Gene set enrichment analysis

Kim Dill-McFarland, kadm@uw.edu

version July 14, 2021

Contents

Overview

In this workshop, we introduce gene set analysis relevant to RNA-sequencing data. In it, we cover:

- Broad Molecular Signatures Database (MSigDB) gene sets
- hypergeometric enrichment with `clusterProfiler`
- gene set enrichment analysis (GSEA) with `fgsea`

During the workshop, we will build an R script together, which will be posted as 'live_notes' after the workshop here.

Prior to the workshop

Please install R, RStudio, and the following packages. See the setup instructions for more details.

```
#Data manipulation
library(tidyverse)
#Broad gene set database
library(msigdbr)
#Hypergeo enrichment
library(clusterProfiler)
#GSEA
library(fgsea)
```

Load data

Briefly, these data are from RNA-sequencing of human monocytes alone (`MEDIA`) or infected with *M. tuberculosis* in vitro (`TB`). Expression data are in an EList object containing expression (`E`), sample/patient metadata (`targets`), and gene metadata (`genes`).

```
#RNAseq expression data
load("data/RSTR_data_clean_subset.RData")
class(dat)
```

```
## [1] "EList"
## attr(,"package")
## [1] "limma"
```

```
names(dat)
```

```
## Loading required package: limma
```

```
## [1] "genes"   "targets" "E"
```

Additionally we have the results of a linear mixed effect model for each gene in the expression data. For this workshop, these data a filtered to just the TB-MEDIA variable.

```
#Linear model results
model.results <- read_csv("data/RSTR.Mtb.model.subset.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   gene = col_character(),
##   variable = col_character(),
##   pval = col_double(),
##   FDR = col_double(),
##   ensembl_gene_id = col_character(),
##   CHR = col_character(),
##   start = col_double(),
##   end = col_double(),
##   entrezgene_id = col_character()
## )
```

```
class(model.results)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

```
head(model.results)
```

```
## # A tibble: 6 x 9
##   gene   variable       pval      FDR ensembl_gene_id CHR       start       end
##   <chr>  <chr>         <dbl>    <dbl> <chr>           <chr>     <dbl>     <dbl>
## 1 A1BG   conditionTB 1.2e- 1 1.51e- 1 ENSG00000121410 19     58345178  58353492
## 2 A2M    conditionTB 7   e- 4 1.19e- 3 ENSG00000175899 12      9067664   9116229
## 3 A2ML1  conditionTB 3.9e- 1 4.40e- 1 ENSG00000166535 12      8822621   8887001
## 4 A4GALT conditionTB 0       0        ENSG00000128274 22     42692121  42721298
## 5 AAAS   conditionTB 6.8e-13 2.21e-12 ENSG00000094914 12     53307456  53324864
## 6 AACS   conditionTB 7.4e- 3 1.12e- 2 ENSG00000081760 12    125065434 125143333
## # ... with 1 more variable: entrezgene_id <chr>
```

Gene sets

Definition: two or more genes with shared function, structure, localization, or any other defining similarity

Use: glean interpretable, biologically-relevant meaning from a list of genes of interest

Gene sets can be as general as "cytosolic" to as specific as "up-regulated in blood vessel cells in response to wound in *Roy et al 2007*". They can be defined by an individual research study (like the latter) up to generally agreed upon knowledge formed from an entire body of literature (like the former). They also vary in size (2 to thousands), are not exclusive (one gene is in many gene sets), and many are redundant (transmembrane, membrane, lipid bilayer. . .).

It is up to you, the researcher, to select what group of gene sets you want to test. More on this below. Keep in mind, this is definitely not a case for "more is better". For example, getting 200 significant gene sets from your 250 significant genes does not help you much.

Broad MSigDB

The Broad Institute has collated many useful gene sets in their Molecular Signatures Database (MSigDB). This includes the following used in this workshop (descriptions modified from Broad).

- Hallmark: summarize and represent specific well-defined biological processes. Generated by a computational methodology based on identifying overlaps between gene sets in other MSigDB collections and retaining genes that display coordinate expression
  - Total: 50
  - Examples: glycolysis, inflammatory response, apoptosis
- Curated (C2)
  - Canonical pathways: from pathway databases including KEGG, REACTOME, etc. Canonical representations of a biological process compiled by domain experts
  - Total: 2922
  - Examples: Caspase pathway, signaling by NOTCH1, DNA repair
- Gene ontology GO (C5)
  - Biological process: molecular-level activities performed by gene products
  - Total: 7481
  - Examples: viral life cycle, vitamin D biosynthetic process, mitochondrial calcium-ion transmembrane transport

Hallmark is the most general group and is usually where we start our analyses. It helps reduce the complexity of a large gene list down to easily understood terms. It also provides some context when looking at more specific gene sets as you can often group the other sets under a more general Hallmark term. However, because Hallmark is the most general, it is also the most incomplete with 50 gene sets representing just under 4400 genes.

Curated and gene ontology sets are more specific than Hallmark, though they contain sets with a range of specificities. They are particularly useful in adding details to a Hallmark term or defining testable hypotheses for follow-up experiments in the lab. They contain a lot more genes than Hallmark but many sets are redundant or overlapping. If there are many significant Hallmark terms, we often do not run these more specific sets as it gives too many results for interpretation. In contrast, when there are few significant Hallmark terms, these more granular gene sets may provide better insight.

There are several other sets in MSigDB that are not described here. Some of these are too specific (individual studies) to be of use in our analyses (C2 chemical and genetic perturbations or C7 immunologic signatures) while others cover areas not relevant to our experimental design (C1 chromosome position, C3 gene regulation, C4/C6 cancer-oriented). There are still more gene sets that are not included in MSigDB, though this database is a pretty comprehensive place to start.

In reality, we usually run all of Hallmark, canonical pathways, and biological process. Then, we consider how many significant gene sets each gives us, think further about the top hits, and perform additional analyses or experiments to see which will be highlighted in publication.

MSigDB in R

We will access MSigDB from within R using the package `msigdbr`. It is important to update this package frequently as Broad continues to improve its gene set collections.

We can see the R version we have

