Pre-processing of Adenocarcinoma data: outlier removal and collapsing of probe-level data to gene-level data

Peter Langfelder and Steve Horvath

June 19, 2012

Contents
1 Overview 1
2 Setting up the R session 1
3 Preparing gene annotation 2
4 Reading of expression data and restriction to Adenocarcinoma samples 4
5 Log-transformation of Affymetrix data and rudimentary cleaning 6
6 Outlier removal 7
7 Collapsing probe-level data to gene-level data 10

1 Overview

This document shows our preprocessing steps that we used to prepare 8 lung cancer expression data sets \[4, 1, 3, 5, 6\] for consensus module analysis. The actual consensus module analysis is described in a separate document. As input we use publicly available data sets from GEO as well as published data not stored on GEO. The entire multi-data set is available from our web page, but we encourage readers interested in the data to contact the authors of the original articles referenced above.

We encourage readers unfamiliar with any\(^1\) of the functions used in this tutorial to open an R session and type

```
help(functionName)
```

(replace `functionName` with the actual name of the function) to get a detailed description of what the functions does, what the input arguments mean, and what is the output.

2 Setting up the R session

After starting R we execute a few commands to set the working directory and load the requisite packages:

```r
# Display the current working directory
getwd();
# If necessary, change the path below to the directory where the data files are stored.
# "." means current directory. On Windows use a forward slash / instead of the usual \.
workingDir = ".";
setwd(workingDir);
# Load the package
```

\(^1\)At this point several of the functions used in this tutorial are not part of the package and hence not documented yet.
library(WGCNA);
# Allow multi-threading in WGCNA
allowWGCNAThreads();
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
# Load custom functions that simplify handling of multi-data sets
source("networkFunctions-extras-05.R");

3 Preparing gene annotation

We begin by identifying directories and file names. The directories here reflect the way the data is stored on our workstations; please adapt the directories as needed if re-running the code.

# Base of the data tree
dataDir = "../../Data-LungCancer8Sets/";
# Next level is split by data set
setDir = c("LungCancer-Shedden/MICH/",
"LungCancer-Shedden/HLM/",
"LungCancer-Shedden/DFCI/",
"LungCancer-Shedden/MSKCC/",
"LungCancer-Bild-GSE03141/",
"LungCancer-Tomida-GSE13213/",
"LungCancer-Taekuchi-GSE11969/",
"LungCancer-Roepman/")
# Under each data set, the relevant data set is stored in this subdirectory
eexprDirNormalized = "Expression/020-normalized/"
# Actual data files
eexprFiles = spaste(dataDir,
setDir,
eexprDirNormalized,
c("GeneExpressionMICH.csv.bz2",
"GeneExpressionHLM.csv.bz2",
"GeneExpressionDFCI.csv.bz2",
"GeneExpressionMSKCC.csv.bz2",
"GeneExpressionGSE3141.csv.bz2",
"GeneExpressionTomida.csv.bz2",
"GeneExpressionTakeuchi.csv.bz2",
"GeneExpressionRoepman.csv.bz2");

Next we define file names for annotation files.

# Directory containing microarray platform annotation files
annotDir = spaste(dataDir, c("LungCancer-Shedden/", spaste(setDir[5:8], "Expression/")),
"ArrayAnnotation/Shortened/");
# Actual annotation files
annotFiles = spaste(dataDir, c("LungCancer-Shedden/", spaste(setDir[5:8], "Expression/")),
"ArrayAnnotation/",
c("Shortened/ShortGeneSymbolHG133A.csv.bz2",
"Shortened/ShortGeneSymbolHG133plus2.csv.bz2",
"Raw-Large/LongTomidaGPL6480-26599.txt.bz2",
"Raw-Large/GeneSymbolTakeuchi.csv.bz2",
"Shortened/Roepman_ArrayAnnotation.txt.bz2");
# Index indicating which annotation files is to be used for each data set
annotInd = c(1,1,1,1,2,3,4,5);
nAnnot = length(annotFiles)
# Set names for pretty-printing
setNames = c("Shedden-MICH",
"Shedden-HLM",
"Shedden-DFCI",
"Shedden-MSKCC",
"Bild-GSE03141",
"Tomida-GSE13213",
"Taekuchi-GSE11969",
"Roepman/")
We now read in manufacturer's annotation files

```r
manAnnotFiles = spaste(annotDir,
  c("HG-U133A.na31.annot-shortened.txt.bz2",
    "HG-U133_Plus_2.na31.annot-shortened.txt.bz2",
    "Agilent014850_D_AA_20070207-Tomida-shortened.txt.bz2",
    "AgilentCustomHGArray-Takeuchi.csv.bz2",
    "Roepman_ArrayAnnotation.txt.bz2"));
type = c(1,1,1,2,1);
mannot = list();
for (a in 1:nAnnot)
{
  if (type[a]==2)
  {
    mannot[a] = read.csv(bzfile(manAnnotFiles[a]), header = TRUE, colClasses = "character");
  } else
    mannot[a] = read.delim(bzfile(manAnnotFiles[a]), header = TRUE, comment.char = ",",
                          colClasses = "character");
}
# Quick look at what is included:
lapply(mannot, colnames)
lapply(mannot, dim)
```

