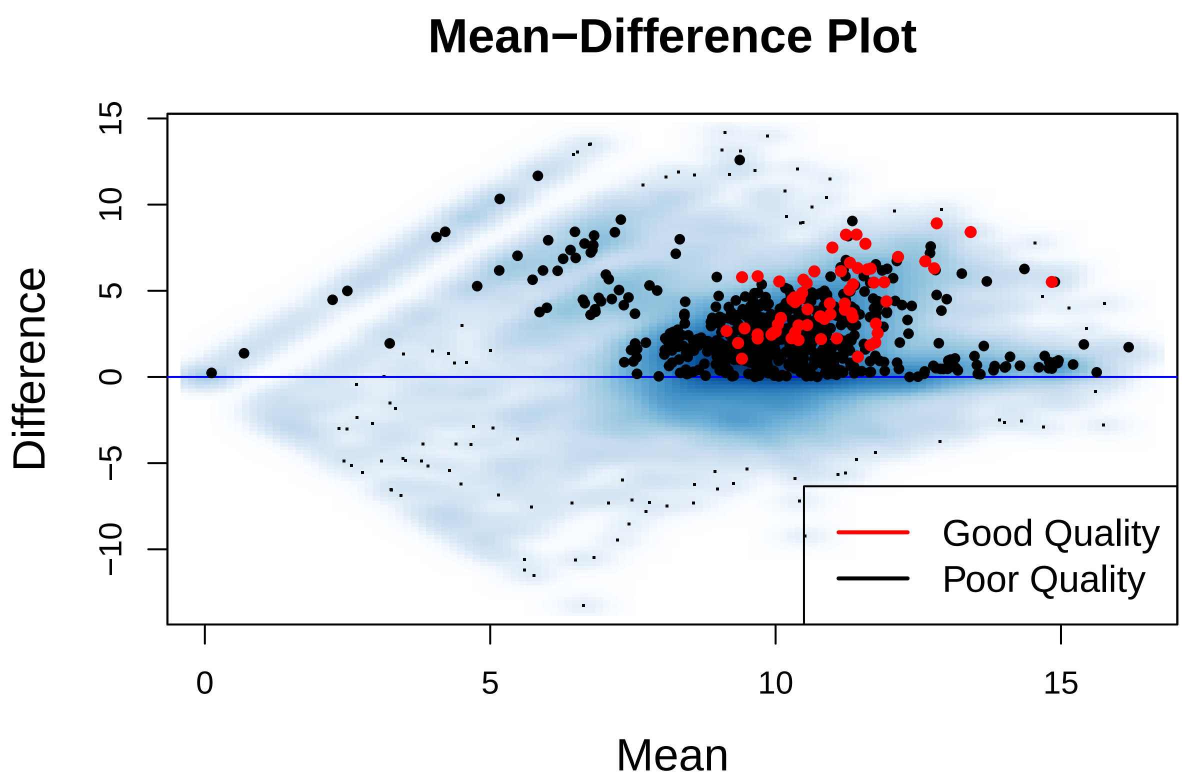
- a filled matrix (features by samples) of log feature abundances (i.e. after using the ‘fillChromPeaks’ function in xcms).
- an unfilled matrix (features by samples) of log feature abundances (i.e. before using the ‘fillChromPeaks’ function in xcms).
- names of pooled QC, blank and biological samples.
- names of identified high and low quality features.

For simplicity, unlike in the manuscript, we do not split features into a training and testing set.

