Multiome tutorial - MultiAssayExperiment

Xiaosai Yao

12 November 2022

Package

epiregulon 1.0.16

Contents

1 Introduction

This tutorial walks through the same dataset used in the "multiome tutorial - archR workflow". This is a dataset generated by infecting LNCaP cells with NKX2-1 and GATA6 to examine the effects of these TFs on AR activity.

2 Installation

Epiregulon is currently available on R/dev

```
library(epiregulon)
```

Alternatively, you could install from gitlab

```
devtools::install_gitlab(repo ='scwg/gene-transcriptional-network/activity-inference/Epiregulon',
                         auth_token = "<gitlab token>",
                         host = "https://code.roche.com/" )

library(epiregulon)
```

3 Data preparation

Single cell preprocessing needs to performed by user's favorite methods prior to using Epiregulon. The following components are required: 1. Peak matrix from scATAC-seq 2. Gene expression matrix from either paired or unpaired scRNA-seq. RNA-seq integration needs to be performed for unpaired dataset. 3. Dimensionality reduction matrix from with either single modalities or joint scRNA-seq and scATAC-seq

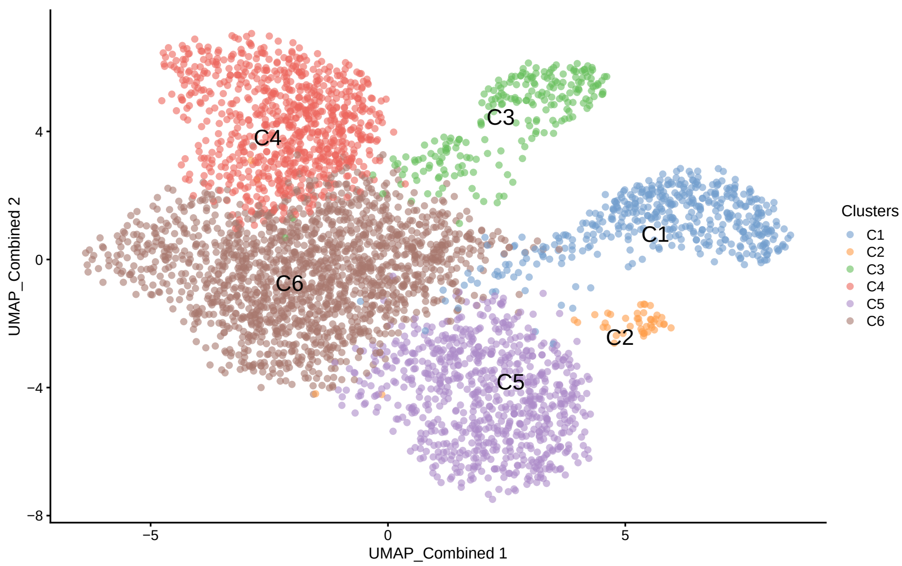
Multiome data can now be conveniently processed by `initiate.archr` and then `gp.sa.archr` to obtain peak matrices. Finally, the archR project can be uploaded into DatasetDB as a MultiAssayExperiment object using `maw.archr::importArchr` or `maw.archr::create.mae.with.multiple.sces.from.archr`

```
# load the MAE object
library(SingleCellExperiment)
mae <- dsassembly::getDataset("DS000013080")
#> 'version=' not specified, using the latest version (2) instead
#> Error in read_csv(path, is_compressed = identical(compression, "gzip"),  :
#>   encountered empty line in a file with non-zero columns
#> falling back to 'read.csv' for a malformed CSV data frame

# peak matrix
PeakMatrix <- mae[["PeakMatrix"]]
rownames(PeakMatrix) <- rowData(PeakMatrix)$idx

# expression matrix
GeneExpressionMatrix <- mae[["GeneExpressionMatrix"]]
rownames(GeneExpressionMatrix) <- rowData(GeneExpressionMatrix)$name
```

```
# dimensional reduction matrix
reducedDimMatrix <- reducedDim(mae[['TileMatrix500']], "LSI_ATAC")
```

Visualize singleCellExperiment by UMAP

```
# transfer UMAP_combined from TileMatrix to GeneExpressionMatrix
reducedDim(GeneExpressionMatrix, "UMAP_Combined") <- reducedDim(mae[['TileMatrix500']], "UMAP_Combined")
scater::plotReducedDim(GeneExpressionMatrix,
                       dimred = "UMAP_Combined",
                       text_by = "Clusters",
                       colour_by = "Clusters")
```