Since the various manufacturers do not always provide the Entrez gene ID which we use in combining probe level data to gene level data, we submitted the available identifiers to the DAVID online database [2] for conversion to Entrez identifiers. The conversion results are contained in files listed below.

```r
idConversionFiles = spaste(annotDir,
  c("Entrez-GBlist-HG133A-convertedByDavid.txt.bz2",
    "Entrez-GBlist-HG133plus2-convertedByDavid.txt.bz2",
    "Entrez-GBlist-Tomida-convertedByDavid.txt.bz2",
    "Entrez-GBlist-Takeuchi-ConvertedByDavid.txt.bz2",
    "Entrez-HSlist-Roepman-convertedByDavid.txt.bz2"));

idConv = list();
# Read in the conversion files
for (a in 1:nAnnot)
{
  idConv[a] = read.delim(bzfile(idConversionFiles[a]), header = TRUE, colClasses = "character");
}
# Remove the superfluous "0" identifiers from Roepman
a = 5
idConv[a] = idConv[a] [ idConv[a][, 1]!="0", ];
# We actually need only the last two id conversions.
mannot.ext = mannot;
a = 4;
table(is.finite(match(idConv[a]$From, mannot[[a]]$GB_LIST)))
mannot.ext[[a]] = merge(mannot[[a]], idConv[[a]], by.x = "GB_LIST", by.y = "From");
colnames(mannot.ext[[a]])[colnames(mannot.ext[[a]])=="To"] = "Entrez";
a = 5;
mannot[[a]]$GeneID = toupper(mannot[[a]]$GeneID);
idConv[[a]]$From = toupper(idConv[[a]]$From);
table(is.finite(match(idConv[[a]]$From, mannot[[a]]$GeneID)))
```
We next determine the number of common genes.

```r
trySets = c(1,2,3,4,5)
for (a in trySets)
{
  if (a==trySets[1])
  {
    common = unique(mannot.ext[[a]][, entrezCol[a]])
  } else
  common = unique(intersect(common, mannot.ext[[a]][, entrezCol[a]]));
  printFlush(paste("After set", a, " have", length(common), "common genes.");
}
```

The results is that there are roughly 9500 common genes. We save the extended annotation files for future use (only for annotations 4 and 5).

```r
extAnnotFiles = spaste(annotDir, 
c(  "HG-U133A.na31.annot-shortened.txt.bz2", 
  "HG-U133_Plus_2.na31.annot-shortened.txt.bz2", 
  "Agilent014850_D_AA_20070207-Tomida-shortened.txt.bz2", 
  "AgilentCustomHGArray-Takeuchi-extendedByDAVID.txt.bz2", 
  "Roepman_ArrayAnnotation-extendedByDAVID.txt.bz2"));
for (a in 4:5)
{
  write.table(mannot.ext[[a]], file= bzfile(extAnnotFiles[[a]]), 
              sep = "\t", quote = FALSE, row.names = FALSE);
}
```

## 4 Reading of expression data and restriction to Adenocarcinoma samples

Here we read in the expression data and sample annotation, and restrict the data to adenocarcinoma samples.

```r
nSets = length(exprFiles);
expr0 = list();
for (set in 1:nSets)
{
  printFlush(paste("Set", exprFiles[set]));
  data = read.csv(bzfile(exprFiles[set]), header = TRUE, comment.char = "#");
  expr0[[set]] = data;
  collectGarbage()
}
# Check that the dimensions make sense
lapply(expr0, dim)
```

