

Multiome tutorial

Shang-Yang Chen*

Xiaosai Yao†

February 4th, 2022

Introduction

This tutorial walks through an example of TF activity inference in single cell multiome data. This is a dataset generated by infecting LNCaP cells with NKX2-1 and GATA6 and examine the effects of these TF on AR activity

Installation

```
devtools::install_github("OncoBioinfo-MO/Epiregulon", host = "https://github.roche.com/api/v3")
library(epiregulon)
library(SingleCellExperiment)
library(scater)
```

Data preparation

Please refer to the full ArchR manual for instructions

Before running Epiregulon, the following analyses need to be completed: 1. Obtain a peak matrix on scATACseq by using `addGroupCoverages > addReproduciblePeakSet > addPeakMatrix`. See chapter 10 from ArchR manual 2. RNAseq integration. a. For unpaired scATACseq, use `addGeneIntegrationMatrix`. See chapter [8] (<https://www.archrproject.com/bookdown/defining-cluster-identity-with-scRNA-seq.html>) from ArchR manual b. For multiome data, use `addGeneExpressionMatrix`. See multiome tutorial 3. Perform dimensionality reduction from with either single modalities or joint scRNAseq and scATACseq using `addCombinedDims`

Copy this ArchR project into your own directory

We verify that “GeneExpressionMatrix” and “PeakMatrix” are present for this tutorial.

```
getAvailableMatrices(proj)
#> [1] "GeneExpressionMatrix" "GeneScoreMatrix"      "NEPCMatrix"
#> [4] "PeakMatrix"          "TileMatrix"
```

We will use the joint reducedDims - “LSI_Combined” and joint embeddings - “UMAP_Combined”

*chens179@gene.com

†yaox19@gene.com

```
proj@reducedDims
#> List of length 3
#> names(3): LSI_ATAC LSI_RNA LSI_Combined
proj@embeddings
#> List of length 3
#> names(3): UMAP_Combined UMAP_RNA UMAP_ATAC
```

Retrieve gene expression matrix from the ArchR project

Change to SingleCellExperiment

Transfer cell and gene information and embeddings from ArchR project to singleCellExperiment

Visualize singleCellExperiment by UMAP

Quick start

1. Retrieve bulk TF ChIP-seq binding sites

First, we retrieve the information of TF binding sites collected from Cistrome and ENCODE ChIP-seq, which are hosted on Genomitory. Currently, human genomes HG19 and HG38 are available. We plan to add options for murine genomes and perhaps other model organisms in the near future.

```
grl <- getTFMotifInfo(genome="hg38")
head(grl)
#> GRangesList object of length 6:
#> $ADNP
#> GRanges object with 20545 ranges and 0 metadata columns:
#>           seqnames           ranges strand
#>           <Rle>             <IRanges> <Rle>
#> [1]      chr1      629819-630076      *
#> [2]      chr1      633892-634164      *
#> [3]      chr1      960443-960765      *
#> [4]      chr1     1011312-1012144      *
#> [5]      chr1     1058025-1058347      *
#> ...
#> [20541]   chrX 154136551-154137792      *
#> [20542]   chrX 154501525-154502608      *
#> [20543]   chrX 154751612-154752444      *
#> [20544]   chrX 155069184-155070016      *
#> [20545]   chrX 155218440-155219571      *
#> -----
#> seqinfo: 25 sequences from an unspecified genome; no seqlengths
#>
#> ...
#> <5 more elements>
```

