# Untitled

*Richard Chen*

*August 21, 2015*

Correlation networks are increasingly being used in bioinformatics applications. For example, weighted gene co-expression network analysis is a systems biology method for describing the correlation patterns among genes across microarray samples. Weighted correlation network analysis (WGCNA) can be used for finding clusters (modules) of highly correlated genes, for summarizing such clusters using the module eigengene or an intramodular hub gene, for relating modules to one another and to external sample traits (using eigengene network methodology), and for calculating module membership measures. Correlation networks facilitate network based gene screening methods that can be used to identify candidate biomarkers or therapeutic targets. These methods have been successfully applied in various biological contexts, e.g. cancer, mouse genetics, yeast genetics, and analysis of brain imaging data. While parts of the correlation network methodology have been described in separate publications, there is a need to provide a user-friendly, comprehensive, and consistent software implementation and an accompanying tutorial.

The WGCNA R software package is a comprehensive collection of R functions for performing various aspects of weighted correlation network analysis. The package includes functions for network construction, module detection, gene selection, calculations of topological properties, data simulation, visualization, and interfacing with external software. While the methods development was motivated by gene expression data, the underlying data mining approach can be applied to a variety of different settings.

## 0. Automatic Installation from CRAN

The WGCNA package is now available from the Comprehensive R Archive Network (CRAN), the standard repository for R add-on packages. Currently, one of the required packages is only available from Bioconductor and needs to be installed separately. To install the required packages and WGCNA, simply type

```r
source("http://bioconductor.org/biocLite.R")
biocLite(c("AnnotationDbi", "impute", "GO.db", "preprocessCore", "org.Hs.eg.db", "impute". "WGCNA"))
```

Then, we load our dependencies.

```r
library(WGCNA)
```

```
## Loading required package: dynamicTreeCut
## Loading required package: fastcluster
##
## Attaching package: 'fastcluster'
##
## The following object is masked from 'package:stats':
##
##     hclust
##
## Creating a generic function for 'nchar' from package 'base' in package 'S4Vectors'
## Loading required package: DBI
##
##
## Attaching package: 'WGCNA'
```

```
##
## The following object is masked from 'package:stats':
##
##     cor
```

```
library(ggplot2)
library(ggdendro)
options(stringsAsFactors = FALSE);
```

# 1. Data input, cleaning and pre-processing

## 1.a Loading expression data

First, we read in raw counts from the breast cancer dataset.

```
BCData = read.csv("TNBC10vNormal10_2_sd.csv")
```

We can take a quick look at what is in the dataset.

```
dim(BCData); head(BCData); names(BCData)
```

```
## [1] 2050   23
```

```
##           Ensembl  HGNC Entrez TNBC1  TNBC2 TNBC3 TNBC4  TNBC5  TNBC6
## 1 ENSG00000167244  IGF2   3481   721    782   863   661    610    494
## 2 ENSG00000189058  APOD    347   214    320   182   783 385103    890
## 3 ENSG00000115414   FN1   2335 74123 351603 42180 17020 103344  86537
## 4 ENSG00000124942 AHNAK  79026 16910  28164  5561  1904  35614  13005
## 5 ENSG00000111341   MGP   4256  2387  31750 42924 29501   1949 267888
## 6 ENSG00000087086   FTL   2512 35567  67249 28968 15437 300220  35556
##   TNBC7 TNBC8 TNBC9 TNBC10 Normal1 Normal2 Normal3 Normal4 Normal5 Normal6
## 1   674    66   183 583239    4283    4032    8449   11820    4450    3233
## 2 10803   166  2443    158   12173   13625   12667   12711   87681   11696
## 3 38634 14152 52065  12911   10179   72480   18748   17637   20482    5230
## 4  3163  7273 12780  20982   45596  142264   81644  229782   87593   83023
## 5 20446 32207  4004   2588  206810   18101   84370   11172   79636   71991
## 6 27195 36701 22954  39937   33490  176245   17875   70548   81094   25022
##   Normal7 Normal8 Normal9 Normal10
## 1    8124    5993    3491     3702
## 2   11054   15817   32942    14442
## 3   16933   12902    3327    19109
## 4  165419  101222   30025   240497
## 5   50623  123079  125833    15452
## 6   44859   32186   25729   139545
```

```
##  [1] "Ensembl"  "HGNC"     "Entrez"   "TNBC1"    "TNBC2"    "TNBC3"
##  [7] "TNBC4"    "TNBC5"    "TNBC6"    "TNBC7"    "TNBC8"    "TNBC9"
## [13] "TNBC10"   "Normal1"  "Normal2"  "Normal3"  "Normal4"  "Normal5"
## [19] "Normal6"  "Normal7"  "Normal8"  "Normal9"  "Normal10"
```