MD-plot filtering

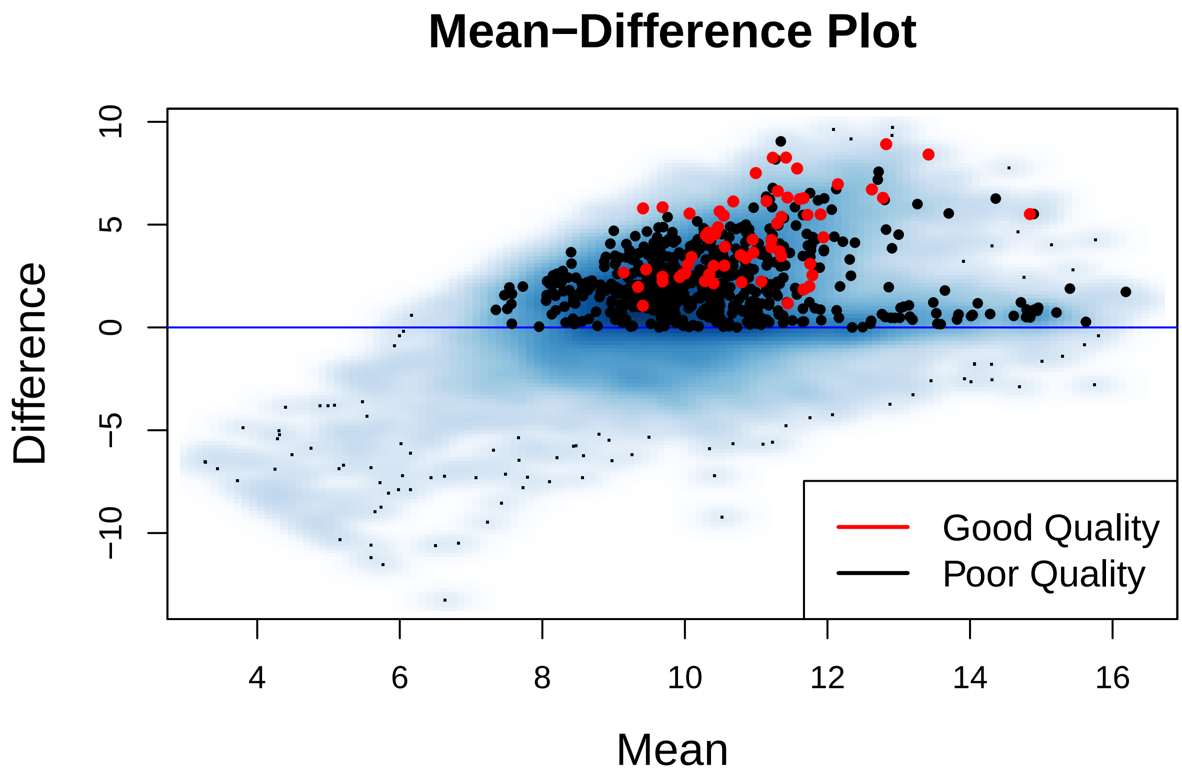
The function ‘MDplot’ creates a mean-difference plot such as the ones described in Schiffman et al., highlighting the high and low quality features within the plot. **Because the positions of the high and low quality features within the MD-plot, the number of blank samples, etc. will vary for each dataset, we only provide a function for creating the MD-plot. We do not provide a function for performing the actual filtering in this step.** However, we demonstrate how filtering based on MD-plots can be done using the CRC dataset. For the CRC dataset, all of the high quality features are in the cluster of features with the highest average abundances. We retain only features that are present in this cluster, and use the distribution of noise below the zero difference line to filter features above the zero difference line, as described in Schiffman et al.

```r
## Create the MD-plot using the 'MDplot' function

# Load the functions

source("MDplot.R")

# Load the CRC data

load("Data.RData")

MDplot(filled, blankNames, obsNames, high, low)
```



After using the ‘MDplot’ function to visualize the positions of the high and low quality features, users can identify how they should filter their features according to average abundance and differences in abundance between blank and biological samples. For example, for the CRC dataset we see that all of the high quality features are detected in all 3 blank samples (which are also the features with the highest average abundances). We again use the MDplot function, but on the subset of data corresponding to features detected in all 3 blank samples.

```
## Demonstrate plotting of thresholds and filtering for CRC
## data

# Calculate the number of blank samples (0-3) each feature
# has a zero value in. Most features are detected in all 3
# blank samples (num.zero=0) and only those features are
# reatianed

num.zero <- apply(filled[, blankNames], 1, function(x) sum(x ==
    0))
names(num.zero) <- rownames(filled)

filled <- filled[num.zero == 0, ]
unfilled <- unfilled[num.zero == 0, ]

MDplot(filled, blankNames, obsNames, high, low)
```

