CellSpectra: Query - Reference analysis from a Seurat object

Konstantin A. Klötzer

2025-01-19

Getting started

CellSpectra compares the gene expression pattern of a function or pathway in a query with the same expression pattern in a reference. A p value indicates if a significant deviation from the expected gene expression in observed.

The following example shows the "gold standard" in CellSpectra analysis starting from a Seurat object (developed with Seurat V4). We realized that differences in cell numbers / expression sparsity can have an impact on single-cell analysis results. This can be significantly improved by performing an "onthefly" analysis. Thereby, reference samples match the total counts of each individual query sample.

```
start.time <- Sys.time()

library(dplyr)
library(CellSpectra)
library(Seurat)

#we should also define an output folder for our analysis
output_folder_base = "output/" #don't forget to add the final / !
```

loading data and gene sets

To run the "onthefly" version of CellSpectra we need two things:

1.) A seurat object storing our single cell raw counts of query and reference samples (Developmental version: Seurat V4)

CellSpectra expects specific metadata columns within this object and raw counts accessible as a matrix. We will prepare the object with one of the CellSpectra functions to make sure everything runs smooth.

2.) A gene set database with genes grouped into functionally-related pathways or functions.

CellSpectra expects the gene sets in a binary matrix of genes (rows) and gene sets (columns) with 0 and 1 indicating if a gene belongs to a gene set. You can import this matrix yourself or you use one of the CellSpectra functions to create such a matrix.

```
#here we use the example ifnb dataset from Seurat.data to show basic functions
seurat_object <- readRDS("ifnb.rds")

#now we adapt the metadata columns. CellSpectra needs to find the cell type
#annotation, sample identifier, conditions (query and reference). This can be
#done with the prepare_seurat_object function
```

```
seurat_object <- prepare_seurat_object(
  seurat_obj = seurat_object, #our object to prepare
  celltype_col = "seurat_annotations", #column in which annotation is stored
  sample_id_col = "orig.ident", #sample identifier
  condition_col = "stim", #column storing the condition information
  query_list = c("STIM"), #cat in the condition_col defining query samples
  control_list = c("CTRL") #cat in the condition_col defining the reference
)
```

Our object should now be ready for further analysis and should include all necessary information to proceed.

But first, we need to load or generate our gene set database. The easiest way to do that is the process_gene_sets function and a library .txt file downloaded from the enrichR website. Here we will use GO terms but any library can be downloaded and used.

```
dir.create(output_folder_base)

#prepare database
process_gene_sets(
  filename = "GO_Biological_Process_2023.txt", #downloaded file from enrichR
  seurat_object = seurat_object,
  output_folder_base = output_folder_base,
  min_genes = 10, #we don't want to run the analysis on tiny gene sets
  max_genes = 100 #you might want to set a max cut-off as well
)
```

Note that if you want to use another database you will have to save the gene sets in the right format as a file in the output_folder_base called "go_sets_modified.rds".

Optional: We also provide a function to generate the correct input gene set database from the GO website (process_gene_sets_from_GO). Check documentation. Again, you would have to save the file manually if using the run_spectra function.

creating the onthefly reference

Now we should have everything to create the onthefly reference. This can be quite memory intensive. If you are working with a large dataset, each cell type should be submitted separately. We are working on ways to make this more efficient in the future.

The function creates pseudobulk per sample and cell type. The onthefly version makes sure that total counts of these reference pseudobulk samples is approximately the total counts of the query. So if the query is quite small we wouldn't consider all cells of the reference samples (basically wasting money).

That's where the num_replicates become important. If the query is smaller than many of the reference samples, it makes sense to create a identical number of replicates per reference to use as many cells as possible. This improves the estimation of V1 (the reference expression pattern).

Rule of thumb: If a query sample has only half the cells (or total counts) compared to a reference sample, we will only use half the cells of the reference. So basically, we can make two replicates from the one reference sample. num_replicates = 2. If the size is the same or the query has even more cells, num_replicates should be one.

In the example below we will simply simulate more replicates by setting num_replicates = 10. This doesn't make sense from the statistical view. This is simply to demonstrate the basic functions.

This function will generate subfolders within the output folder for each cell type of interest (specify in cell_types) with subfolders for each query sample. The cell_number_threshold makes sure that only samples with a sufficient number of cells will be analyzed. We recommend at least 10 cells per sample. Higher numbers might increase robustness.

```
#create references
create_references(
  seurat_object = seurat_object,
  output_folder_base = output_folder_base,
  num_replicates = 10, #this dataset doesn't include any biological replicates
  cell_types = c("CD14 Mono", "CD4 Naive T", "CD4 Memory T"),
  cell_number_threshold = 10,
  seed = 123
)
```

Note: In a real world dataset CellSpectra benefits from a wide and heterogeneous reference representing the natural (technical and biological) variance across samples. The ifnb dataset is not really well suited for this kind of analysis without any biological replicates. The gene expression of any gene set is almost identical across our 10 sampling replicates. Therefore, the query will be significantly different from this pattern in most gene sets.

Running CellSpectra

If we followed these instructions we should have an output folder now containing everything we need to run spectra. While this can take quite long for large datasets and many cell types and gene sets, splitting the reference generation from the CellSpectra analysis will save resources (creating references is memory intensive, running spectra is not).