Each row corresponds to a gene, and each column corresponds to a sample name or a gene annotation. We can remove the gene annotation data nad transpose the expression data for further analysis.

```
datExpr = as.data.frame(t(BCData[, -c(1:3)]))
names(datExpr) = BCData$Entrez
rownames(datExpr) = names(BCData)[-c(1:3)]
```

### 1.b Checking data for excessive missing values and identification of outlier microarray samples

We first check for genes and samples with too many missing values:

```
gsg <- goodSamplesGenes(datExpr, verbose = 3);
```

```
##  Flagging genes and samples with too many missing values...
##   ..step 1
```

```
gsg$allOK
```

```
## [1] TRUE
```

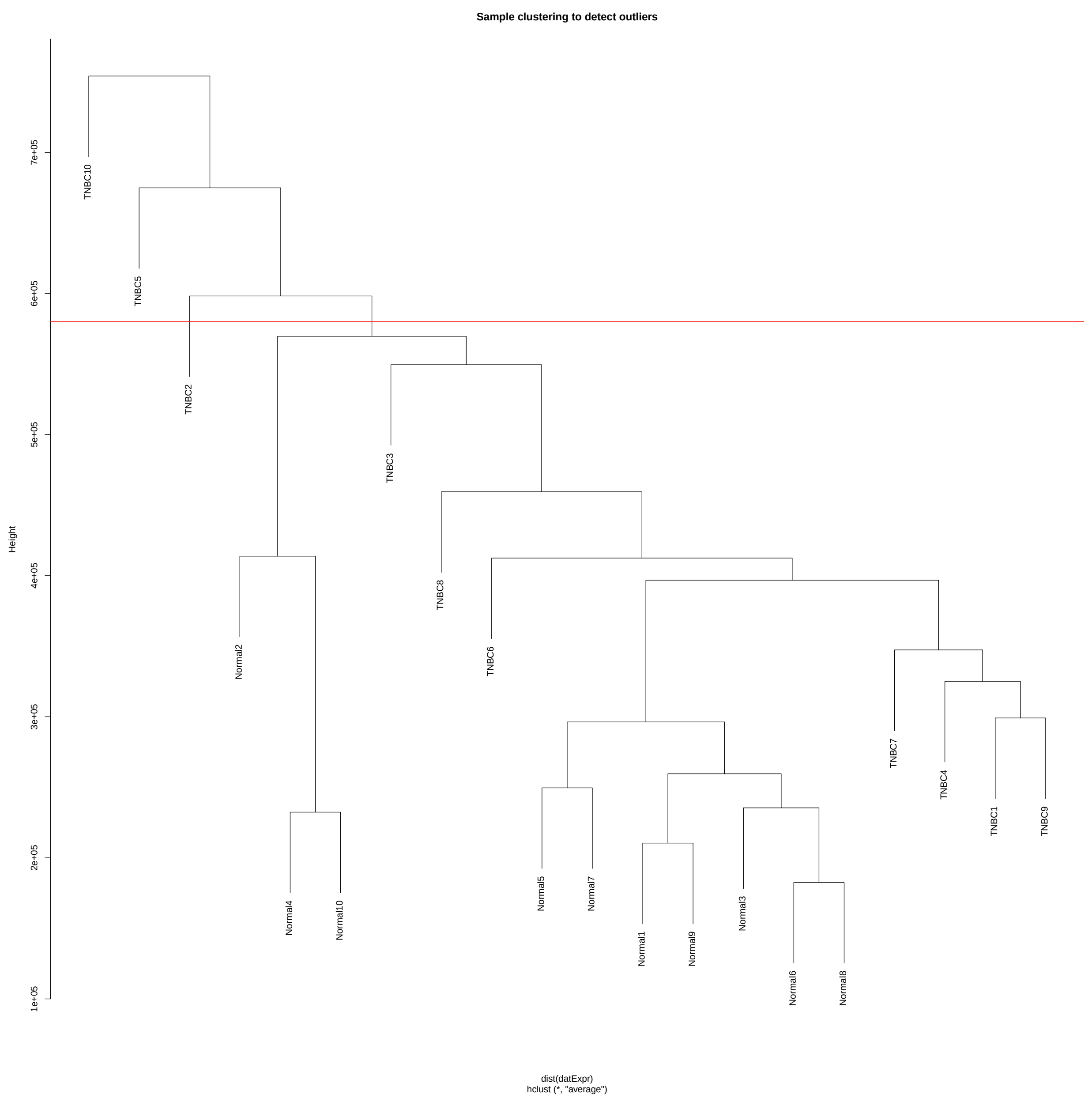
If the last statement returns TRUE, all genes have passed the cuts. If not, we remove the offending genes and samples from the data:

```
if (!gsg$allOK)
{
    # Optionally, print the gene and sample names that were removed
    if (sum(!gsg$goodGenes)>0)
        printFlush(paste("Removing genes:", paste(names(datExpr)[!gsg$goodGenes], collapse = ", ")))
    if (sum(!gsg$goodSamples)>0)
        printFlush(paste("Removing samples:", paste(rownames(datExpr)[!gsg$goodSamples], collapse = ",
    # Remove the offending genes and samples from the data
    datExpr = datExpr[gsg$goodSamples, gsg$goodGenes]
}
```