4 Quick start

4.1 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 and mouse mm10 are available.

```
grl <- getTFMotifInfo(genome = "hg38")
#> redirecting from 'GMTY162:hg38_motif_bed_granges@REVISION-4' to 'GMTY162:hg38_motif_bed_granges@24c22e4f42
head(grl)
#> GRangesList object of length 6:
#> $`5-hmC`
#> GRanges object with 24048 ranges and 0 metadata columns:
#>           seqnames          ranges strand
#>              <Rle>       <IRanges>  <Rle>
#>       [1]     chr1     10000-10685      *
#>       [2]     chr1     13362-13694      *
```

```
#>       [3]     chr1       29631-29989      *
#>       [4]     chr1       40454-40754      *
#>       [5]     chr1     135395-135871      *
#>       ...      ...               ...    ...
#>   [24044]     chrY 56864377-56864627      *
#>   [24045]     chrY 56876124-56876182      *
#>   [24046]     chrM           84-2450      *
#>   [24047]     chrM       13613-14955      *
#>   [24048]     chrM       15134-16490      *
#>   -------
#>   seqinfo: 25 sequences from an unspecified genome; no seqlengths
#>
#> ...
#> <5 more elements>
```

4.2 Link ATAC-seq peaks to target genes

Next, we compute peak to gene correlations using a custom algorithm that has similar performance to ArchR's P2G function.

```
p2g <- calculateP2G(peakMatrix = PeakMatrix,
                    expMatrix = GeneExpressionMatrix,
                    reducedDim = reducedDimMatrix)
#> Using epiregulon to compute peak to gene links...
#> performing k means clustering to form metacells
#> Computing correlation

head(p2g)
#>     idxATAC  chr  start    end idxRNA     target Correlation distance       FDR
#> 4         4 chr1 827287 827787     14 AL669831.2   0.6095423    66177 0.1751071
#> 33        5 chr1 858735 859235     17  LINC01128   0.6430039    30938 0.1197822
#> 6         6 chr1 869650 870150     14 AL669831.2   0.6050473   108540 0.1835652
#> 335       6 chr1 869650 870150     26     KLHL17   0.5380863    88432 0.3247173
#> 338       9 chr1 920452 920952     26     KLHL17   0.5747814    37630 0.2438505
#> 478       9 chr1 920452 920952     29 AL645608.7   0.5111153    75097 0.3868542
```

4.3 Add TF motif binding to peaks

The next step is to add the TF binding information by overlapping 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(grl = grl, p2g = p2g, peakMatrix = PeakMatrix)
#> Computing overlap...
#> Success!
head(overlap)
```

```
#>     idxATAC idxTF   tf
#> 161       4     2 5-mC
#> 162       4     8 AFF1
#> 163       4     9 AFF4
#> 164       4    10 AGO1
#> 165       4    11 AGO2
#> 166       4    14  AHR
```

4.4 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 = p2g, overlap = overlap, aggregate = FALSE)
head(regulon)
#>   idxATAC idxTF   tf  chr  start    end idxRNA     target      corr distance       FDR
#> 1       4     2 5-mC chr1 827287 827787     14 AL669831.2 0.6095423    66177 0.1751071
#> 2       4     8 AFF1 chr1 827287 827787     14 AL669831.2 0.6095423    66177 0.1751071
#> 3       4     9 AFF4 chr1 827287 827787     14 AL669831.2 0.6095423    66177 0.1751071
#> 4       4    10 AGO1 chr1 827287 827787     14 AL669831.2 0.6095423    66177 0.1751071
#> 5       4    11 AGO2 chr1 827287 827787     14 AL669831.2 0.6095423    66177 0.1751071
#> 6       4    14  AHR chr1 827287 827787     14 AL669831.2 0.6095423    66177 0.1751071
```

4.5 Network pruning (highly recommended)

Epiregulon prunes the network by performing tests of independence on the observed number of cells jointly expressing transcription factor (TF), regulatory element (RE) and target gene (TG) vs the expected number of cells if TF/RE and TG are independently expressed. We implement two tests, the binomial test and the chi-square test. In the binomial test, the expected probability is P(TF, RE) * P(TG), and the number of trials is the total number of cells, and the observed successes is the number of cells jointly expressing all three elements. In the chi-square test, the expected probability for having all 3 elements active is also P(TF, RE) * P(TG) and the probability otherwise is 1- P(TF, RE) * P(TG). The observed cell count for the active category is the number of cells jointly expressing all three elements, and the cell count for the inactive category is n - n_triple.