We next transpose the data and fix probe names that have an ‘X’ prepended to them by R.

```r
expr = list();
probes0 = list();
for (set in 1:nSets)
{
  expr[[set]] = t(expr0[[set]][, -1]);
  colnames(expr[[set]]) = expr0[[set]][, 1];
  probes0[[set]] = expr0[[set]][, 1];
}
collectGarbage();

# Special processing of probe names...
probes = probes0;
for (set in 1:4)
  probes[[set]] = substring(probes0[[set]], 2);

We now check overlap of probe IDs in expression data and in annotation data.

probeCol = c(1, 1, 1, 3, 2);
for (set in 1:nSets)
{
  printFlush(paste("Set", exprFiles[set]));
  a = annotInd[set]
  expr2annot = match(probes[[set]], mannot.ext[[ a ]][, probeCol[a ]]);
  print( table(is.finite(expr2annot)))
  if (sum(is.finite(expr2annot)) < 1000) print("Something is wrong.");
}

Next we load the sample annotations to restrict the data to adenocarcinoma samples only.

sampleFiles = spaste(dataDir, setDir, "SampleAnnotation/",
c("SampleAnnotationMICH.csv",
  "SampleAnnotationHLM.csv",
  "SampleAnnotationDFCI.csv",
  "SampleAnnotationMSKCC.csv",
  "SampleAnnotationGSE3141.csv",
  "SampleAnnotationTomida.csv",
  "SampleAnnotationTakeuchi.csv",
  "SampleAnnotationRoepman.csv"));
sampleAnnot = list();
# Read in sample annotations
for (set in 1:nSets)
{
  sampleAnnot[[set]] = read.csv(file = sampleFiles[set], header = TRUE, comment.char = "#");
}
# Check that the results makes sense
lapply(sampleAnnot, colnames)
# The sample ID is always the first column
sampleIDcol = rep(1, nSets);
# Check that sample IDs in expression and sample annotation agree
for (set in 1:nSets)
{
  common = intersect(sampleAnnot[[set]][, sampleIDcol[set] ], rownames(expr[[set]]))
  print(data.frame(common = length(common), annot = nrow(sampleAnnot[[set]]), expr = nrow(expr[[set]])));}

We now remove all non-adenocarcinoma samples.

colnames = lapply(sampleAnnot, colnames);
match1 = function(haystack, needle, ...) {match(needle, haystack, ...)
histoCol = sapply(lapply(colnames, match1, c("Histology", "CellType")), I);
histoCol = apply(histoCol, 2, max, na.rm = TRUE)
histoCol[!is.finite(histoCol)] = NA
keepSamples = list(); # refers to rows of expr
for (set in 1:nSets) {
  if (is.na(histoCol[[set]])) {
    keepSamples[[set]] = rep(TRUE, nrow(expr[[set]]));
  } else {
    expr2annot = match(rownames(expr[[set]]), sampleAnnot[[set]][, sampleIDcol[[set]]]);
    type = sampleAnnot[[set]][ expr2annot, histoCol[[set]] ];
    levels = sort(unique(type));
    if (is.finite(match("AD", levels))) {
      keepSamples[[set]] = type=='AD';
    } else if (is.finite(match("A", levels))) {
      keepSamples[[set]] = type=='A';
    } else printFlush(paste("Problem in set", set))
    expr[[set]] = expr[[set]][keepSamples[[set]], ];
  }
}

5 Log-transformation of Affymetrix data and rudimentary cleaning

In the next step we log-transform all Affymetrix data sets, that is, sets 1–5.

```r
# Log-transform the affy sets (1--5)
lexpr = expr;
for (set in 1:5) {
  lexpr[[set]][ lexpr[[set]] < 1 ] = 1;
  lexpr[[set]] = log2(lexpr[[set]]);
}
collectGarbage();
```

We now remove bad probes and samples, defined as probes/samples with too many missing entries.

```r
lprobes = probes;
for (set in 1:nSets) {
  printFlush(paste("Set: ", set))
  gsg = goodSamplesGenes(lexpr[[set]], verbose = 2)
  if (!gsg$sallOK) {
    lexpr[[set]] = lexpr[[set]][ gsg$goodSamples, gsg$goodGenes ];
    lprobes[[set]] = probes[[set]][gsg$goodGenes];
  }
}
collectGarbage();
for (set in 1:nSets)
  colnames(lexpr[[set]]) = lprobes[[set]];
```

At this point we save the data.