Next we cluster the samples (in contrast to clustering genes that will come later) to see if there are any obvious outliers. There are two outliers, TNBC2, TNBC5, and TNBC10. One can remove it by hand, or use an automatic approach. Choose a height cut that will remove the offending sample, say 5.8e+05 (the red line in the plot), and use a branch cut at that height. The variable datExpr now contains the expression data ready for network analysis.

```
sampleTree <- hclust(dist(datExpr), method = "average");
plot(sampleTree, main = "Sample clustering to detect outliers")

abline(h = 5.8e+05, col = "red")
```



```r
clust = cutreeStatic(sampleTree, cutHeight = 5.8e+05, minSize = 10)
table(clust)
```

```
## clust
##  0  1
##  3 17
```

```r
# clust 1 contains the samples we want to keep.
keepSamples <- (clust==1)
datExpr <- datExpr[keepSamples, ]
nGenes <- ncol(datExpr)
nSamples <- nrow(datExpr)
```

Weighted gene co-expression network analysis (WGCNA) is a systems biology method for describing the correlation patterns among genes and/or gene products. WGCNA can be used for finding clusters of highly correlated genes, for summarizing these clusters using the module eigengene or an intramodular hub gene, for relating clusters to one another and to external sample traits, and for calculating cluster membership measures.

Correlation networks facilitate network based gene screening methods that can be used to identify candidate biomarkers or therapeutic targets.

## 1. Hierarchical Clustering

Given a set of N items to be clustered, and an NxN distance (or similarity) matrix: 1. Start by assigning each item to its own cluster, so that if you have N items, you now have N clusters, each containing just one item. Let the distances (similarities) between the clusters equal the distances (similarities) between the items they contain 2. Find the closest (most similar) pair of clusters and merge them into a single cluster = N-1 Clusters 3. Compute distances (similarities) between the new cluster and each of the old clusters 4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size N