2. Link ATACseq peaks to target genes

Next, we compute peak to gene correlations using the addPeak2GeneLinks function from ArchR package. The user would need to supply a path to an ArchR directory with a project contains peak and gene matrices, as well as already performed Latent semantic indexing (LSI) dimensionality reduction and integration with a scRNA-seq dataset.

```
# path to ArchR project
p2g <- getP2Glinks(myarchR_project_path, reducedDims = "LSI_Combined", useMatrix = "GeneExpressionMatrix")
```

```
#> Setting ArchRLogging = FALSE
#> 2022-03-01 11:11:41 : Getting Available Matrices, 0 mins elapsed.
#> No predictionScore found. Continuing without predictionScore!
#> 2022-03-01 11:11:41 : Filtered Low Prediction Score Cells (0 of 3903, 0), 0 mins elapsed.
#> 2022-03-01 11:11:41 : Computing KNN, 0.002 mins elapsed.
#> 2022-03-01 11:11:58 : Identifying Non-Overlapping KNN pairs, 0.279 mins elapsed.
#> 2022-03-01 11:12:02 : Identified 491 Groupings!, 0.348 mins elapsed.
#> 2022-03-01 11:12:03 : Getting Group RNA Matrix, 0.376 mins elapsed.
#> 2022-03-01 11:12:38 : Getting Group ATAC Matrix, 0.956 mins elapsed.
#> 2022-03-01 11:13:32 : Normalizing Group Matrices, 1.848 mins elapsed.
#> 2022-03-01 11:13:36 : Finding Peak Gene Pairings, 1.924 mins elapsed.
#> 2022-03-01 11:13:37 : Computing Correlations, 1.934 mins elapsed.
#> 2022-03-01 11:13:43 : Completed Peak2Gene Correlations!, 2.041 mins elapsed.
head(p2g)
#>   idxATAC Chrom idxRNA      Gene Correlation
#> 1    268  chr1    100      SKI    0.5467930
#> 2    484  chr1    171     RPL22    0.5035875
#> 3    499  chr1    181     ESPN    0.5192811
#> 4    513  chr1    181     ESPN    0.5149298
#> 5    627  chr1    207     UTS2    0.5067324
#> 6    628  chr1    200 AL359881.3    0.8701104
```

3. Add TF motif binding to peaks

The next step is to add the TF motif binding information by overlapping the regions of the peak matrix with the bulk chip-seq database loaded in 2. The user can supply either an archR project path and this function will retrieve the peak matrix, or a peakMatrix in the form of a Granges object or RangedSummarizedExperiment.

```
overlap= addTFMotifInfo(archR_project_path=archR_project_path, grl=grl, p2g=p2g)
#> Successfully loaded ArchRProject!
#> Computing overlap...
#> Success!
```

4. Generate regulons

A long format dataframe, representing the inferred regulons, is then generated. The dataframe consists of three columns:

- tf (transcription factor)
- target gene
- peak to gene correlation between tf and target gene

```
regulon <- getRegulon(p2g, overlap, aggregate=TRUE)
head(regulon)
#>      tf target      corr
#> 1    AR  AAK1 0.5284217
#> 2 ARID1B  AAK1 0.5284217
#> 3  ATF1  AAK1 0.5284217
#> 4  ATF3  AAK1 0.5284217
#> 5 ATOH8  AAK1 0.5284217
#> 6  BCL3  AAK1 0.5284217
```

EpiRegulon outputs two different correlations. The first, termed “corr”, is the correlation between chromatin accessibility of regulatory elements vs expression of target genes calculated by ArchR. The second, termed “weight”, can be generated by the addWeights function, which compute the correlation between gene expressions of TF vs expressions of target genes, shown below. The user is required to supply the clustering or batch

labels of the scRNA-seq dataset when running addWeights. “Weight” is the preferred metric for calculating activity.

5. Calculate TF activity

Finally, the activities for a specific TF in each cell are computed by averaging expressions of target genes linked to the TF weighted by the correlation variable of user’s choice.

$$y = \frac{1}{n} \sum_{i=1}^n x_i * corr_i$$

where y is the activity of a TF for a cell n is the total number of targets for a TF x_i is the log count expression of target i where i in $\{1, 2, \dots, n\}$ $corr_i$ is the weight of TF and target i