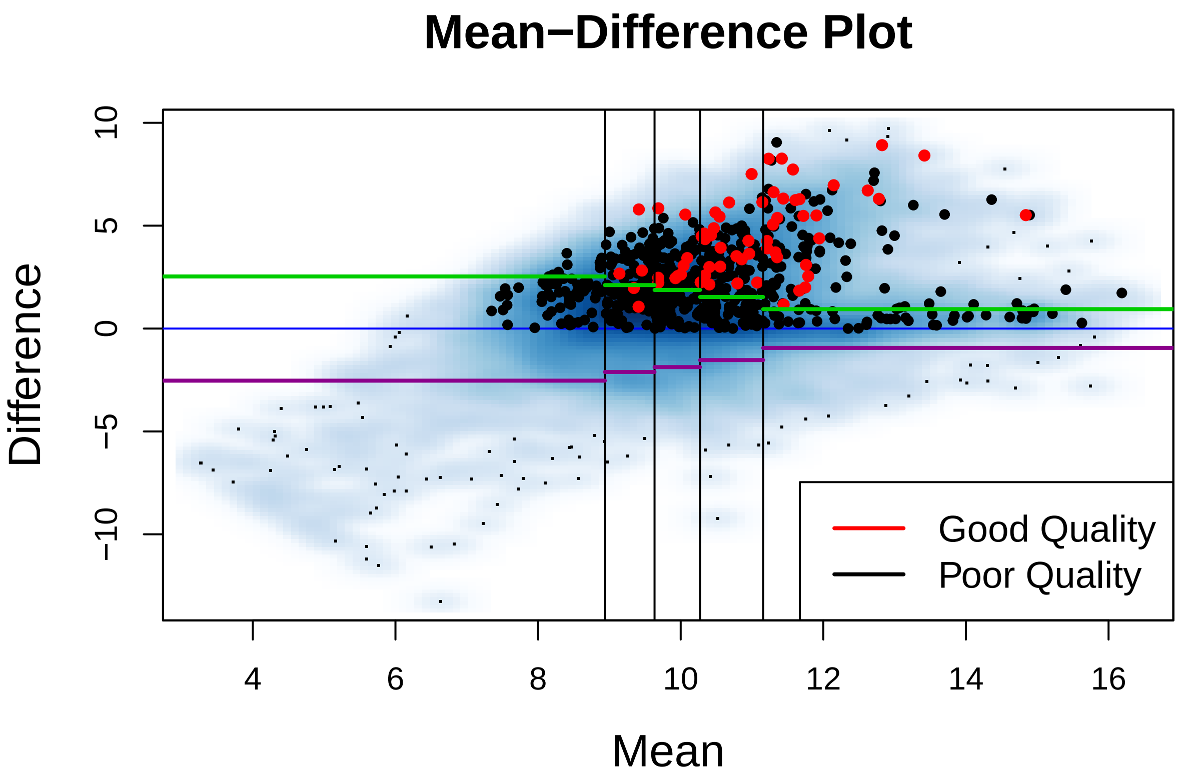


We now demonstrate how to partition the remaining features along the x-axis based on quantiles (20th,40th,60th,80th) and how to visualize and select appropriate filtering cutoffs within each partition (e.g. absolute value of the lower quartile of features below the zero difference line).

```
obsMean <- as.vector(apply(filled[, obsNames], 1, mean))
names(obsMean) <- rownames(filled)
blankMean <- apply(filled[, blankNames], 1, mean)
names(blankMean) <- rownames(filled)

# quantiles to partition the features along the x-axis
quantiles <- c(0.2, 0.4, 0.6, 0.8, 1)
breaks <- quantile((blankMean + obsMean)/2, quantiles)

# difference in average log abundances between biological and
# blank samples
diff <- (obsMean) - (blankMean)
# average log abundances in biological and blank samples
mean <- (blankMean + obsMean)/2

# find features in each partition above the absolute value of
# the lower quartile of differences below the zero-difference
# line in each partition

less1 <- diff[diff < 0 & mean <= breaks[1]]
bin1 <- rownames(filled)[diff > 0 & mean <= breaks[1] & diff >
    abs(summary(less1)[2])]

less2 <- diff[diff < 0 & mean <= breaks[2] & mean > breaks[1]]
bin2 <- rownames(filled)[diff > 0 & mean <= breaks[2] & mean >
    breaks[1] & diff > abs(summary(less2))[2]]
```

```
less3 <- diff[diff < 0 & mean <= breaks[3] & mean > breaks[2]]
bin3 <- rownames(filled)[diff > 0 & mean <= breaks[3] & mean >
    breaks[2] & diff > abs(summary(less3)[2])]

less4 <- diff[diff < 0 & mean <= breaks[4] & mean > breaks[3]]
bin4 <- rownames(filled)[diff > 0 & mean <= breaks[4] & mean >
    breaks[3] & diff > abs(summary(less4)[2])]

less5 <- diff[diff < 0 & mean > breaks[4]]
bin5 <- rownames(filled)[diff > 0 & mean > breaks[4] & diff >
    abs(summary(less5)[2])]


# plot the paritions and filtering cutoffs for this cluster

smoothScatter((blankMean + obsMean)/2, (obsMean) - (blankMean),
    xlab = "Mean", ylab = "Difference", main = "Mean-Difference Plot",
