Homework 3

Ryan Yancey

27 July 2021

1.) Load the golub data training set in the multtest library. Also load Biobase and annotate libraries, if they are not loaded with the multtest library. Remember that the golub data training set is in the multtest library, so see the help file for information on this data set.

```r
# Load in local scripts
source("scripts/quiet_load.R")
source("scripts/t.test.all.genes.R")
source("scripts/wilcox.test.all.genes.R")

# Attach libraries quietly
libs <- c("multtest", "Biobase", "annotate", "limma")
sapply(libs, quiet_load)

# Import Golub et al. 1999 dataset
data(golub)
```

2.) Cast the matrix to a data frame and label the gene names as numbers (e.g. "g1","g2",etc).

```r
# Cast golub matrix to data frame
df.golub <- data.frame(golub)

# Set rownames to f(row) = "g.row"
rownames(df.golub) <- paste0("g.", rownames(df.golub))
```

3.) Get the sample labels (see lecture notes) and set the sample labels to the data frame.

```r
# Set colnames to "ALL" or "AML" based on golub.cl tumor class vector
colnames(df.golub) <- ifelse(golub.cl == 0, "ALL", "AML")
```

4.) Use the t-test function in the lecture #7 notes and modify it to `wilcox.test` instead of `t.test`. Change the `$p.value` argument to `$statistic`. Assign the following arguments to the function: `exact = FALSE`, `alternative = "two.sided"`, `correct = TRUE`. Run the function on all of the genes in the dataset and save it as `original.wmw.run`.

```r
# Logical vector of ALL samples
ALL.sample <- colnames(df.golub) == "ALL"

# Run function and separate df.golub rows by s1=ALL vs s2=AML
original.wmw.run <- apply(df.golub,
                          1,
                          wilcox.test.all.genes,
                          s1 = ALL.sample,
                          s2 = !ALL.sample)
```

5.) Now write a for loop to iterate 500 times, where in each iteration, the columns of the data frame are shuffled (class labels mixed up), the WMW test is calculated on all of the genes, and the maximum test statistic (W) is saved in a list.

```r
# Initialize an empty vector to store max test stats
W <- c()

# Loop 500 times
for (i in 1:500) {
  shuffled_cols <- sample(ncol(df.golub)) # shuffle columns
  df.golub.mix <- df.golub[, shuffled_cols] # reorder data frame
  # run wilcox test on all genes
  tmp.wmw.run <- apply(df.golub.mix,
                       1,
                       wilcox.test.all.genes,
                       s1 = ALL.sample,
                       s2 = !ALL.sample)
  # append to list
  W <- c(W, max(tmp.wmw.run))
}
```

6.) Once you have the list of maximum test statistics, get the 95% value test statistic. Subset the original.wmw.run list of values with only those that have a higher test statistic than the 95% value that you calculated. Print the gene names and test statistics out.

```
# 95% value statistic
x <- quantile(W, probs = 0.95)

# Top genes
(subset.original.wmw.run <- original.wmw.run[original.wmw.run > x])
```

```
##   g.96  g.283  g.329  g.345  g.394  g.422  g.523  g.546  g.561  g.621  g.648
##    274    271    275    274    290    270    283    274    281    269    271
##  g.703  g.704  g.717  g.738  g.746  g.835  g.838  g.839  g.849  g.866  g.922
##    285    273    283    271    277    272    283    272    272    269    272
##  g.984 g.1006 g.1037 g.1042 g.1045 g.1086 g.1271 g.1368 g.1524 g.1598 g.1811
##    283    275    281    284    270    278    268    279    282    275    284
## g.1817 g.1834 g.1869 g.1883 g.1909 g.1916 g.1920 g.1939 g.1959 g.1978 g.1995
##    272    290    272    281    275    273    274    268    272    271    287
## g.2002 g.2122 g.2266 g.2289 g.2386 g.2418 g.2489 g.2616 g.2645 g.2702 g.2801
##    283    270    272    274    288    279    288    270    272    279    274
## g.2829 g.2851 g.2860 g.2879 g.2939 g.3046
##    273    281    272    272    291    276
```

7.) Now we want to compare these results to those using the empirical Bayes method in the `limma` package. Load this library and calculate p-values for the same dataset using the `eBayes()` function.

```
# Design (1 = ALL, 0 = AML)
design <- cbind(Grp1 = 1, Grp2vs1 = c(rep(1, sum(ALL.sample)), rep(0, sum(!ALL.sample))))

# Empirical Bayes and extract p-values
fit <- lmFit(df.golub, design)
fit <- eBayes(fit)$p.value[,2]
```

8.) Sort the empirical Bayes p-values and acquire the lowest n p-values, where n is defined as the number of significant test statistics that you found in problem 6. Intersect the gene names for your two methods and report how many are in common between the two differential expression methods, when choosing the top n genes from each set.