6. Perform differential activity

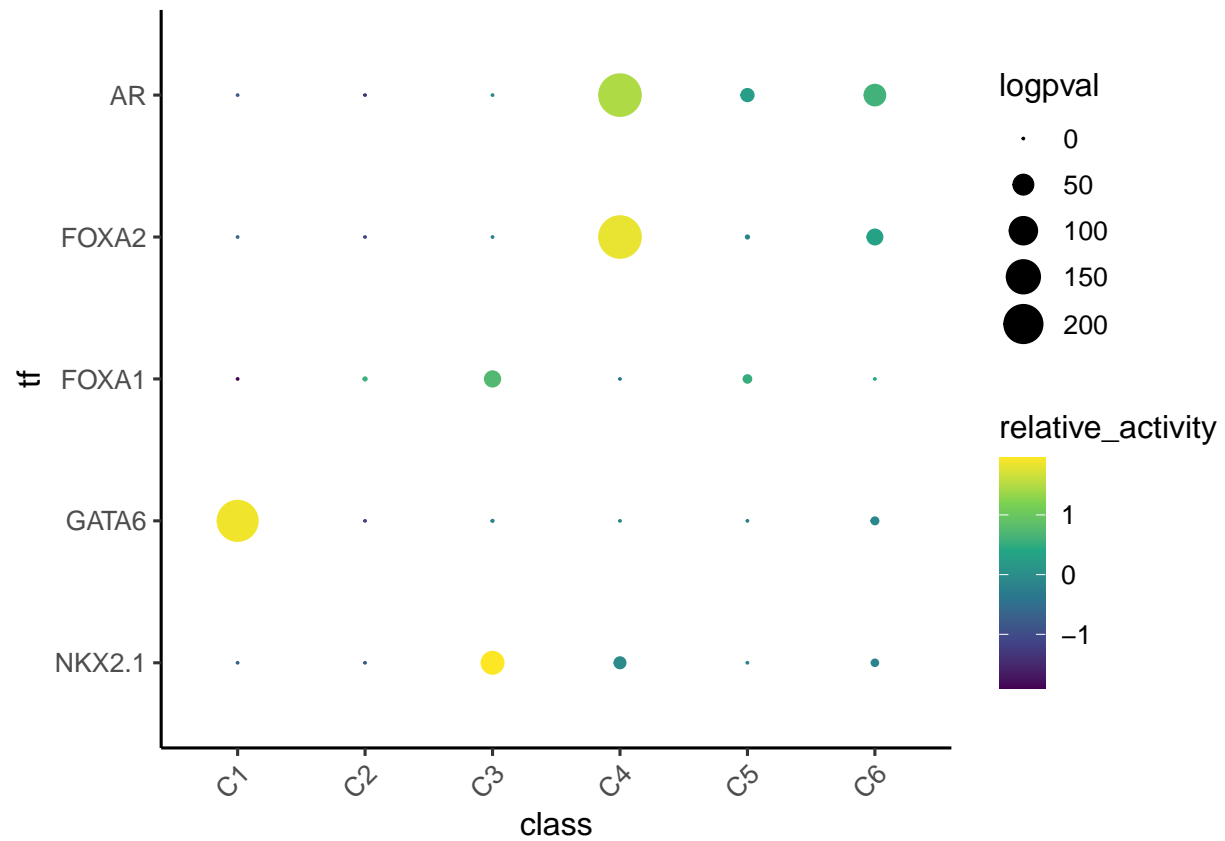
Take the top TFs

```
markers.sig <- getSigGenes(markers, topgenes = 5 )  
#> Using a logFC cutoff of 1 for class 1  
#> Using a logFC cutoff of 1 for class 2  
#> Using a logFC cutoff of 0.3 for class 3  
#> Using a logFC cutoff of 0.9 for class 4  
#> Using a logFC cutoff of 0.4 for class 5  
#> Using a logFC cutoff of 0.6 for class 6
```