    cex.lab = 1.4, cex.main = 1.5)
abline(h = 0, lwd = 1, col = "blue")
legend("bottomright", legend = c("Good Quality", "Poor Quality"),
    col = c("red", "black"), lwd = 2, cex = 1.2)

blanks.low <- blankMean[names(blankMean) %in% low]
obs.low <- (obsMean)[names(obsMean) %in% low]
points((blanks.low + obs.low)/2, obs.low - blanks.low, pch = 19,
    cex = 0.6)
blanks.high <- blankMean[names(blankMean) %in% high]
obs.high <- obsMean[names(obsMean) %in% high]
points((blanks.high + obs.high)/2, obs.high - blanks.high, col = "red",
    pch = 19, cex = 0.7)

abline(v = breaks[1], lwd = 1)
abline(v = breaks[2], lwd = 1)
abline(v = breaks[3], lwd = 1)
abline(v = breaks[4], lwd = 1)
segments(x0 = -1, y0 = summary(less1)[2], x1 = breaks[1], col = "darkmagenta",
    lwd = 2)
segments(x0 = -1, y0 = abs(summary(less1)[2]), x1 = breaks[1],
    col = "green3", lwd = 2)
segments(x0 = breaks[1], y0 = summary(less2)[2], x1 = breaks[2],
    col = "darkmagenta", lwd = 2)
segments(x0 = breaks[1], y0 = abs(summary(less2)[2]), x1 = breaks[2],
    col = "green3", lwd = 2)
segments(x0 = breaks[2], y0 = summary(less3)[2], x1 = breaks[3],
    col = "darkmagenta", lwd = 2)
segments(x0 = breaks[2], y0 = abs(summary(less3)[2]), x1 = breaks[3],
    col = "green3", lwd = 2)
segments(x0 = breaks[3], y0 = summary(less4)[2], x1 = breaks[4],
    col = "darkmagenta", lwd = 2)
segments(x0 = breaks[3], y0 = abs(summary(less4)[2]), x1 = breaks[4],
    col = "green3", lwd = 2)
segments(x0 = breaks[4], y0 = summary(less5)[2], x1 = 20, col = "darkmagenta",
    lwd = 2)
```

```r
segments(x0 = breaks[4], y0 = abs(summary(less5)[2]), x1 = 20,
    col = "green3", lwd = 2)
```



We filter features that are below the green lines in the MD-plot.

```r
# features that are above the green lines and are retained

batch1.features <- c(bin1, bin2, bin3, bin4, bin5)


filled <- filled[rownames(filled) %in% batch1.features, ]
unfilled <- unfilled[rownames(unfilled) %in% batch1.features,
    ]
```

