How to use the SimpleCOMPASS Interface

Greg Finak

2019-01-04

Contents

1 Purpose . . . 2
1.1 Prerequisites . . . 2
2 Using SimpleCOMPASS . . . 2
2.1 Reading tabular ICS data . . . 2
2.2 Formatting the data . . . 3
2.3 Preparation . . . 3
2.4 Fit a COMPASS model . . . 7
3 Visualization of results . . . 7
References . . . 9

1 Purpose

This vignette describes how to use the `SimpleCOMPASS` interface in the `COMPASS` package to fit a polyfunctionality response model to tabular ICS data.

In contrast to `COMPASS()` which is tied more closely to the `GatingSet` data structures available in the core BioConductor flow cytometry infrastructure, `SimpleCOMPASS()` allows the user to used tabular cell count data (exported from FlowJo, for example) directly for `COMPASS` model fitting.

This approach requires a bit more care on the user's part, to ensure data are formatted and ordered correctly, but can can be less intimidating than working with the core flow infrastructure.

1.1 Prerequisites

We'll assume that the data are provided as an `excel` document. We'll use some of the data from the paper: Rakshit et al. (2017). The data are available in the COMPASS package as of version 1.17.2.

We need a good package to load the excel file, we'll use the tidyverse pacakge `readxl`. If it's not installed, you can install it with the command

```
library(BiocManager)
BiocManager::install("readxl")
```

```
library(COMPASS)
library(readxl)
```

2 Using SimpleCOMPASS

Let's get started.

2.1 Reading tabular ICS data

The data used in this vignette are installed along with the `COMPASS` package, and can be read into R as follows:

```
# retrieve the path to the data file in the package installation directory
example_data_path =
  system.file("extdata",
              "SimpleCOMPASSExample.xlsx",
              package = "COMPASS")
```

With the path to the data in hand we load the file.

```
compass_data = read_excel(example_data_path)
```

2.2 Formatting the data

Let's start by looking at the input data:

```
dim(compass_data)
#> [1]  42 260
colnames(compass_data)[1:5]
#> [1] "Stimulation"
#> [2] "PATIENT ID"
#> [3] "GROUP"
#> [4] "S/Avid-/L/CD3+/CD4+,Count"
#> [5] "S/Avid-/L/CD3+/CD4+/IFNg+2+10+17A+17F+22+MIP+TNF+,Count"
```

We see we have a table of 42 rows by 260 columns. The rows correspond to subjects identified by the `PATIENT ID` column under different `Stimulation` conditions from different treatment `GROUP`s. There are 256 (2^8) different boolean cell combinations defined by 8 different cytokine markers.

A peek at the data shows us that the first three columns correspond to `metadata`, while the remaining columns are `integer` cell counts (NOTE: Not proportions, `COMPASS` and `SimpleCOMPASS` require cell counts).

2.3 Preparation

We'll need to do several things to prepare the data to fit the model.

1. Separate out the metadata from the count data.
2. Split the count data matrix into two matrices: one for the **stimulated** cell counts, and one for the **non-stimulated** cell counts.
3. Construct and specify a sample-specific unique identifier that maps `metadata` rows to count data rows.
4. Name and order the rows of the count matrices so that they have the same ordering as the metadata and as each other (i.e. row 1 of the metadata corresponds to row 1 of the count data matrices, and so on).
5. Reformat the boolean cell population column names so that they are compatible with the expectations of COMPASS.
6. Ensure the last column of the count matrices corresponds to the non-stimulated boolean cell subset.

This seems like a lot of work, but it's relatively straightforward. When using the `COMPASS` interface, this is all handled for the user.