7. Visualize the results

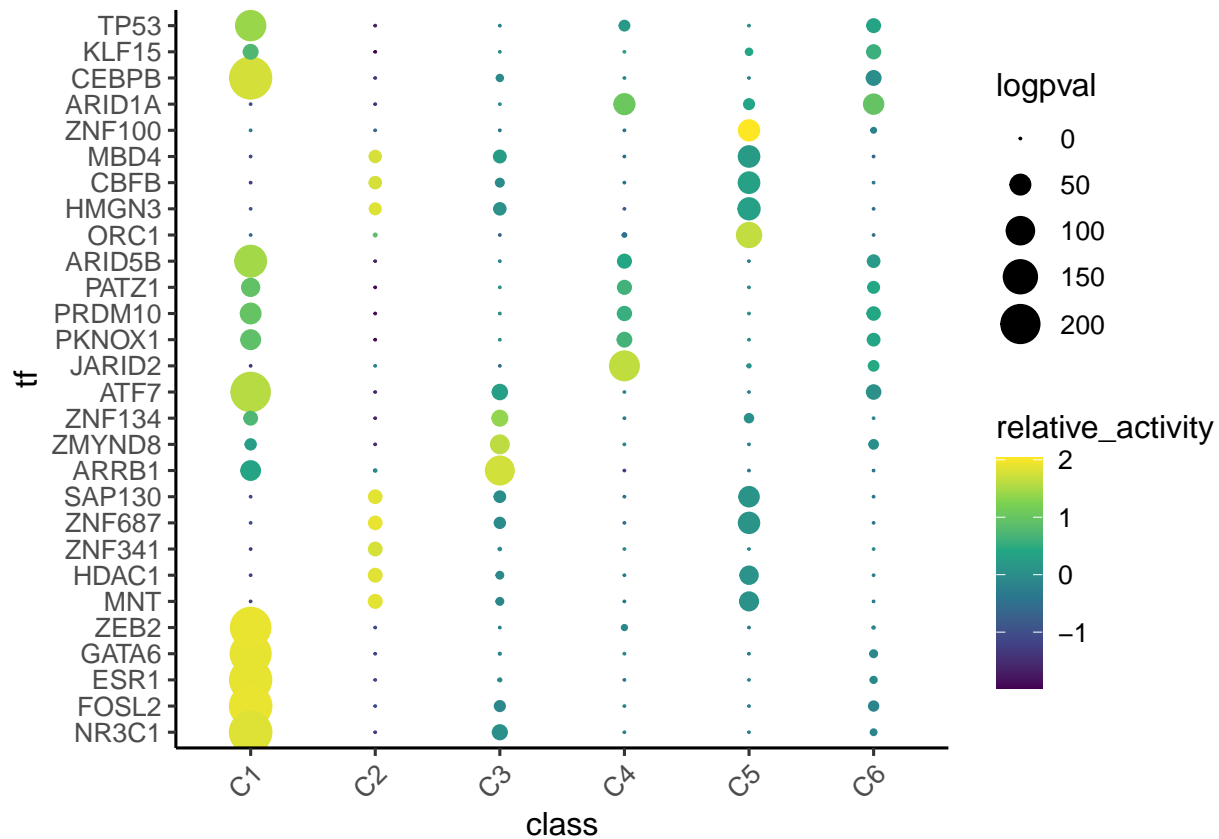
First visualize the known differential TFs by bubble plot

```
plotBubble(activity_matrix = score.combine, tf.list = c("NKX2-1", "GATA6", "FOXA1", "FOXA2", "AR"), class=
```



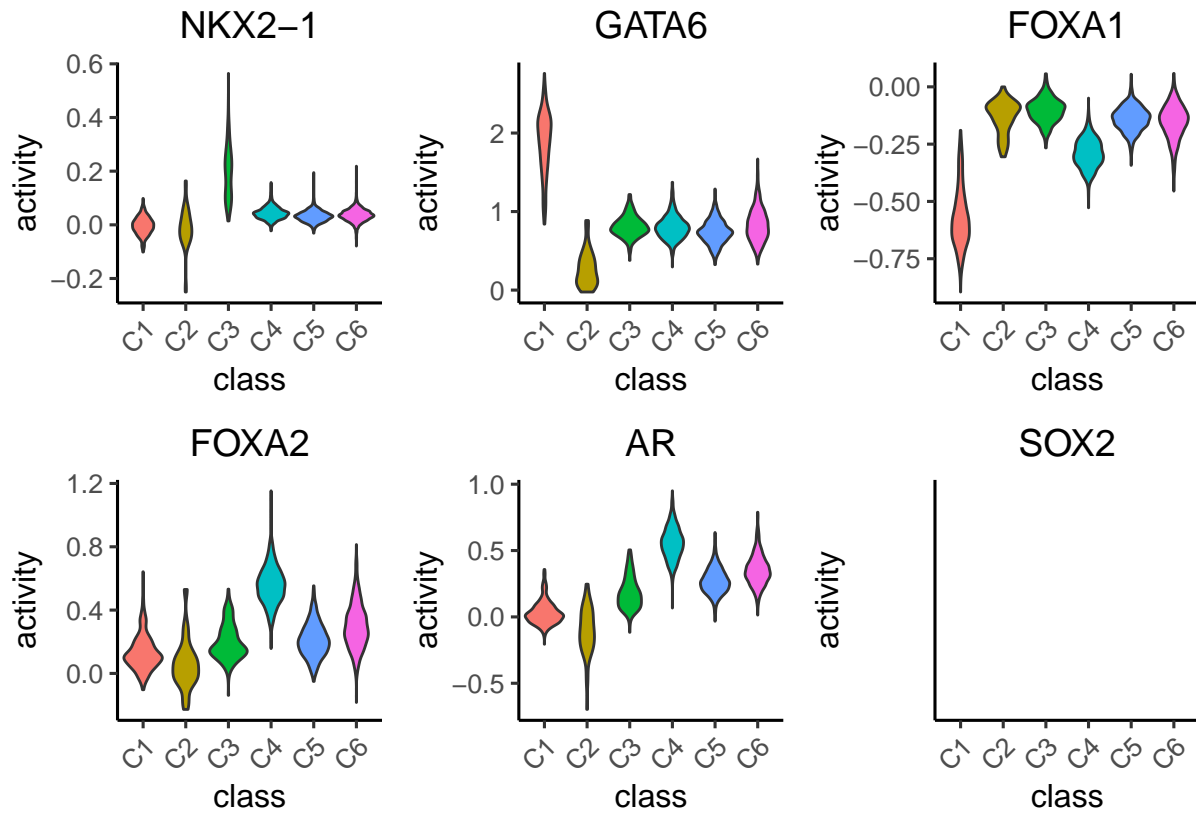
Then visualize the most differential TFs by clusters

```
plotBubble(activity_matrix = score.combine, tf.list = markers.sig$tf, class=GeneExpressionMatrix$Cluster
```