```
packageVersion("msigdbr")
```

```
## [1] '7.4.1'
```

And verify that it is up-to-date with the latest online versions at https://www.gsea-msigdb.org/gsea/downloads.jsp

Next we will use gene sets to unravel how monocytes respond to *M. tuberculosis* infection.

Hypergeometric enrichment

This analysis is a simple enrichment. We ask if our list of significant genes is enriched in gene sets more than random chance allows.

Get gene set data

First, we extract the lists of genes in Broad Hallmark terms from the package `msigdbr`. We format it as a data frame so we can use the `tidyverse`.

```
#Hallmark
H <- as.data.frame(msigdbr(species = "Homo sapiens",
                           category = "H"))
```

Here, we see the start of the Hallmark data including

- gs_cat: gene set category, in this case H for Hallmark
- gs_name: short gene set name
- gene_symbol: HGNC symbol for each each in the gene set
- gs_description: sentence describing the gene set name further
- additional variables: other gene identifiers like ENSEMBL, ENTREZ, etc

```
head(H)
```

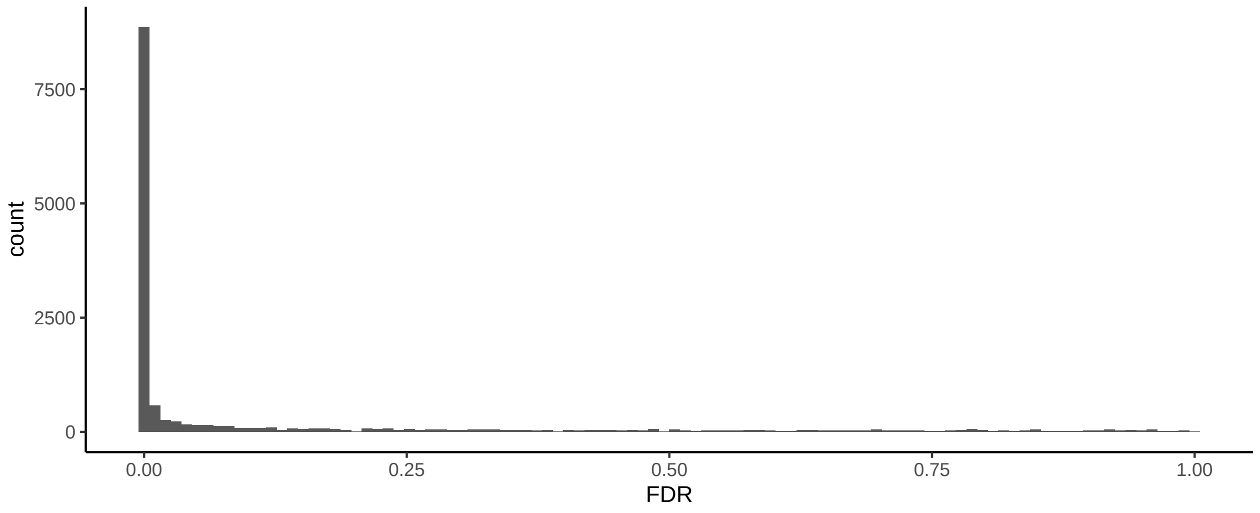
```
##   gs_cat gs_subcat               gs_name gene_symbol entrez_gene
## 1      H           HALLMARK_ADIPOGENESIS       ABCA1          19
## 2      H           HALLMARK_ADIPOGENESIS       ABCB8       11194
## 3      H           HALLMARK_ADIPOGENESIS       ACAA2       10449
## 4      H           HALLMARK_ADIPOGENESIS       ACADL          33
## 5      H           HALLMARK_ADIPOGENESIS       ACADM          34
## 6      H           HALLMARK_ADIPOGENESIS       ACADS          35
##      ensembl_gene human_gene_symbol human_entrez_gene human_ensembl_gene gs_id
## 1 ENSG00000165029             ABCA1                19    ENSG00000165029 M5905
## 2 ENSG00000197150             ABCB8             11194    ENSG00000197150 M5905
## 3 ENSG00000167315             ACAA2             10449    ENSG00000167315 M5905
## 4 ENSG00000115361             ACADL                33    ENSG00000115361 M5905
## 5 ENSG00000117054             ACADM                34    ENSG00000117054 M5905
## 6 ENSG00000122971             ACADS                35    ENSG00000122971 M5905
##   gs_pmid gs_geoid gs_exact_source gs_url
## 1
## 2
## 3
## 4
## 5
## 6
##                                                          gs_description
## 1 Genes up-regulated during adipocyte differentiation (adipogenesis).
## 2 Genes up-regulated during adipocyte differentiation (adipogenesis).
## 3 Genes up-regulated during adipocyte differentiation (adipogenesis).
## 4 Genes up-regulated during adipocyte differentiation (adipogenesis).
## 5 Genes up-regulated during adipocyte differentiation (adipogenesis).
## 6 Genes up-regulated during adipocyte differentiation (adipogenesis).
```

Define significant genes

Next, we must define our "significant genes". This type of enrichment is binary in that it only considers if a gene is significant or not. There is no granularity beyond yes/no. Thus, it is very important to consider our FDR cutoff as it is an arbitrary value that can dramatically impact results.

Let's plot our FDR values as a histogram to help us determine a cutoff. We see that the data are heavily skewed. This is not uncommon when you have a large biological perturbation like bacterial infection.