2.3.1 Step 1. Separate the metadata and count data

We assign the first three columns of the data set to a new `metadata` variable. These are the `PATIENT ID`, `Stimulation` and `Group` variables. We also want `metadata` to be a `data.frame` rather than a `tibble`.

```
metadata = compass_data[,1:3]
metadata = as.data.frame(metadata)
```

2.3.2 2. Split the count matrix into **stimulated** and **non-stimulated** counts.

```
# Take the remaining columns
counts = as.matrix(compass_data[,4:260])
dim(counts)
#> [1]  42 257
# NOTE that we still have too many columns.
# We should have 256, not 257.
# We'll fix this. The first column is the total. We'd need this for COMPASS, but not for SimpleCOMPASS.

# drop the total count
counts = counts[,-1L]
dim(counts)
#> [1]  42 256
```

Read the code comments above, there are some imporant details there.

Next we split the counts matrix, but we notice that we have a typo in our metadata `Stimulation` variable, we'll fix that and proceed.

We assign the stimulated counts to the new `n_s` variable and the unstimulated counts to the `n_u` variable.

```
unique(metadata$Stimulation)
#> [1] "ESAT6/CFP10"  "UnStimulated" "Unstimulated"

# Which entries have the typos
typos = which(metadata$Stimulation=="UnStimulated")

#Correct them
metadata$Stimulation[typos] = "Unstimulated"
# Better
unique(metadata$Stimulation)
#> [1] "ESAT6/CFP10"  "Unstimulated"

# old school R
n_u = subset(counts, metadata$Stimulation == "Unstimulated")
n_s = subset(counts, metadata$Stimulation == "ESAT6/CFP10")
```

2.3.3 3. A Unique Sample Specific Identifier

In this case, this is simple as each subject corresponds to a unique sample. In more complex cases where there are multiple timepoints, we would construct a unique identifier by concatenating the visit and the subject ids.

```
metadata$unique_id = metadata$`PATIENT ID`
```

2.3.4 4. Name and order rows of all the matrices.

The rows of `metadata`, `n_s` and `n_u` are all consistent, although `metadata` has twice as many rows as `n_s` or `n_u` since we split the counts matrix.

```
# assign consistent row names to n_u and n_s
rownames(n_u) = subset(metadata$unique_id,
       metadata$Stimulation=="Unstimulated")

rownames(n_s) = subset(metadata$unique_id,
       metadata$Stimulation=="ESAT6/CFP10")

# Now all matrices have the same dimensions and appropriate rownames
metadata = subset(metadata,
       metadata$Stimulation=="ESAT6/CFP10")
```

We have subset metadata to include only the stimulated rows. We have named the columns of the split count matrices according to the unique ids of the metadata matrix entries for the stimulated or non-stimulated entries, as appropriate.

2.3.5 5. Reformat cell population names.

Let's look at the current population names.

```
colnames(n_s)[1]
#> [1] "S/Avid-/L/CD3+/CD4+/IFNg+2+10+17A+17F+22+MIP+TNF+,Count"
```

There is a gating path prefix (we won't need that), and a trailing ",Count" string, presumably to specify that these are count values as oppposed to proportions. We won't need that either.

The cytokine names are in a short form (i.e. 17F corresponds to IL17F), and they use the "+/-" naming scheme to indicate cells that are positive or negative for a particular cytokine. This is not the format required by `COMPASS`.

We need to drop the path prefix, and use boolean operators to specify the cell combinations. For example "A+/B+C-" would translate to "A&B&!C", read as "A and B and NOT C".

COMPASS provides a convenience function to do this for you `translate_marker_names()`.

```
# remove the path
nms = basename(colnames(n_s))
# translate the marker names to a COMPASS compatible format.
nms = COMPASS:::translate_marker_names(nms)
sample(nms,2)
#> [1] "!IFNg&2&!10&!17A&17F&22&MIP&!TNF" "!IFNg&2&10&17A&17F&22&MIP&TNF"
colnames(n_s) = nms

nms = basename(colnames(n_u))
# translate the marker names to a COMPASS compatible format.
```

```
nms = COMPASS:::translate_marker_names(nms)
sample(nms,2)
#> [1] "!IFNg&!2&10&17A&17F&!22&!MIP&TNF"   "!IFNg&!2&!10&!17A&17F&22&!MIP&!TNF"
colnames(n_u) = nms
```