Percent missing filtering

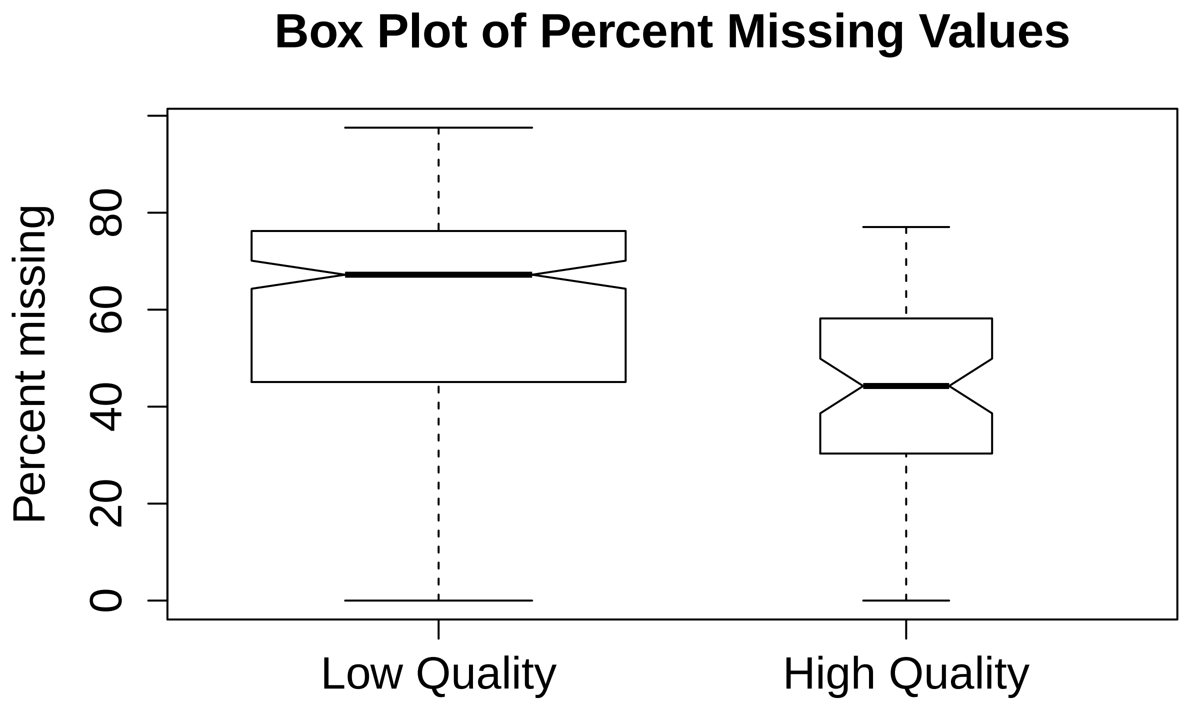
The functions 'boxplot.na' and 'density.na' allow users to visualize an appropriate cutoff for percent missing. The 'boxplot.na' function also provides users with the extreme of the lower whisker, the lower 'hinge', the median, the upper 'hinge' and the extreme of the upper whisker for each box plot to help with threshold selection.

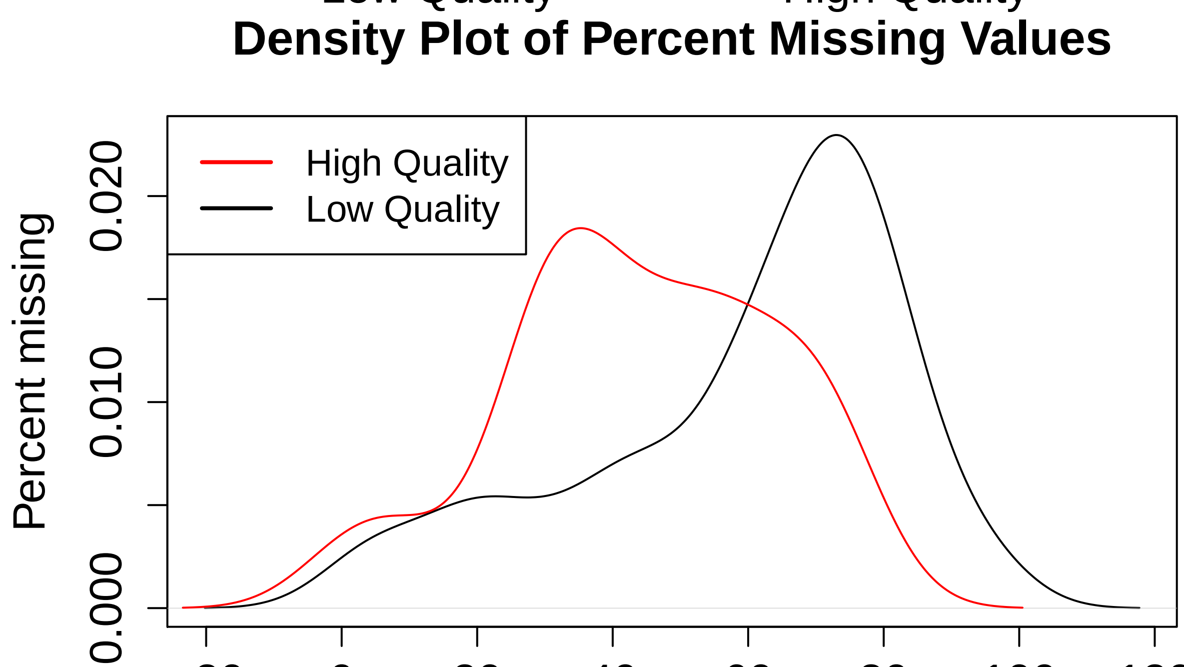
```r
source("Percent_missing.R")

# Identify a filtering threshold for percent missing

boxplot.na(unfilled, c(obsNames, obsNames2), high, low)
```

```
## $LowQuality
##          [,1]
```

```
## [1,]  0.00000
## [2,] 45.08197
## [3,] 67.21311
## [4,] 76.22951
## [5,] 97.54098
##
## $HighQuality
##          [,1]
## [1,]  0.00000
## [2,] 30.32787
## [3,] 44.26230
## [4,] 58.19672
## [5,] 77.04918
```

```r
density.na(unfilled, c(obsNames, obsNames2), high, low)
```