```
ggplot(model.results, aes(x=FDR)) +
  geom_histogram(bins=100) +
  theme_classic()
```



In fact, there is a large proportion of FDR values that are zero.

```
table(model.results$FDR == 0)
```

```
##
## FALSE  TRUE
## 10476  3502
```

These zeroes are actually a result of limitations in our linear modeling function. After a certain point (in this case, 1E-16), the function stops computing and saves the P-value as zero to save time.

Here, we will use FDR < 1E-16 to see the genes that change the most with Mtb infection. *This is not necessarily what I'd recommend for these data.* It's just for workshop purposes.

```
signif <- model.results %>%
  filter(FDR <= 1E-16)
```

Run enrichment

Now we can run the enrichment in `clusterProfiler`. We can use any gene ID in our data set; it just has to match the gene IDs in the database.

Here, we run the enrichment using ENTREZ ID. Note that you must select ONLY the columns needed from the Hallmark database. `enricher( )` isn't smart enough to know which columns to use on its own.

```
#Select unique EMTREZ IDs for significant genes
signif.entrez <- unique(signif$entrezgene_id)
#Select matching ID column in database
H.entrez <- select(H, gs_name, entrez_gene)
#enrichment
enrich.H <- enricher(gene = signif.entrez, TERM2GENE = H.entrez)
```

Similarly, we could use ENSEMBL IDs.

```
#Select unique ENSEMBL IDs for significant genes
signif.ensembl <- unique(signif$ensembl_gene_id)
#Select matching ID column in database
H.ensembl <- select(H, gs_name, ensembl_gene)
#enrichment
enrich.H <- enricher(gene = signif.ensembl, TERM2GENE = H.ensembl)
```

Importantly, these will yield slightly different results, because the ID databases are not perfect. We must just accept that this is life and choose whichever ID we have / prefer. Since the expression data use ENSEMBL, we'll move forward with that.

```
length(signif.ensembl)
```

```
## [1] 3492
```

```
length(signif.entrez)
```

```
## [1] 3496
```

Extract enrichment results

`clusterProfiler` outputs results in an S4 object.

```
class(enrich.H)
```

```
## [1] "enrichResult"
## attr(,"package")
## [1] "DOSE"
```

We briefly mentioned this data type in intro R. It works very similarly to S3 (like the expression data) except you use `@` to extract data. For example, the enrichment results are in a data frame.

```
enrich.H@result
```

This includes

```
colnames(enrich.H@result)
```

```
## [1] "ID"            "Description" "GeneRatio"   "BgRatio"     "pvalue"
## [6] "p.adjust"      "qvalue"      "geneID"      "Count"
```

- ID: Unique name for gene set
- Description: Descriptive text for gene set, often the same as ID
- GeneRatio: Significant genes in gene set / Significant genes in all gene sets tested
- BgRatio: Total genes in gene set / Total genes in all gene sets tested
- pvalue: Significance
- p.adjust: FDR adjusted significance
- qvalue: Q-value used in P-value calculation
- geneID: vector of significant gene IDs in gene set. Separated by /
- Count: Significant genes in gene set

These column names aren't the most informative and R doesn't understand the ratios as numbers.