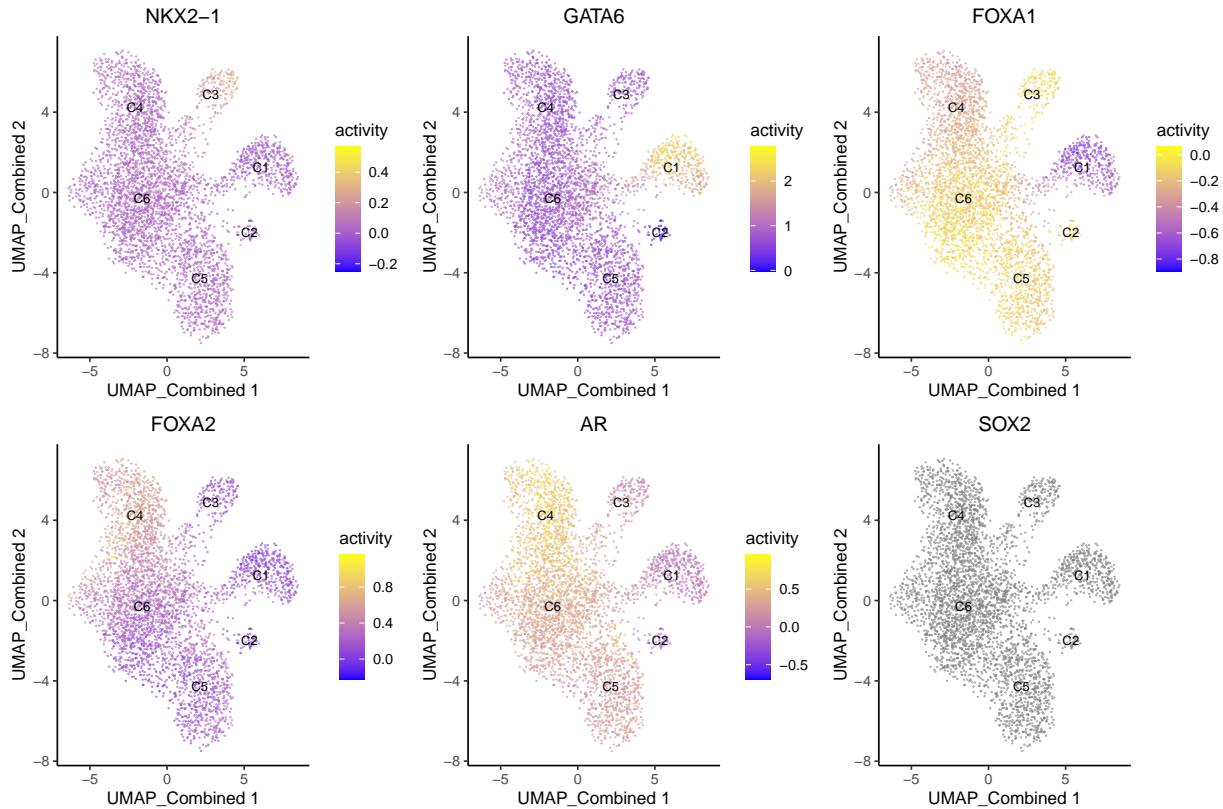
Visualize the known differential TFs by violin plot. Note there is no activity calculated for SOX2 because the expression of SOX2 is 0 in all cells.

```
plotActivityViolin(score.combine, c("NKX2-1", "GATA6", "FOXA1", "FOXA2", "AR", "SOX2"), GeneExpressionMatrix)
#> Warning: Removed 3903 rows containing non-finite values (stat_ydensity).
#> Warning in max(data$density): no non-missing arguments to max; returning -Inf
#> Warning: Computation failed in `stat_ydensity()`:
#> replacement has 1 row, data has 0
```