```r
exprDirLogCleaned = spaste(dataDir, setDir, "Expression/023-Logged-Cleaned");
exprFiles.Log.Clean = spaste(
  exprDirLogCleaned, "/",
  c("GeneExpr-MICH-Log-Clean.csv.bz2",
   "GeneExpr-HLM-Log-Clean.csv.bz2",
   "GeneExpr-DFCI-Log-Clean.csv.bz2",
   "GeneExpr-MSKCC-Log-Clean.csv.bz2"));
```
6 Outlier removal

We remove outliers using a two-stage process. The basic idea is to identify outliers as those samples whose connectivity in a sample network is low compared to the mean connectivity. At the same time, we would like to ensure that the data sets are relatively homogenous, i.e., the samples in each data set do not form several well-separated clusters. Hence, we first identify large, well-defined clusters in each data set; remove outliers from each cluster; combine clusters by scaling each gene to the same median value across the clusters; and finally remove outliers from the combined data set. An outlier is defined as a sample whose connectivity scaled to mean 0 and variance 1, $Z_i$, is below a threshold $Z_{\text{min}}$. The threshold is defined heuristically and depends on the number of samples $m$ (in each cluster or the full data set) as

$$Z_{\text{min}} = \min(-1.5, -1.25 \times \log_{10} n)$$

(1)

This functional form means that $Z_{\text{min}} = -1.5$ if the number of samples is below 16, then rises slowly (logarithmically) with the number of samples $n$. The aim is to make the outlier removal more sensitive for small sample sizes since it is less likely to observe a very low standardized connectivity $Z_i$ by chance in smaller numbers of samples.

We start by calculating clustering trees of samples in each set. The trees determine the large clusters necessary for outlier removal, and facilitate visualization of which samples were removed.

```r
dists = list();
trees = list();
for (set in 1:nSets)
{
  printFlush(paste("Set: ", set));
  dists[[set]] = dist(leexpr[[set]]);
  trees[[set]] = hclust(dists[[set]], method = "ward.d2");
}
# Find sample clusters. Note deepSplit=0 in the tree cut to get large, well-separated clusters.
sampleLabels = list();
for (set in 1:nSets)
{
  sampleLabels[[set]] = cutreeDynamic(trees[[set]], distM = as.matrix(dists[[set]]),
                                      cutHeight = 10000, deepSplit = 0, minClusterSize = 5);
}
collectGarbage();
```

The outlier removal is coded in a separate set of functions that we “source” into the R session.

```r
source("outlierRemovalFunctions.R");
lexpr.or = list();
batchOutliers = list();
allOutliers = list();
```
for (set in 1:nSets)
{
    printFlush(paste("Set: ", set));
    x = removeOutliersAndBatches(lexpr[[set]], sampleLabels[[set]], indent = 2);
    lexpr.or[[set]] = x$expr;
    batchOutliers[[set]] = x$batchOutliers;
    allOutliers[[set]] = x$allOutliers;
    collectGarbage();
}

We save the cleaned data both as an R object as well as plain text tables.

save(lexpr.or, batchOutliers, allOutliers, trees, sampleLabels,
    meanIAD, file = "Data-Intermediates/outlierRemoval-lexpr.or.RData");

# Save data one more time.
# the absence of the trailing slash should make file.exists compatible with Windows
exprDir.OR = spaste(dataDir, setDir, "Expression/030-OutliersRemoved");
exprFiles.OR = spaste(
  exprDir.OR, "/",
  c("GeneExpr-MICH-OR.csv.bz2",
   "GeneExpr-HLM-OR.csv.bz2",
   "GeneExpr-DFCI-OR.csv.bz2",
   "GeneExpr-MSKCC-OR.csv.bz2",
   "GeneExpr-GSE3141-AdenoOnly-OR.csv.bz2",
   "GeneExpr-Tomida-OR.csv.bz2",
   "GeneExpr-Takeuchi-AdenoOnly-OR.csv.bz2",
   "GeneExpr-Roepman-AdenoOnly-OR.csv.bz2"));

for (set in 1:nSets)
{
    if (!file.exists(exprDir.OR[[set]]))
    {
        printFlush(paste("Creating directory", exprDir.OR[[set]]));
        dir.create(exprDir.OR[[set]], recursive = FALSE);
    }
    write.csv(cbind(SampleID = rownames(lexpr.or[[set]]), lexpr.or[[set]]), file = bzfile(exprFiles.OR[[set]]),
        quote = FALSE, row.names = FALSE);
}

We now plot the sample clustering trees together with indicators of sample clusters and outlier status.

sizeGrWindow(12,9);
#pdf(file="Plots/sampleClustering-goodSamplesAndGenes-log-EuclideanDist-2.pdf", wi=12, he=9);
layout(matrix( c(1,2, 7,8, 13,14, 3,4, 9,10, 15,16, 5,6, 11,12,17,18), 6,3),
    heights = c(0.8, 0.2, 0.8, 0.2, 0.8, 0.2));
for (set in 1:nSets)
{
    max = max(abs(meanIAD[[set]]));
    plotDendroAndColors(trees[[set]],
        cbind(sampleLabels[[set]],
            as.numeric(batchOutliers[[set]])*2 , as.numeric(allOutliers[[set]])*2 ),
        c("Clusters", "b.outliers", "all.outliers"),
        dendroLabels = FALSE, marAll = c(0.2, 4, 2, 0.2), main = setNames[set],
        setLayout = FALSE);
}
dev.off();
Figure 1: Sample clustering trees together with indicators of sample clusters and outlier status.

The result is shown in Figure 1. The sample trees in some of the data sets show the presence (before cluster adjustment) of clear separate clusters.
7 Collapsing probe-level data to gene-level data

We next “collapse” the probe-level data in each set to gene level. This will allow us to compare the data sets directly (after restricting to a common set of genes). We use the function `collapseRows`.