We calculate cluster-specific p-values if users supply cluster labels. This is useful if we are interested in cluster-specific networks. The pruned regulons can then be used to visualize differential networks for transcription factors of interest. See section on differential networks.

```
pruned.regulon <- pruneRegulon(expMatrix = GeneExpressionMatrix,
                               exp_assay = "counts",
                               peakMatrix = PeakMatrix,
```

```
                                peak_assay = "counts",
                                test = "binom",
                                regulon = regulon[regulon$tf %in% c("NKX2-1","GATA6","FOXA1","FOXA2","AR","SO
                                #regulon = regulon,
                                clusters = GeneExpressionMatrix$Clusters,
                                prune_value = "pval",
                                regulon_cutoff = 0.05,
                                BPPARAM = BiocParallel::MulticoreParam(workers = 2, progressbar = TRUE)
                                )
#> pruning network with binom tests using a regulon cutoff of pval<0.05
```

4.6 Add Weights

While the 'pruneRegulon' function provides statistics on the joint occurrence of TF-RE-TG, we would like to further estimate the strength of regulation. Biologically, this can be interpreted as the magnitude of gene expression changes induced by transcription factor activity. Epiregulon estimates the regulatory potential using one of the four measures: 1) correlation between TF and target gene expression, 2) mutual information between the TF and target gene expression, 3) Wilcoxon test statistics of target gene expression in cells jointly expressing all 3 elements vs cells that do not, or 4) log 2 fold difference of target gene expression in cells jointly expressing all 3 elements vs cells that do not.

Three measures (correlation, Wilcoxon statistics and log 2 fold difference) give both the magnitude and directionality of changes whereas mutational information is always positive. The correlation and mutual information statistics are computed on the grouped pseudobulks by user-supplied cluster labels, whereas the Wilcoxon and log fold change group cells based on the joint expression of TF, RE and TG in each single cell.

```
regulon.w <- addWeights(regulon = pruned.regulon,
                        expMatrix  = GeneExpressionMatrix,
                        exp_assay  = "counts",
                        peakMatrix = PeakMatrix,
                        peak_assay = "counts",
                        clusters = GeneExpressionMatrix$Clusters,
                        block_factor = NULL,
                        method = "corr")
#> adding weights using corr
#> calculating average expression across clusters...
```

```
head(regulon.w)
#>         idxATAC idxTF tf   chr     start       end idxRNA   target      corr distance
#> 1391357   46808    25 AR chr16  70289316  70289816  13581     AARS 0.5243981        0
#> 1392135   46809    25 AR chr16  70299292  70299792  13581     AARS 0.5306886     9550
#> 711058    24880    25 AR chr11 106121522 106122022   6438 AASDHPPT 0.7641509    43661
#> 2212899   69758    25 AR chr20  31698801  31699301  21689   ABALON 0.8866921    20204
#> 2214102   69759    25 AR chr20  31706227  31706727  21689   ABALON 0.6775096    12778
#> 3064485  108846    25 AR  chr7  48187607  48188107  31332   ABCA13 0.6459955    15950
#>                  FDR    pval_all    pval_C1 pval_C2   pval_C3    pval_C4   pval_C5
#> 1391357 0.3554204745 0.038119692 0.59075922       1 0.4163387 0.38798877 0.0614148
```

```
#> 1392135 0.3414011483 0.008479282 0.07209064       1 0.3070657 0.40283703 0.5195936
#> 711058  0.0146494992 0.187058621 1.00000000       1 0.1800487 1.00000000 1.0000000
#> 2212899 0.0001681256 0.002463565 0.76916971       1 0.1365022 0.46337799 0.2334431
#> 2214102 0.0752528210 0.015938884 1.00000000       1 0.4864779 0.03940726 1.0000000
#> 3064485 0.1155586402 0.065882736 1.00000000       1 0.0355546 1.00000000 1.0000000
#>            pval_C6 stats_all  stats_C1 stats_C2   stats_C3  stats_C4 stats_C5  stats_C6
#> 1391357 0.85582013  2.073565 0.5377361        0 -0.8127896 0.8632705 1.870497 0.1816975
#> 1392135 0.12912577  2.632365 1.7985453        0  1.0213984 0.8365652 0.643972 1.5175584
#> 711058  0.03650325  1.319331 0.0000000        0  1.3406052 0.0000000 0.000000 2.0912769
#> 2212899 0.49464715  3.027781 0.2934611        0  1.4889438 0.7332959 1.191536 0.6829362
#> 2214102 0.24442117  2.410312 0.0000000        0  0.6959217 2.0599088 0.000000 1.1640070
#> 3064485 1.00000000  1.839221 0.0000000        0  2.1019849 0.0000000 0.000000 0.0000000
#>         padj_all padj_C1 padj_C2 padj_C3 padj_C4 padj_C5 padj_C6     weight
#> 1391357        1       1       1       1       1       1       1  0.7300760
#> 1392135        1       1       1       1       1       1       1  0.7300760
#> 711058         1       1       1       1       1       1       1 -0.4385840
#> 2212899        1       1       1       1       1       1       1  0.1362656
#> 2214102        1       1       1       1       1       1       1  0.1362656
#> 3064485        1       1       1       1       1       1       1  0.1895212
```

4.7 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 test statistics of choice, chosen from either correlation, mutual information, Wilcoxon test statistics or log fold change.