Visualize the known differential TFs by UMAP

```
plotActivityDim(sce=GeneExpressionMatrix, activity_matrix=score.combine, tf= c("NKX2-1", "GATA6", "FOXA1"))
```



8. Geneset enrichment Sometimes we are interested to know what pathways are enriched in the regulon of a particular TF. We can perform geneset enrichment using the `enricher` function from `clusterProfiler`.

```
#retrieve genesets
```

```
H = EnrichmentBrowser::getGenesets(org = "hsa", db = "msigdb", cat = "H", gene.id.type = "SYMBOL" )
C6 = EnrichmentBrowser::getGenesets(org = "hsa", db = "msigdb", cat = "C6", gene.id.type = "SYMBOL" )
```

```
#combine genesets and convert genesets to be compatible with enricher
```

```
gs=c(H,C6)
gs.list=do.call(rbind,lapply(names(gs), function(x) {data.frame(gs=x, genes=gs[[x]])}))
```

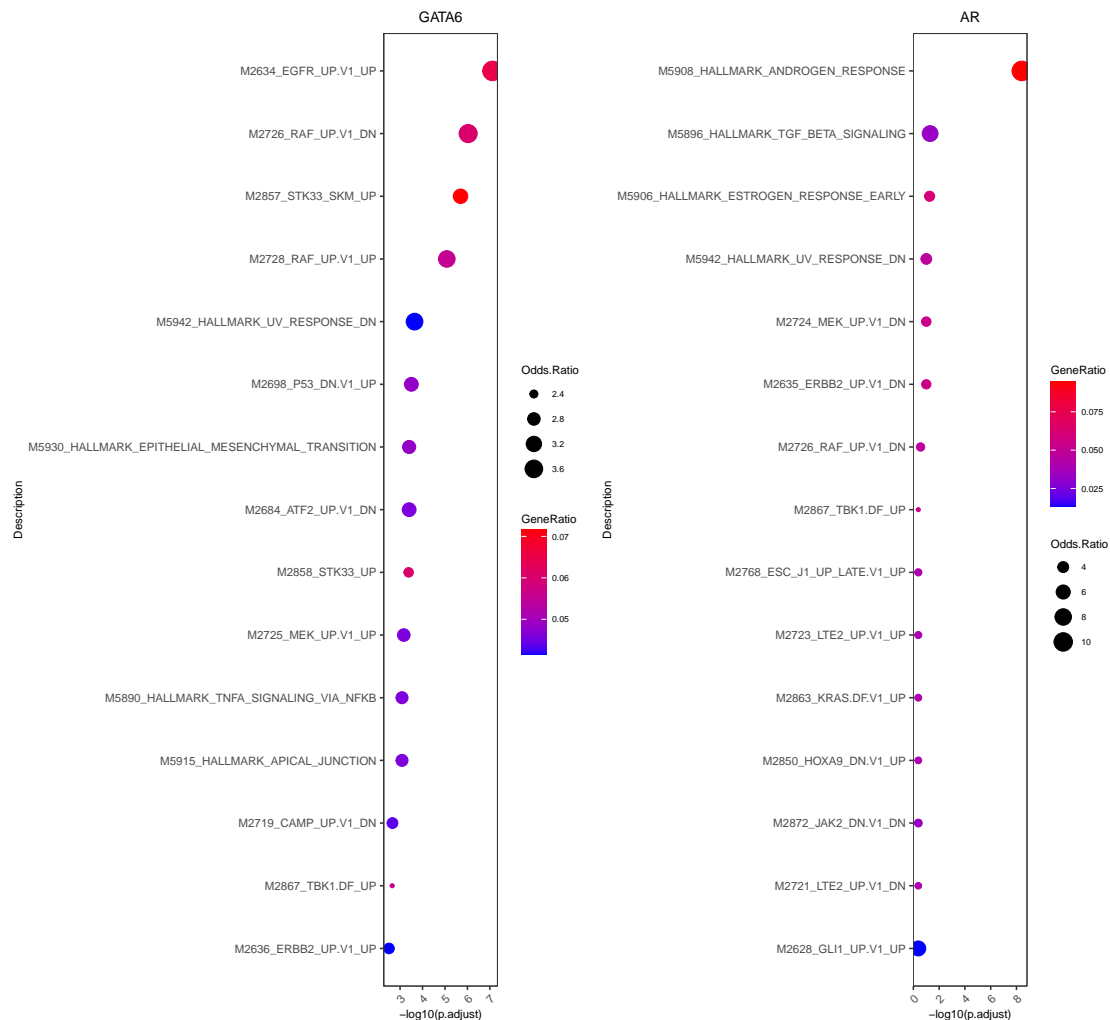
```
enrichresults=regulonEnrich(c("GATA6","AR"), regulon=regulon.w, corr="weight", corr_cutoff=0.5, geneset=gs)
#>
```

```
#plot results
```

```
enrichPlot(results=enrichresults)
```

```
#> Warning: `panel.margin` is deprecated. Please use `panel.spacing` property
#> instead
```

```
#> Warning: `panel.margin` is deprecated. Please use `panel.spacing` property
#> instead
```