When filtering by percent missing, however, we do not want to remove features that are differentially missing in the phenotype of interest. Because we have a binary biological factor of interest (CRC status) we use the 'diff.miss.fish' function to test for association between missing values and the biology of interest. The 'diff.miss.fish' function calculates p-values for the Fisher exact test of the association between missing values and the binary phenotype of interest (CRC). If users have a multi-level categorical phenotype of interest, they can call the function 'diff.miss.chi' to get p-values using the Chi-square test. If users have a continuous phenotype of interest, they can call the function 'diff.miss.wilc' to get p-values using the Wilcoxon rank sum test.

```
# Calculate p-values
```

```
fish.pvals <- diff.miss.fish(unfilled, obsNames, bio)
```

Users then can specify their desired percent missing cutoff, 'na.threshold', and the percentile of the p-value distribution to use as a cutoff. The functions 'filter.na' and 'filter.pvals' identify features that pass the missing and p-value thresholds, respectively, and the union of such features is taken. Here we use the median of the missing values for the low quality features as a cutoff, and the hundredth percentile of the p-value distribution as a cutoff.

```
# Choose the missing threshold (0.67, the median of missing
# values for low quality features)

na.threshold <- 0.67

# Choose the percentile of the p-value distribution

pval.thresh <- 0.01

# Identify the features that have less than the percent
# missing threshold (using 'filter.na') OR have a p-value
# below the specified percentile.

keep.features <- unique(c(filter.na(na.threshold, unfilled, c(obsNames,
    obsNames2)), filter.pvals(pval.thresh, fish.pvals)))

# Filter data

filled <- filled[rownames(filled) %in% keep.features, ]
unfilled <- unfilled[rownames(unfilled) %in% keep.features, ]
```