```
class(enrich.H@result$GeneRatio)
```

```
## [1] "character"
```

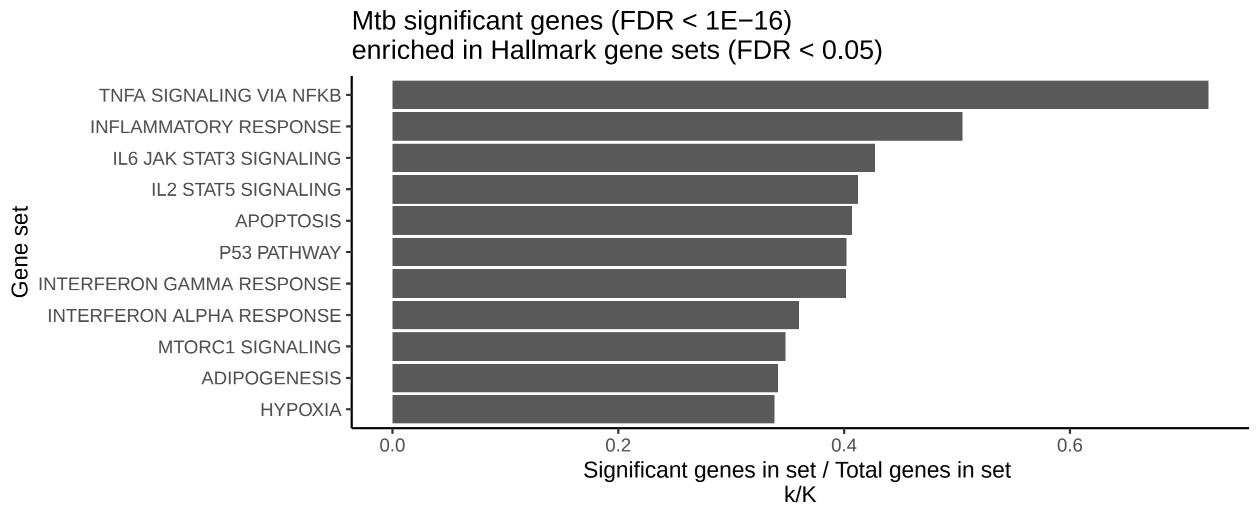
So we'll do some additional formatting. We refer to a gene set as a "term" and the Hallmark database as a "category" for brevity.

```
enrich.H.df <- enrich.H@result %>%
  #separate ratios into 2 columns of data
  separate(BgRatio, into=c("size.term","size.category"), sep="/") %>%
  separate(GeneRatio, into=c("size.overlap.term", "size.overlap.category"),
               sep="/") %>%
  #convert to numeric
  mutate_at(vars("size.term","size.category",
                     "size.overlap.term","size.overlap.category"),
                as.numeric) %>%
  #Calculate k/K
  mutate("k.K"=size.overlap.term/size.term)
```

Visualize significant enrichment

The most common visualization for enrichment is k/K as it is the ratio of significant genes relative to total genes in the gene set. We also subset to significant enrichment at a defined FDR. Here, let's use FDR < 0.05. This is actually pretty strict for enrichment and people often go up to FDR < 0.2.

```
enrich.H.df %>%
  filter(p.adjust <= 0.05) %>%
  #Beautify descriptions by removing _ and HALLMARK
  mutate(Description = gsub("HALLMARK_","", Description),
         Description = gsub("_"," ", Description)) %>%

ggplot(aes(x=reorder(Description, k.K), #Reorder gene sets by k/K values
           y=k.K)) +
  geom_col() +
  theme_classic() +
  #Some more customization to pretty it up
  #Flip x and y so long labels can be read
  coord_flip() +
  #fix labels
  labs(y="Significant genes in set / Total genes in set \nk/K",
       x="Gene set",
       title = "Mtb significant genes (FDR < 1E-16)\nenriched in Hallmark gene sets (FDR < 0.05)")
```



Exercises

Here's some helpful code to extract the other gene sets of interest.