Session Info

```
sessionInfo()
#> R version 4.1.0 (2021-05-18)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Ubuntu 18.04.5 LTS
#>
#> Matrix products: default
#> BLAS: /usr/local/lib/R/lib/libRblas.so
#> LAPACK: /usr/local/lib/R/lib/libRlapack.so
#>
#> locale:
#> [1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
#> [3] LC_TIME=en_US.UTF-8 LC_COLLATE=en_US.UTF-8
#> [5] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
#> [7] LC_PAPER=en_US.UTF-8 LC_NAME=C
#> [9] LC_ADDRESS=C LC_TELEPHONE=C
#> [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
```

```

#>
#> attached base packages:
#> [1] parallel stats4 stats graphics grDevices utils datasets
#> [8] methods base
#>
#> other attached packages:
#> [1] org.Hs.eg.db_3.13.0 AnnotationDbi_1.54.1
#> [3] msigdb_7.4.1 nabor_0.5.0
#> [5] scater_1.20.1 scuttle_1.2.1
#> [7] SingleCellExperiment_1.14.1 epiregulon_0.0.0.9000
#> [9] ArchR_1.0.2 magrittr_2.0.2
#> [11] rhdf5_2.36.0 Matrix_1.3-4
#> [13] data.table_1.14.0 SummarizedExperiment_1.22.0
#> [15] Biobase_2.52.0 GenomicRanges_1.44.0
#> [17] GenomeInfoDb_1.28.2 IRanges_2.26.0
#> [19] S4Vectors_0.30.0 BiocGenerics_0.38.0
#> [21] MatrixGenerics_1.4.3 matrixStats_0.60.1
#> [23] ggplot2_3.3.5
#>
#> loaded via a namespace (and not attached):
#> [1] utf8_1.2.2 tidyselect_1.1.1
#> [3] RSQLite_2.2.8 grid_4.1.0
#> [5] BiocParallel_1.26.2 scatterpie_0.1.7
#> [7] devtools_2.4.2 munsell_0.5.0
#> [9] ScaledMatrix_1.0.0 base64url_1.4
#> [11] codetools_0.2-18 statmod_1.4.36
#> [13] scran_1.20.1 withr_2.4.3
#> [15] colorspace_2.0-2 GOSeqSim_2.18.1
#> [17] genomitory_1.2.9 filelock_1.0.2
#> [19] highr_0.9 knitr_1.33
#> [21] rstudioapi_0.13 DOSE_3.18.2
#> [23] labeling_0.4.2 KEGGgraph_1.52.0
#> [25] GenomeInfoDbData_1.2.6 polyclip_1.10-0
#> [27] bit64_4.0.5 farver_2.1.0
#> [29] rprojroot_2.0.2 downloader_0.4
#> [31] treeio_1.16.2 vctrs_0.3.8
#> [33] generics_0.1.2 xfun_0.29
#> [35] BiocFileCache_2.0.0 R6_2.5.1
#> [37] graphlayouts_0.7.1 ggbeeswarm_0.6.0
#> [39] rsud_1.0.5 gp.version_1.2.1
#> [41] locfit_1.5-9.4 bitops_1.0-7
#> [43] rhdf5filters_1.4.0 cachem_1.0.6
#> [45] fgsea_1.18.0 DelayedArray_0.18.0
#> [47] assertthat_0.2.1 scales_1.1.1
#> [49] ggraph_2.0.5 enrichplot_1.12.2
#> [51] beeswarm_0.4.0 gtable_0.3.0
#> [53] beachmat_2.8.1 metacommons_1.4.0
#> [55] processx_3.5.2 tidygraph_1.2.0
#> [57] rlang_0.4.11 splines_4.1.0
#> [59] lazyeval_0.2.2 gp.auth_1.2.1
#> [61] yaml_2.2.2 reshape2_1.4.4
#> [63] backports_1.2.1 qvalue_2.24.0
#> [65] clusterProfiler_4.0.5 tools_4.1.0