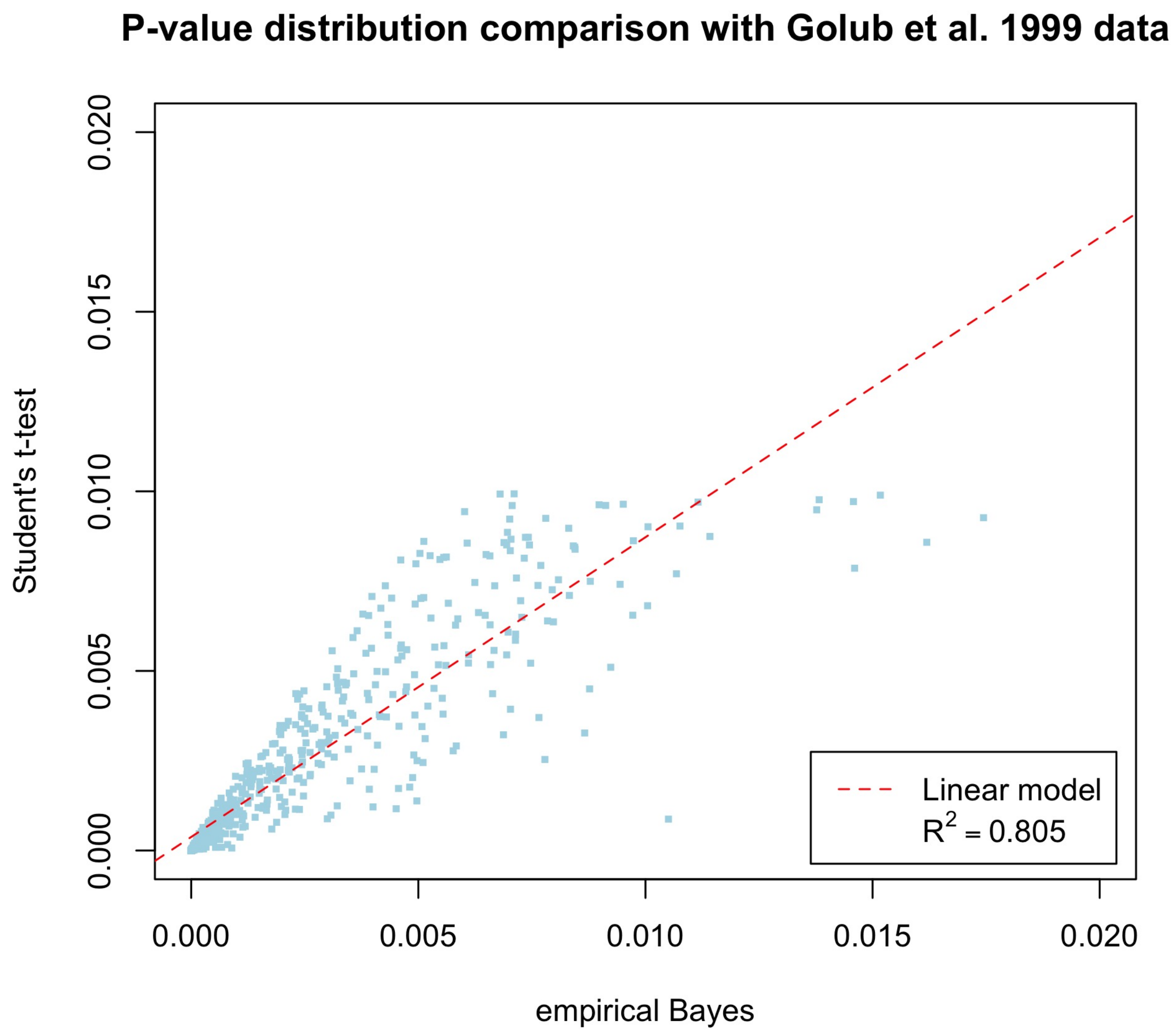
```
# Sort descending
sorted.fit <- sort(fit)
n <- length(subset.original.wmw.run)

# Lowest n p-values
lowest.pvals <- sorted.fit[1:n]

# Which ones intersect with the original list
(i <- intersect(names(subset.original.wmw.run), names(lowest.pvals)))
```

```
##  [1] "g.394"  "g.523"  "g.561"  "g.717"  "g.746"  "g.849"  "g.1037" "g.1042"
##  [9] "g.1524" "g.1811" "g.1834" "g.1883" "g.1995" "g.2266" "g.2289" "g.2386"
## [17] "g.2489" "g.2702" "g.2939"
```

The two differential expression methods found 19 (31.15%) significant genes in common.