```
#Canonical pathways
#Note that we have to call each database and combine them together
C2.CP <- as.data.frame(msigdbr(species = "Homo sapiens",
                               category = "C2",
                               subcategory = "CP:BIOCARTA")) %>%
        bind_rows(as.data.frame(msigdbr(species = "Homo sapiens",
                                category = "C2",
                                subcategory = "CP:KEGG"))) %>%
        bind_rows(as.data.frame(msigdbr(species = "Homo sapiens",
                                category = "C2",
                                subcategory = "CP:PID"))) %>%
        bind_rows(as.data.frame(msigdbr(species = "Homo sapiens",
                                category = "C2",
                                subcategory = "CP:REACTOME")))
#Gene ontology biological process
C5.BP <- as.data.frame(msigdbr(species = "Homo sapiens",
                               category = "C5", subcategory = "GO:BP"))
```

1. Run enrichment of the current significant gene list against the canonical pathways `C2.CP` database. Do you get more or fewer significant gene sets? Is this what you'd expect give what you know about C2?
2. Run Hallmark enrichment on genes significant at FDR < 0.01. Do you get more or fewer significant gene sets? Why might this be the case?
3. Further modify the final enrichment ggplot. Add color, change the FDR cutoff, do whatever your heart desires!

Gene set enrichment analysis (GSEA)

Gene set enrichment analysis (GSEA) is another way to use gene sets to understand changes in gene expression. The term GSEA sounds general, but it is understood in the field to specifically refer to the following type of analysis. We often clarify it as “fold change GSEA” in our group since it can be confused with the enrichment we performed above.

GSEA compares expression in two biological states and determines if gene sets show significant, concordant change. That is, GSEA determines if genes in a set generally increase or decrease in expression. Some important differences between GSEA and simple enrichment are

- GSEA uses data from all genes; enrichment uses only your defined significant genes. This means you don’t have to worry about FDR cutoffs for picking genes to go into GSEA.
- GSEA uses numeric fold change values; enrichment uses binary significant vs not. Thus, GSEA can identify pathways where lots of small changes not significant at the gene level add up to a significant change at the gene set level. However, since GSEA incorporates direction, you may miss gene sets where discordant changes result in similar biology. For example, an increase in an activator having the same effect as a decrease in a repressor.
- GSEA uses mean fold change per gene; enrichment depends on how you defined significant genes (e.g. your model could have used individual values, means, etc). Thus, if you have lots of variation between samples, GSEA does not capture this and may give misleading results. If you suspect another factor has significant impacts on gene expression, you can run GSEA on subsets so that the mean values are more representative. For example, mean fold change in children separate from adults if age is an important factor.
- GSEA compares only two states; enrichment can compare any number of states. This is similar to an ANOVA telling you there is a significant effect of variable X but needing a posthoc pairwise test to say if within that variable, A differs from B, B from C, etc. You can define significant genes in enrichment by the overall ANOVA or several pairwise lists. In contrast, GSEA only compares pairwise tests since it uses fold change. Often, when you have more than two states, the pairwise tests are more interpretable anyway.

We usually run enrichment and GSEA on a data set and use both in interpreting results.

Get gene set data

We use the same gene set data as with enrichment. As a reminder, this is extracted from `msigdbr` with

```
#Hallmark
H <- as.data.frame(msigdbr(species = "Homo sapiens",
                           category = "H"))
```

However, `fgsea` needs the database as a list, not a data frame. There are many ways we can accomplish this. Here is one using the `tidyverse`. Remember to select the gene ID column that matches your fold change data!

```
H.ensembl.ls <- H %>%
  #Keep gene ID that match expression data gene ID
  select(gs_name, ensembl_gene) %>%
  #Collapse all genes in each gene set into 1 row each
  group_by(gs_name) %>%
  summarise(all.genes = list(unique(ensembl_gene))) %>%
  #Convert to list
  deframe()
```

Calculate fold change

GSEA compares mean gene expression fold change between two states. Thus, we calculate this from the expression data in `dat`.

If you are running this outside the live workshop, be sure to look at how the data frame changes with each step. To learn more about the `tidyverse` and how this is done, see Intro R.

```
#Extract expression data
FC <- as.data.frame(dat$E) %>%
  #Move gene IDs from rownames to a column
  rownames_to_column("ensembl_gene_id") %>%
  #Make long format with all expression values in 1 column
  pivot_longer(-ensembl_gene_id,
               names_to = "libID", values_to = "expression") %>%
  #Extract RSID and TB condition from libID
  #If this info was not in the libID, we could get it by joining
  # with dat$targets
  separate(libID, into = c("RSID","condition"), sep="_") %>%
  #Make wide with media and tb expression in separate columns
  pivot_wider(names_from = condition, values_from = expression) %>%
  #Calculate tb minus media fold change (delta for change)
  #Because expression values are log2, subtraction is the same as division
  mutate(delta = TB-MEDIA) %>%
  #Calculate mean fold change per gene
  group_by(ensembl_gene_id) %>%
  summarise(mean.delta = mean(delta, na.rm=TRUE)) %>%
  #Arrange by descending fold change
  arrange(desc(mean.delta))

head(FC)
```

```
## # A tibble: 6 x 2
##   ensembl_gene_id mean.delta
##   <chr>                <dbl>
## 1 ENSG00000122641      10.3
## 2 ENSG00000115009       9.20
## 3 ENSG00000164400       9.08
## 4 ENSG00000136244       8.87
## 5 ENSG00000108342       8.69
## 6 ENSG00000102794       8.55
```

`fgsea` requires our fold changes to be in a decreasing, named vector (or list of vectors if you have multiple groups). So we format as a vector for our one group of TB-MEDIA fold changes.

```
#Vector of mean fold change values
FC.vec <- FC$mean.delta
#Add names
#Importantly, since both columns come from the same data frame,
#we are confident that they are in the same order
  names(FC.vec) <- FC$ensembl_gene_id
```

Determine score type

GSEA calculates significance differently if you have only positive (`pos`), only negative (`neg`), or mixed (`std`) fold change values. This is analogous to a one- vs two-tailed t-test.

We can set this by-hand by checking our minimum and maximum fold changes. We see we have positive and negative, thus our type is `std`.