If we want, we can rename the cytokines to familiar form.

```
#2 to IL2
colnames(n_s) = gsub("([&!])2([&!])","\\1IL2\\2",colnames(n_s))
#10 to IL10
colnames(n_s) = gsub("([&!])10([&!])","\\1IL10\\2",colnames(n_s))
# 17A to IL17A
colnames(n_s) = gsub("([&!])17A([&!])","\\1IL17A\\2",colnames(n_s))

# 17F to IL17F
colnames(n_s) = gsub("([&!])17F([&!])","\\1IL17F\\2",colnames(n_s))

# 22 to IL22
colnames(n_s) = gsub("([&!])22([&!])","\\1IL22\\2",colnames(n_s))
```

We can put the above in a function:

```
rename_cytokines = function(nms){
  #2 to IL2
  nms = gsub("([&!])2([&!])",
       "\\1IL2\\2", nms)
  #10 to IL10
   nms = gsub("([&!])10([&!])",
        "\\1IL10\\2", nms)
  # 17A to IL17A
 nms = gsub("([&!])17A([&!])",
      "\\1IL17A\\2", nms)
  # 17F to IL17F
 nms = gsub("([&!])17F([&!])",
      "\\1IL17F\\2", nms)
 # 22 to IL22
 nms = gsub("([&!])22([&!])",
      "\\1IL22\\2", nms)
}
```

And apply it as so:

```
colnames(n_s) = rename_cytokines(colnames(n_s))
colnames(n_u) = rename_cytokines(colnames(n_u))
```

2.3.6 6. Last column should be the all-negative boolean combination.

The last column of the count matrices should be the all-negative cell subset. Let's confirm that is is.

```
colnames(n_s)[ncol(n_s)]
#> [1] "!IFNg&!IL2&!IL10&!IL17A&!IL17F&!IL22&!MIP&!TNF"
colnames(n_u)[ncol(n_u)]
#> [1] "!IFNg&!IL2&!IL10&!IL17A&!IL17F&!IL22&!MIP&!TNF"
```

2.4 Fit a COMPASS model

The arguments required by `SimpleCOMPASS` are:

```
args(COMPASS::SimpleCOMPASS)
#> function (n_s, n_u, meta, individual_id, iterations = 10000,
#>     replications = 8, verbose = TRUE)
#> NULL
```

We have already defined `n_s`, `n_u`, and `meta` (metadata). The `individual_id` is a character name of the variable for the unique_id, in our case "unique_id".

We'll limit the number of iterations and replications for the purpose of the vignette, so we won't have a good fit, but it's only for demonstration.

If you see a message about reordering the rows of n_s and n_u, double check your work, it likely means that the rownames of the count matrices don't match the `unique_id` variable, or are not consistent between each other.

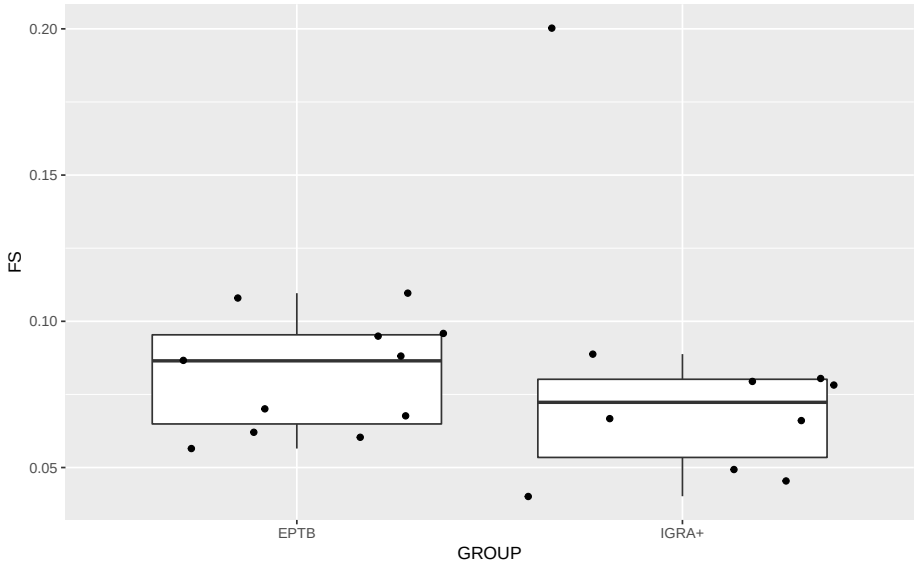
```
fit = COMPASS::SimpleCOMPASS(n_s = n_s,
                        n_u = n_u,
                        meta = metadata,
                        individual_id = "unique_id",
                        iterations = 1000,
                        replications = 3)
#> Initializing parameters...
#> Computing initial parameter estimates...
#> Iteration 1000 of 1000.
#> Fitting model with 3 replications.
#> Running replication 1 of 3...
#> Iteration 1000 of 1000.
#> Running replication 2 of 3...
#> Iteration 1000 of 1000.
#> Running replication 3 of 3...
#> Iteration 1000 of 1000.
#> Done!
```