ICC filtering

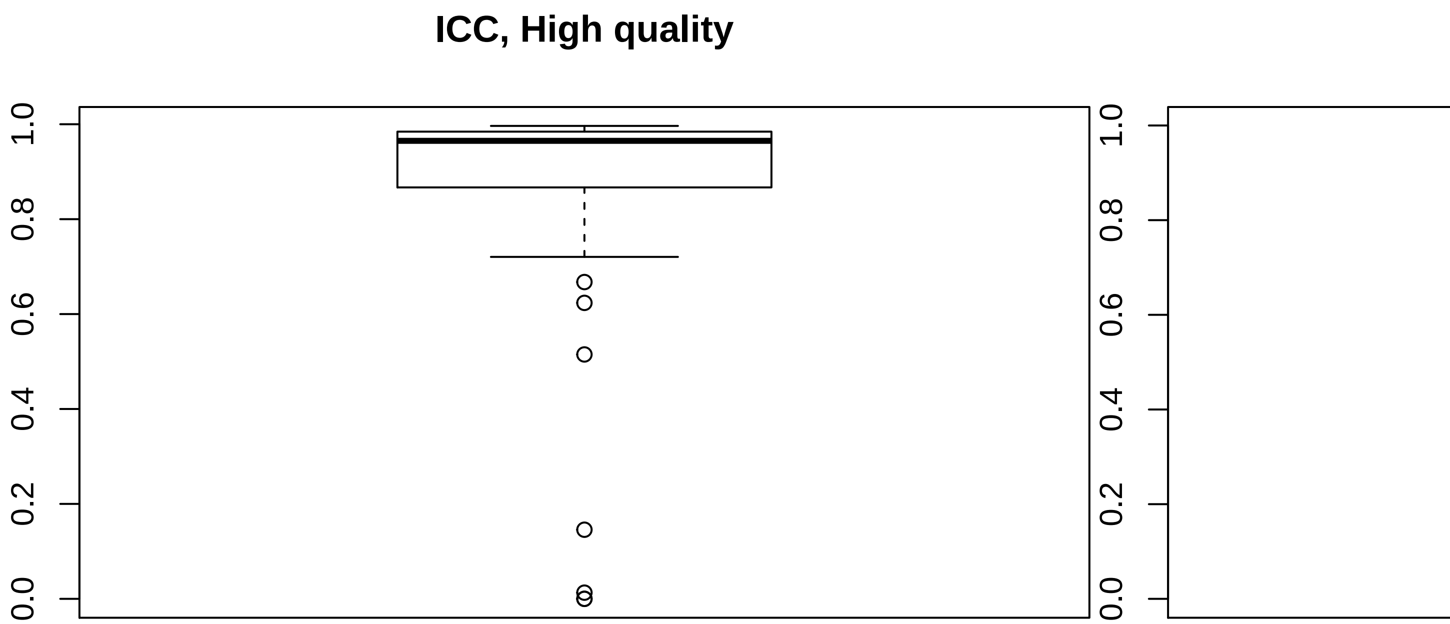
Finally, the function 'calc.icc' calculates ICC values for each feature. As with percent missing, the functions 'boxplot.icc' and 'density.icc' allow users to visualize the distribution of ICC values for high and low quality features and to select a cutoff.

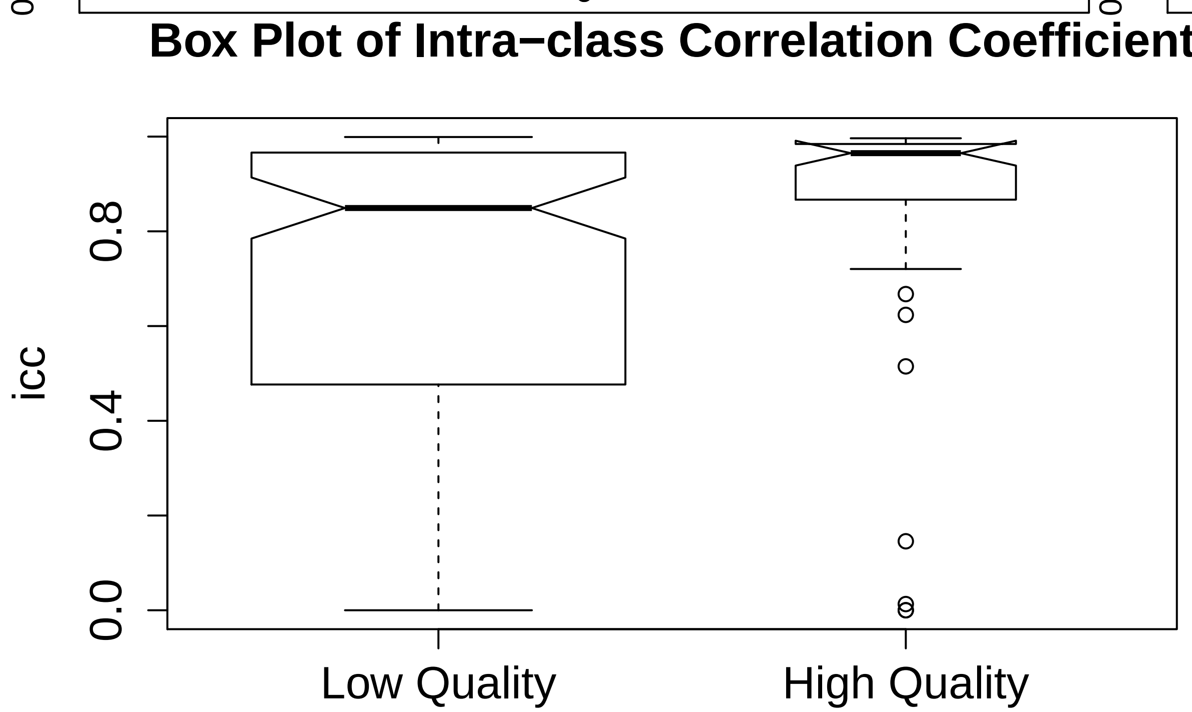
```
source("ICC.R")
registerDoParallel(cores = detectCores())

# Calculate ICC values

ICC <- calc.icc(filled, obsNames, qcNames)

# Visualize appropriate cutoffs

boxplot.icc(ICC, high, low)
```