```
min(FC.vec)
```

```
## [1] -6.260521
```

```
max(FC.vec)
```

```
## [1] 10.28815
```

```
scoreType <- "std"
```

Or here is a function to automatically set this!

```
if(min(FC.vec) < 0 & max(FC.vec) > 0){
        scoreType <- "std"
      } else if(max(FC.vec) <= 0){
        scoreType <- "neg"
      } else if(min(FC.vec) >= 0){
        scoreType <- "pos"
      } else{
        stop("Could not determine score type from fold changes.")
      }
```

Run GSEA

Using our formatted data, we run GSEA. We also set the number of permutations to run with `nperm`, which determines how many times the algorithm is run to estimate p-values.

```
gsea.H <- as.data.frame(fgseaSimple(pathways = H.ensembl.ls,
                                    stats = FC.vec,
                                    nperm = 1000,
                                    scoreType = scoreType))
```

```
## Warning in preparePathwaysAndStats(pathways, stats, minSize, maxSize, gseaParam, : There are ties in
## The order of those tied genes will be arbitrary, which may produce unexpected results.
```

You'll sometimes see a warning message about ties in the data, as seen above. This means some genes have the exact same fold change value. This can occur for several reasons including 1) rounding, 2) genes with multiple IDs in a database, and 3) true results where genes have the same fold change. It is not something to worry about as long as the percentage is low. Here, we have 0.22% (which equates to 32 out of 14,576 genes). If the percentage is high, check that your data are correct with 1 mean fold change value per gene.

You may also see a warning like

```
There were X pathways for which P-values were not calculated properly due to unbalanced gene-level stat
```

This occurs when `nperm` is not large enough to accurately estimate very small p-values (like those zeros in the gene linear model results!) If you see this error, try increasing `nperm` by orders of 10. Keep in mind that more permutations increase the time it takes to run GSEA. You may reach a point where it's no longer worth it and should just fill in these NA with zeroes.

Extract GSEA results

GSEA outputs a data frame of results.