For a complete run, iterations should be 400000, and replications should be 8 (the defaults for `COMPASS`).

3 Visualization of results

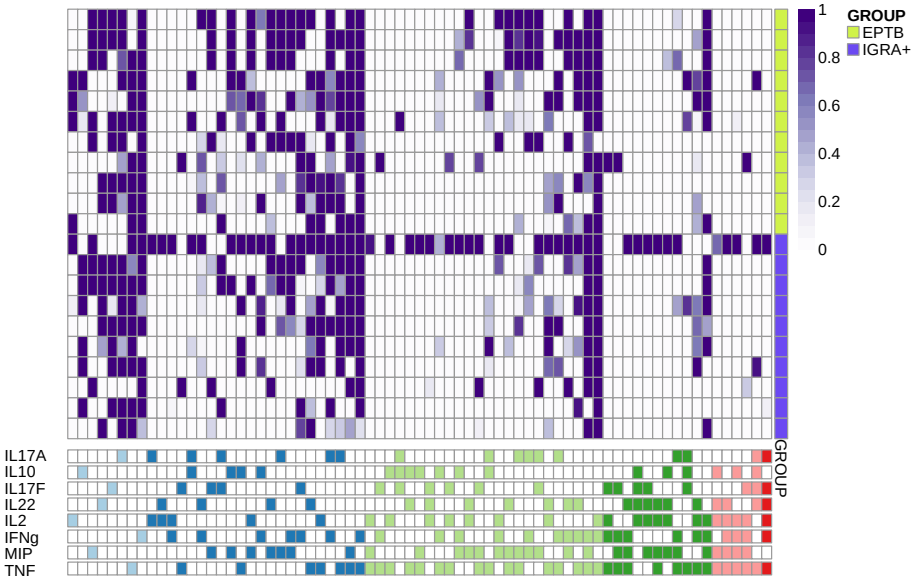
We can extract polyfunctionality scores with the `scores()` function. We look at the distribution of the functionality score (FS) across groups.

```
library(ggplot2)
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
scores(fit) %>%
  ggplot() +
  geom_boxplot(outlier.colour = NA) +
  geom_jitter() +
  aes(y = FS, x = GROUP)
```



A heatmap can be drawn using `plot()`.

```
plot(fit,"GROUP")
#> The 'threshold' filter has removed 184 categories:
#> IFNg&IL2&IL10&IL17A&IL17F&IL22&MIP&TNF, IFNg&IL2&IL10&IL17A&IL17F&IL22&MIP&!TNF, IFNg&IL2&IL10&IL17A&IL17
```



References

Rakshit, Srabanti, Vasista Adiga, Soumya Nayak, Pravat Nalini Sahoo, Prabhat Kumar Sharma, Krista E van Meijgaarden, Anto Jesuraj Uk J, et al. 2017. "Circulating Mycobacterium Tuberculosis DosR Latency Antigen-Specific, Polyfunctional, Regulatory IL10+ Th17 CD4 T-Cells Differentiate Latent from Active Tuberculosis." *Sci. Rep.* 7 (1):11948.