```r
method = "absMaxMean"
geneExpr = list();
representatives= list();
collectGarbage();
for (set in 1:nSets)
{
  printFlush(paste("Set", exprFiles[[set]]));
  a = annotInd[[set]]
  expr2annot = match(lprobes[[set]], mannot.ext[[ a ]][, probeCol[ a ]]);
  fin = is.finite(expr2annot);
  tmp.expr = lexpr.or[[set]][, fin];
  tmp.probes = lprobes[[set]][fin];
  expr2annot = expr2annot[fin];
  entrez1 = mannot.ext[[a]][ expr2annot, entrezCol[ a ]];
  print(table(!is.na(entrez1)))
  colnames(tmp.expr) = tmp.probes;
  print(system.time( {cr = collapseRows(t(tmp.expr), rowGroup = entrez1,
                     rowID = tmp.probes, method = method);}) );
  geneExpr[[set]] = t(cr$datETcollapsed);
  representatives[[set]] = cr$group2row;
  collectGarbage();
}
save(geneExpr, representatives,
     file = spaste(dataDir, "MultiData-8Sets/042-geneExpr-RObject/", "collapse-", method, "-geneExpr-representatives.RData"));
```

We also save the collapsed data in plain-text format.

```r
exprDir.collapsedToGenes = spaste(dataDir, setDir, "Expression/040-CollapsedToGenes");
exprFiles.collapsedToGenes = spaste( exprDir.collapsedToGenes, "/",
c("GeneExpr-MICH-collapsedToGenes.csv.bz2",
"GeneExpr-HLM-collapsedToGenes.csv.bz2",
"GeneExpr-DFCI-collapsedToGenes.csv.bz2",
"GeneExpr-MSKCC-collapsedToGenes.csv.bz2",
"GeneExpr-GSE3141-AdenoOnly-collapsedToGenes.csv.bz2",
"GeneExpr-Tomida-collapsedToGenes.csv.bz2",
"GeneExpr-Takeuchi-AdenoOnly-collapsedToGenes.csv.bz2",
"GeneExpr-Roepman-AdenoOnly-collapsedToGenes.csv.bz2"));
for (set in 1:nSets)
{
  if (!file.exists(exprDir.collapsedToGenes[[set]])
  {
    printFlush(paste("Creating directory", exprDir.collapsedToGenes[[set]]));
    dir.create(exprDir.collapsedToGenes[[set]], recursive = FALSE);
  }
  write.csv(cbind(SampleID = rownames(geneExpr[[set]]), geneExpr[[set]]),
            file = bzfile(exprFiles.collapsedToGenes[[set]]),
            quote = FALSE, row.names = FALSE);
}
```

The preprocessing ends here.
References