The following function we compute p values, fdr corrected p values, and R2 values for each cell type, query sample, and gene set. We report the QC summary for each query and set the expression threshold to -1 to not remove any low expressed genes

```
run_spectra(
  output_folder_base = output_folder_base,
  cell_types = c("CD14 Mono", "CD4 Naive T", "CD4 Memory T"), #parallelize
  CHISQ.MAX = 4, #this value cuts off the contribution of individual genes
  expression_threshold = -1, #this makes sure no genes are genes are filtered
  QC_report = TRUE

)
```

You can now check the csv results and use them for further downstream analysis.

Final Remarks

We hope this helps to run CellSpectra "onthefly" starting from a Seurat object and an enrichR library text file of any database. It's also possible to run CS from any (pseudo) bulk matrix without "onthefly" reference generation. We did that to compute the coordination within conditions (loo_coordination_from_matrix) or to analyse bulk RNA-seq data (run_spectra_from_matrix).

Check our Repositories for more information.

```r
end.time <- Sys.time()

execution_time <- end.time - start.time

print(execution_time)
```

```
## Time difference of 3.485023 mins
```

```r
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS 15.1.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/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] Seurat_5.0.3       SeuratObject_5.0.1 sp_2.1-3           CellSpectra_0.1.0
## [5] dplyr_1.1.4
##
## loaded via a namespace (and not attached):
##   [1] Rtsne_0.17             colorspace_2.1-0       deldir_2.0-4
##   [4] ellipsis_0.3.2         ggridges_0.5.6         XVector_0.38.0
##   [7] RcppHNSW_0.6.0         spatstat.data_3.0-4    rstudioapi_0.14
##  [10] leiden_0.4.3.1         listenv_0.9.1          ggrepel_0.9.3
##  [13] RSpectra_0.16-1        fansi_1.0.6            codetools_0.2-19
##  [16] splines_4.2.2          knitr_1.45             polyclip_1.10-6
##  [19] spam_2.10-0            jsonlite_1.8.4         ica_1.0-3
##  [22] cluster_2.1.4          png_0.1-8              uwot_0.1.14
##  [25] spatstat.sparse_3.0-3  sctransform_0.4.1      shiny_1.8.0
##  [28] compiler_4.2.2         httr_1.4.5             Matrix_1.6-5
##  [31] fastmap_1.1.1          lazyeval_0.2.2         cli_3.6.3
##  [34] later_1.3.0            htmltools_0.5.4        tools_4.2.2
##  [37] igraph_1.3.5           dotCall64_1.1-1        gtable_0.3.1
##  [40] glue_1.8.0             GenomeInfoDbData_1.2.9 reshape2_1.4.4
##  [43] RANN_2.6.1             Rcpp_1.0.10            scattermore_1.2
##  [46] vctrs_0.6.5            Biostrings_2.66.0      nlme_3.1-164
##  [49] spatstat.explore_3.2-7 progressr_0.14.0       lmtest_0.9-40
##  [52] spatstat.random_3.2-3  stringr_1.5.1          xfun_0.42
##  [55] globals_0.16.3         mime_0.12              miniUI_0.1.1.1
##  [58] lifecycle_1.0.4        irlba_2.3.5.1          goftest_1.2-3
##  [61] future_1.33.2          zlibbioc_1.44.0        MASS_7.3-58.2
##  [64] zoo_1.8-12             scales_1.3.0           spatstat.utils_3.0-4
##  [67] promises_1.2.0.1       parallel_4.2.2         RColorBrewer_1.1-3
##  [70] yaml_2.3.7             gridExtra_2.3          reticulate_1.34.0
```

```
##  [73] pbapply_1.7-2          ggplot2_3.5.0          stringi_1.8.4
##  [76] S4Vectors_0.36.1       fastDummies_1.7.3      BiocGenerics_0.44.0
##  [79] GenomeInfoDb_1.34.9    rlang_1.1.4            pkgconfig_2.0.3
##  [82] matrixStats_0.63.0     bitops_1.0-7           evaluate_0.20
##  [85] lattice_0.20-45        tensor_1.5             ROCR_1.0-11
##  [88] purrr_1.0.2            patchwork_1.2.0        htmlwidgets_1.6.1
##  [91] cowplot_1.1.3          tidyselect_1.2.1       parallelly_1.37.1
##  [94] RcppAnnoy_0.0.20       plyr_1.8.8             magrittr_2.0.3
##  [97] R6_2.5.1               IRanges_2.32.0         generics_0.1.3
## [100] withr_3.0.2            pillar_1.9.0           fitdistrplus_1.1-11
## [103] abind_1.4-5            survival_3.5-0         KEGGREST_1.38.0
## [106] RCurl_1.98-1.10        tibble_3.2.1           future.apply_1.11.1
## [109] crayon_1.5.2           KernSmooth_2.23-20     utf8_1.2.4
## [112] spatstat.geom_3.2-9    plotly_4.10.3          rmarkdown_2.20
## [115] grid_4.2.2             data.table_1.14.8      digest_0.6.31
## [118] xtable_1.8-4           tidyr_1.3.1            httpuv_1.6.14
## [121] stats4_4.2.2           munsell_0.5.0          viridisLite_0.4.1
```