```
class(gsea.H)
```

```
## [1] "data.frame"
```

This includes

```
colnames(gsea.H)
```

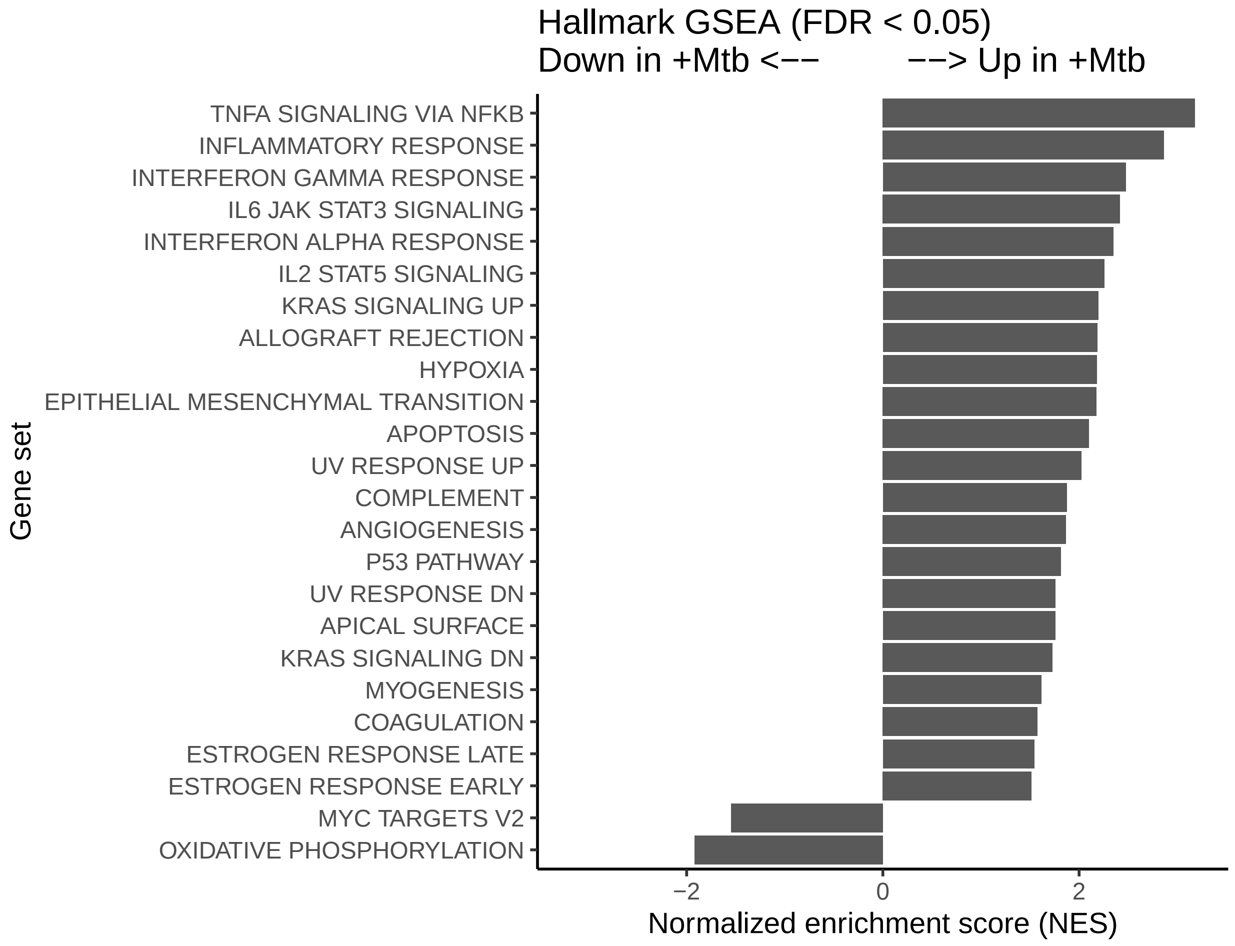
```
## [1] "pathway"      "pval"         "padj"         "ES"           "NES"
## [6] "nMoreExtreme" "size"         "leadingEdge"
```

- pathway: Unique name for gene set
- pval: Significance
- padj: FDR adjusted significance
- ES: Enrichment score (same as Broad's online GSEA program)
- NES: Enrichment score normalized to mean enrichment of random samples of the same size
- nMoreExtreme: Number of times a random gene set had a more extreme enrichment score value
- size: Total genes in gene set also found in fold change data
- leadingEdge: Leading edge genes that drive the enrichment,

Visualize significant GSEA

The most common visualization for GSEA is the enrichment or normalized enrichment score. Let's plot `NES` for the subset of significant GSEA at FDR < 0.05. This is actually pretty strict for GSEA and people often go up to FDR < 0.2.

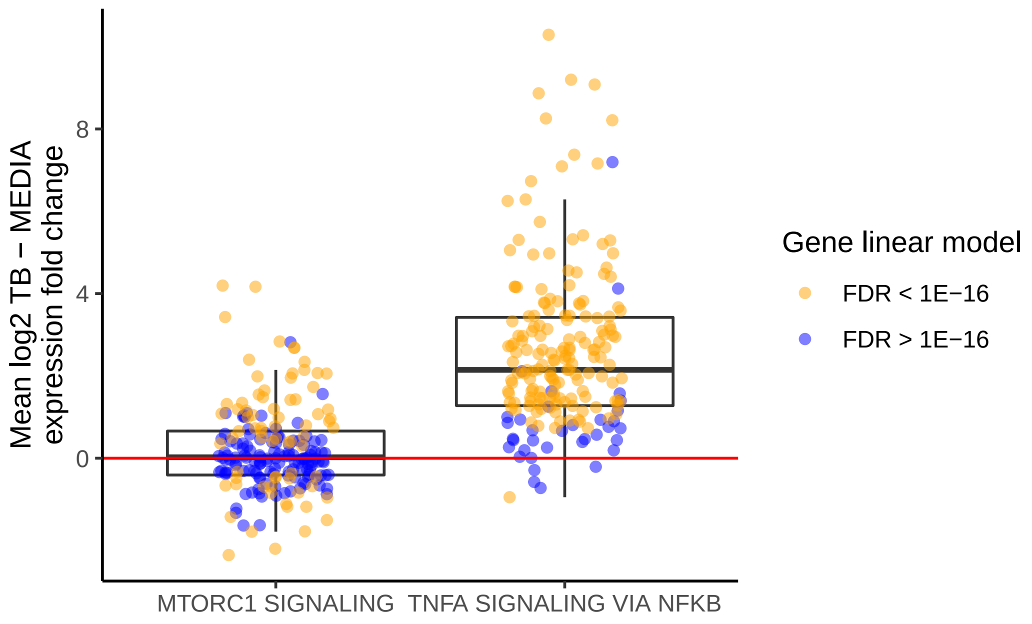
```
gsea.H %>%
  filter(padj <= 0.05) %>%
  #Beautify descriptions by removing _ and HALLMARK
  mutate(pathway = gsub("HALLMARK_","", pathway),
         pathway = gsub("_"," ", pathway)) %>%