```

```

#> [67] usethis_2.0.1           ellipsis_0.3.2
#> [69] RColorBrewer_1.1-2      sessioninfo_1.1.1
#> [71] Rcpp_1.0.7             plyr_1.8.6
#> [73] sparseMatrixStats_1.4.2 zlibbioc_1.38.0
#> [75] purrr_0.3.4            RCurl_1.98-1.4
#> [77] ps_1.6.0               prettyunits_1.1.1
#> [79] viridis_0.6.2          cowplot_1.1.1
#> [81] ggrepel_0.9.1          cluster_2.1.2
#> [83] fs_1.5.0               DO.db_2.9
#> [85] pkgload_1.2.1          patchwork_1.1.1
#> [87] evaluate_0.14          xtable_1.8-4
#> [89] XML_3.99-0.7           gridExtra_2.3
#> [91] testthat_3.0.4         compiler_4.1.0
#> [93] tibble_3.1.4           shadowtext_0.0.8
#> [95] crayon_1.5.0           gp.cache_1.2.7
#> [97] htmltools_0.5.2       ggfun_0.0.3
#> [99] aplot_0.0.6            ArtifactDB_1.4.12
#> [101] tidyrr_1.1.3           DBI_1.1.1
#> [103] tweenr_1.0.2           dbplyr_2.1.1
#> [105] MASS_7.3-54            rappdirs_0.3.3
#> [107] babelgene_21.4         cli_3.2.0
#> [109] metapod_1.0.0          igraph_1.2.6
#> [111] pkgconfig_2.0.3        getPass_0.2-2
#> [113] ggtree_3.0.4           annotate_1.70.0
#> [115] vipor_0.4.5            dqrng_0.3.0
#> [117] XVector_0.32.0         yulab.utils_0.0.2
#> [119] stringr_1.4.0          callr_3.7.0
#> [121] digest_0.6.29          graph_1.70.0
#> [123] Biostrings_2.60.2      rmarkdown_2.10
#> [125] fastmatch_1.1-3        tidytree_0.3.4
#> [127] edgeR_3.34.0           DelayedMatrixStats_1.14.3
#> [129] GSEABase_1.54.0        curl_4.3.2
#> [131] gtools_3.9.2           nlme_3.1-152
#> [133] lifecycle_1.0.0        jsonlite_1.7.3
#> [135] Rhdf5lib_1.14.2        BiocNeighbors_1.10.0
#> [137] desc_1.3.0             viridisLite_0.4.0
#> [139] limma_3.48.3           fansi_1.0.2
#> [141] pillar_1.6.2           lattice_0.20-44
#> [143] KEGGREST_1.32.0        fastmap_1.1.0
#> [145] httr_1.4.2             pkgbuild_1.2.0
#> [147] GO.db_3.13.0           glue_1.6.1
#> [149] remotes_2.4.0          png_0.1-7
#> [151] bluster_1.2.1          bit_4.0.4
#> [153] Rgraphviz_2.36.0       ggforce_0.3.3
#> [155] stringi_1.7.6          EnrichmentBrowser_2.22.2
#> [157] blob_1.2.2             BiocSingular_1.8.1
#> [159] memoise_2.0.0          dplyr_1.0.7
#> [161] ape_5.5                irlba_2.3.3

```