9.) Finally, compare the results from a Student's t-test with the empirical Bayes method. To do this, first calculate a two sample (two-tailed) Student's t-test on all genes. Make sure that you are running a Student's t-test and not a Welch's t-test. Then extract only those genes with a p-value less than 0.01 from this test. Plot the gene p-values < 0.01 for the Student's t-test vs. the same genes in the empirical Bayes method. Make sure to label the axes and title appropriately.

```r
# Two-tailed t-test
t.test.run <- apply(df.golub,
                    1,
                    t.test.all.genes,
                    s1 = ALL.sample,
                    s2 = !ALL.sample)

# Extract those with p-values less than 0.01
t.test.run <- t.test.run[t.test.run < 0.01]

# Extract the same genes from eBayes fit
lowest.pvals <- fit[names(t.test.run)]

# Student's t-test vs Empirical Bayes
plot(
  t.test.run ~ lowest.pvals,
  xlab = "empirical Bayes",
  xlim = c(0, 0.02),
  ylab = "Student's t-test",
  ylim = c(0, 0.02),
  main = "P-value distribution comparison with Golub et al. 1999 data",
  cex = 0.5,
  col = "lightblue",
  pch = 15
)

# Add linear model line
lm <- lm(t.test.run ~ lowest.pvals)
abline(lm, col = "red", lty = 2) # plot linear model

# Add legend to plot
rsquared <- round(summary(lm)$r.squared, 3)
lab1 <- parse(text = sprintf('R^2 == %s', rsquared))
legend(
  "bottomright",
  legend = c("Linear model", lab1),
  col = c("red", NULL),
  lty = c(2, 0),
  inset = 0.02
)
```



From this plot, one can determine the two significance testing methods are strongly correlated, as a linear model accounts for 80.5% of the variation between the two sets.

Source any scripts imported below

```r
# quiet_load() attaches libraries without messages output (quietly)
quiet_load
```

```
## function (x)
## {
##     if (x %in% installed.packages()) {
##         suppressPackageStartupMessages(library(x, character.only = TRUE))
##     }
## }
## <bytecode: 0x7f836f013270>
```

```r
# wilcox.test.all.genes() computes the Wilcoxon-Mann-Whitney test for each gene
wilcox.test.all.genes
```

```
## function (x, s1, s2)
## {
##     x1 <- x[s1]
##     x2 <- x[s2]
##     x1 <- as.numeric(x1)
##     x2 <- as.numeric(x2)
##     wilcox.out <- wilcox.test(x1, x2, exact = FALSE, alternative = "two.sided",
##         correct = TRUE)
##     out <- as.numeric(wilcox.out$statistic)
##     return(out)
## }
## <bytecode: 0x7f8367fc6090>
```

```r
# t.test.test.all.genes() computes the Student's t-test for each gene
t.test.all.genes
```

```
## function (x, s1, s2)
## {
##     x1 <- x[s1]
##     x2 <- x[s2]
##     x1 <- as.numeric(x1)
##     x2 <- as.numeric(x2)
##     t.out <- t.test(x1, x2, alternative = "two.sided", var.equal = T)
##     out <- as.numeric(t.out$p.value)
##     return(out)
## }
## <bytecode: 0x7f836d386400>
```

Session info

```r
sessionInfo()
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
## [1] limma_3.48.1         annotate_1.70.0      XML_3.99-0.6
## [4] AnnotationDbi_1.54.1 IRanges_2.26.0       S4Vectors_0.30.0
## [7] multtest_2.48.0      Biobase_2.52.0       BiocGenerics_0.38.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.7             XVector_0.32.0         GenomeInfoDb_1.28.1
##  [4] compiler_4.1.0         zlibbioc_1.38.0        bitops_1.0-7
##  [7] tools_4.1.0            digest_0.6.27          bit_4.0.4
## [10] RSQLite_2.2.7          evaluate_0.14          memoise_2.0.0
## [13] lattice_0.20-44        png_0.1-7              rlang_0.4.11
## [16] Matrix_1.3-4           DBI_1.1.1              yaml_2.2.1
## [19] xfun_0.24              fastmap_1.1.0          GenomeInfoDbData_1.2.6
## [22] stringr_1.4.0          httr_1.4.2             knitr_1.33
## [25] Biostrings_2.60.1      vctrs_0.3.8            bit64_4.0.5
## [28] grid_4.1.0             R6_2.5.0               survival_3.2-11
## [31] rmarkdown_2.9          blob_1.2.1             magrittr_2.0.1
## [34] htmltools_0.5.1.1      MASS_7.3-54            splines_4.1.0
## [37] KEGGREST_1.32.0        xtable_1.8-4           stringi_1.6.2
## [40] RCurl_1.98-1.3         cachem_1.0.5           crayon_1.4.1
```