ggplot(aes(x=reorder(pathway, NES), #Reorder gene sets by NES values
           y=NES)) +
  geom_col() +
  theme_classic() +
  #Force equal max min
  lims(y=c(-3.2,3.2)) +
  #Some more customization to pretty it up
  #Flip x and y so long labels can be read
  coord_flip() +
  #fix labels
  labs(y="Normalized enrichment score (NES)",
       x="Gene set",
       title = "Hallmark GSEA (FDR < 0.05)\nDown in +Mtb <--         --> Up in +Mtb")
```



You'll see that GSEA identifies all the same Hallmark gene sets as simple enrichment (except MTORC1 SIGNALING) plus a bunch more. This is because we did not subset to just the *most* significant genes. We are able to see more when including fold change of all genes available.

We can also label GSEA as up/down expression because we know that we calculated fold change as TB - MEDIA. Thus, positive enrichment corresponds to higher expression in TB.

So why does MTORC1 SIGNALING show up in enrichment but not GSEA? This is because there are many genes in this gene set that change with Mtb infection but they are not concordant. We see that some MTORC1 genes have positive while others have negative mean fold change. In contrast, TNFA signaling (which was significant in both enrichment and GSEA) has mostly genes with positive fold change. Thus, MTORC GSEA is closer to zero and not significant, while TNFA signaling GSEA is positive and significant.



Exercises

1. Run GSEA against the canonical pathways `C2.CP` database. Do the most enriched gene sets correspond to any significant Hallmark GSEA?
2. Further modify the final GSEA ggplot for `C2.CP`. How would you improve readability with so many significant gene sets? Hint: could facets help?

Additional resources

Groups

- Rladies Seattle Not just for ladies! A pro-actively inclusive R community with both in-person and online workshops, hangouts, etc.
- TidyTuesday A weekly plotting challenge
- R code club Dr. Pat Schloss opened his lab's coding club to remote participation.
- Seattle useR Group

Online

- R cheatsheets also available in RStudio under Help > Cheatsheets
- The Carpentries
- topGO Another R package for running enrichment of gene ontology gene sets

R session

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/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] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] limma_3.44.3          fgsea_1.14.0          clusterProfiler_3.16.1
##  [4] msigdbr_7.4.1         forcats_0.5.1         stringr_1.4.0
##  [7] dplyr_1.0.7           purrr_0.3.4           readr_1.4.0
## [10] tidyr_1.1.3           tibble_3.1.2          ggplot2_3.3.5
## [13] tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##   [1] colorspace_2.0-2     ggridges_0.5.3       ellipsis_0.3.2
##   [4] qvalue_2.20.0        fs_1.5.0             rstudioapi_0.13
##   [7] farver_2.1.0         urltools_1.7.3       graphlayouts_0.7.1
##  [10] ggrepel_0.9.1        bit64_4.0.5          scatterpie_0.1.6
##  [13] AnnotationDbi_1.50.3 fansi_0.5.0          lubridate_1.7.10
##  [16] xml2_1.3.2           splines_4.0.2        cachem_1.0.5
##  [19] GOSemSim_2.14.2      knitr_1.33           polyclip_1.10-0
##  [22] jsonlite_1.7.2       broom_0.7.8          GO.db_3.11.4
##  [25] dbplyr_2.1.1         ggforce_0.3.3        BiocManager_1.30.16
##  [28] compiler_4.0.2       httr_1.4.2           rvcheck_0.1.8
##  [31] backports_1.2.1      assertthat_0.2.1     Matrix_1.3-4
##  [34] fastmap_1.1.0        cli_3.0.0            tweenr_1.0.2
##  [37] htmltools_0.5.1.1    prettyunits_1.1.1    tools_4.0.2
##  [40] igraph_1.2.6         gtable_0.3.0         glue_1.4.2
##  [43] reshape2_1.4.4       DO.db_2.9            fastmatch_1.1-0
##  [46] Rcpp_1.0.7           enrichplot_1.8.1     Biobase_2.48.0
##  [49] cellranger_1.1.0     vctrs_0.3.8          babelgene_21.4
##  [52] ggraph_2.0.5         xfun_0.24            rvest_1.0.0
##  [55] lifecycle_1.0.0      DOSE_3.14.0          europepmc_0.4
##  [58] MASS_7.3-54          scales_1.1.1         tidygraph_1.2.0
##  [61] hms_1.1.0            parallel_4.0.2       RColorBrewer_1.1-2
##  [64] yaml_2.2.1           memoise_2.0.0        gridExtra_2.3
##  [67] downloader_0.4       triebeard_0.3.0      stringi_1.6.2
##  [70] RSQLite_2.2.7        highr_0.9            S4Vectors_0.26.1
##  [73] BiocGenerics_0.34.0  BiocParallel_1.24.1  rlang_0.4.11
##  [76] pkgconfig_2.0.3      evaluate_0.14        lattice_0.20-44
##  [79] labeling_0.4.2       cowplot_1.1.1        bit_4.0.4
##  [82] tidyselect_1.1.1     plyr_1.8.6.9000      magrittr_2.0.1
##  [85] R6_2.5.0             IRanges_2.22.2       generics_0.1.0
##  [88] DBI_1.1.1            pillar_1.6.1         haven_2.4.1
##  [91] withr_2.4.2          modelr_0.1.8         crayon_1.4.1
##  [94] utf8_1.2.1           rmarkdown_2.9        viridis_0.6.1
##  [97] progress_1.2.2       grid_4.0.2           readxl_1.3.1
## [100] data.table_1.14.0    blob_1.2.1           reprex_2.0.0
## [103] digest_0.6.27        gridGraphics_0.5-1   stats4_4.0.2
## [106] munsell_0.5.0        viridisLite_0.4.0    ggplotify_0.0.7
```