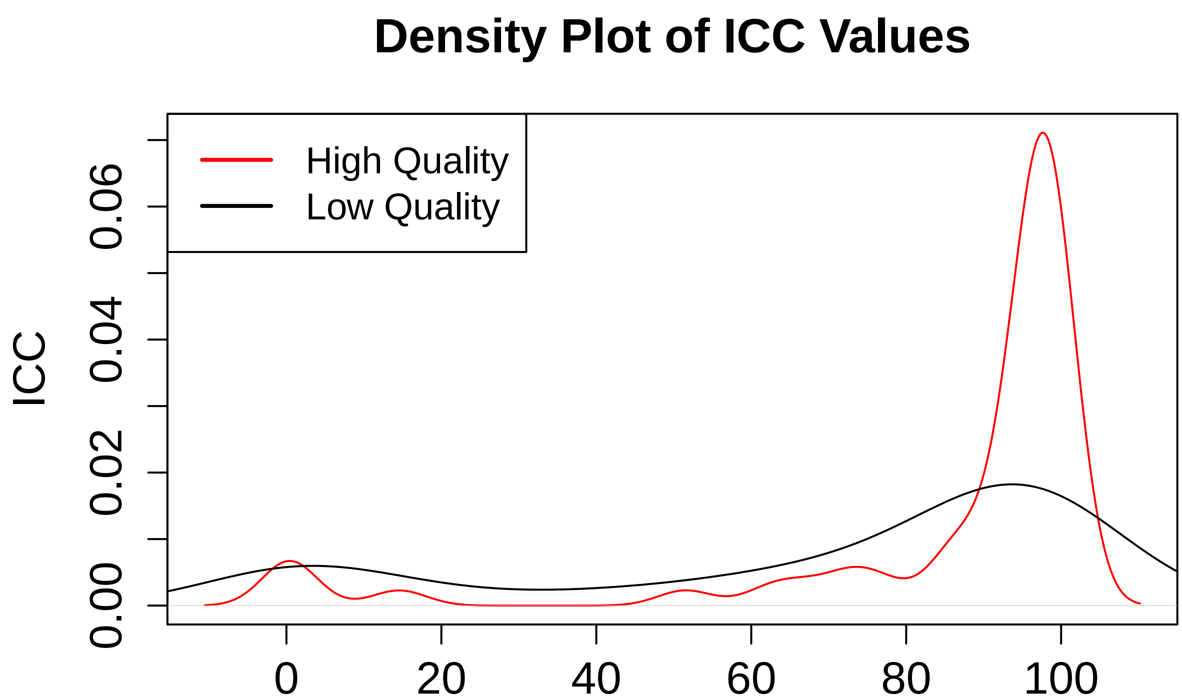
```
## $LowQuality
##                [,1]
## [1,] 3.430397e-10
## [2,] 4.766144e-01
## [3,] 8.492017e-01
## [4,] 9.662846e-01
## [5,] 9.991053e-01
```

```
##
## $HighQuality
##            [,1]
## [1,] 0.7204488
## [2,] 0.8668857
## [3,] 0.9650184
## [4,] 0.9844072
## [5,] 0.9964607
```

```
density.icc(ICC, high, low)
```







The function 'filter.icc' identifies features that pass a specified ICC threshold. We subset the data to just those features with ICC values above the lower whisker of the boxplot for high quality features, giving a final filtered dataset.

```
# Set the ICC threshold (0.72 lower whisker of high quality
# feature box plot)

threshold <- 0.72

# Use function 'filter.icc' to identify features that pass
# the threshold

filled <- filled[rownames(filled) %in% filter.icc(ICC, threshold),
    ]
unfilled <- unfilled[rownames(unfilled) %in% filter.icc(ICC,
    threshold), ]
```