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

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,...,n} $weights_i$ is the weight of TF and target i

```
score.combine <- calculateActivity(expMatrix = GeneExpressionMatrix,
                                   regulon = regulon.w,
                                   mode = "weight",
                                   method = "weightedMean",
                                   exp_assay = "counts")
#> calculating TF activity from regulon using weightedmean
```

4.8 Perform differential activity

```
markers <- findDifferentialActivity(activity_matrix = score.combine,
                                    groups = GeneExpressionMatrix$Clusters,
                                    pval.type = "some",
                                    direction = "up",
                                    test.type = "t")
```

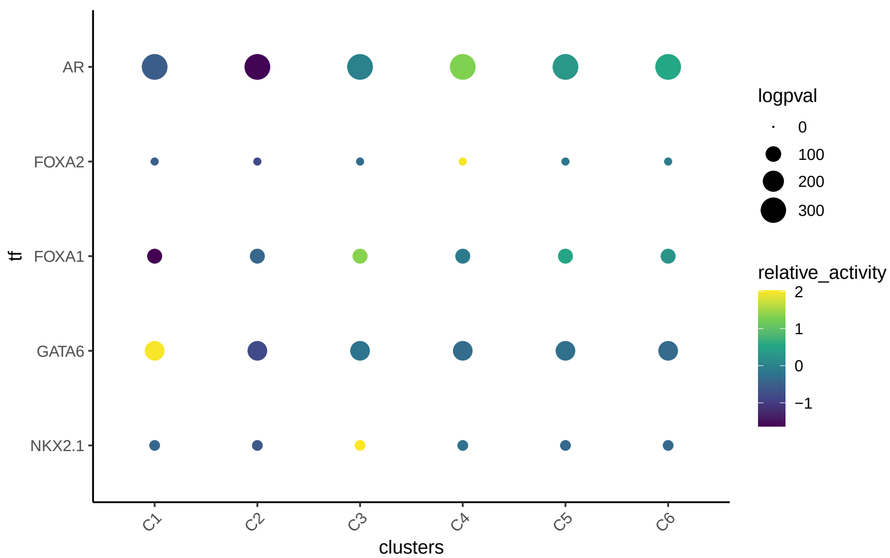
Take the top TFs

```
markers.sig <- getSigGenes(markers, topgenes = 5 )
#> Using a logFC cutoff of 0.4 for class C1
#> Using a logFC cutoff of 0 for class C2
#> Using a logFC cutoff of 0.3 for class C3
#> Using a logFC cutoff of 0.1 for class C4
#> Using a logFC cutoff of 0.1 for class C5
#> Using a logFC cutoff of 0 for class C6
```

4.9 Visualize the results

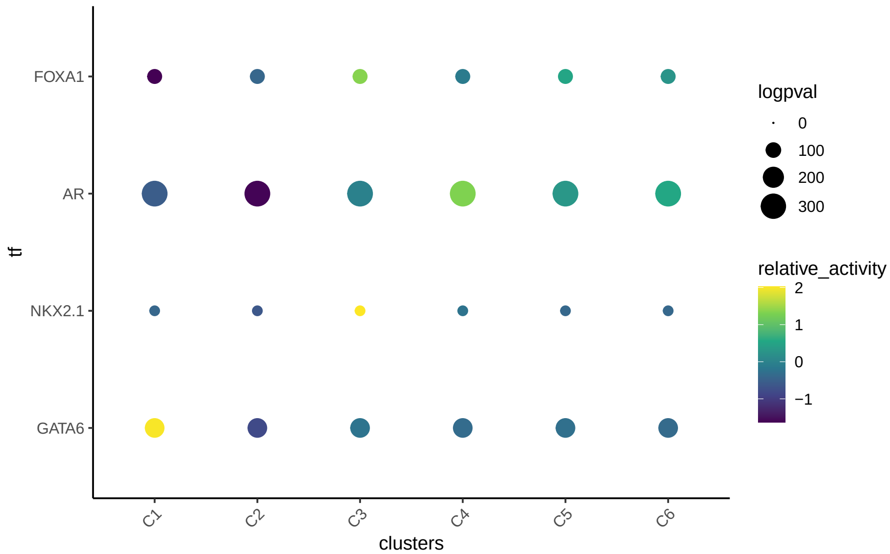
First visualize the known differential TFs by bubble plot

```
plotBubble(activity_matrix = score.combine,
           tf = c("NKX2-1","GATA6","FOXA1","FOXA2", "AR"),
           clusters = GeneExpressionMatrix$Clusters)
```



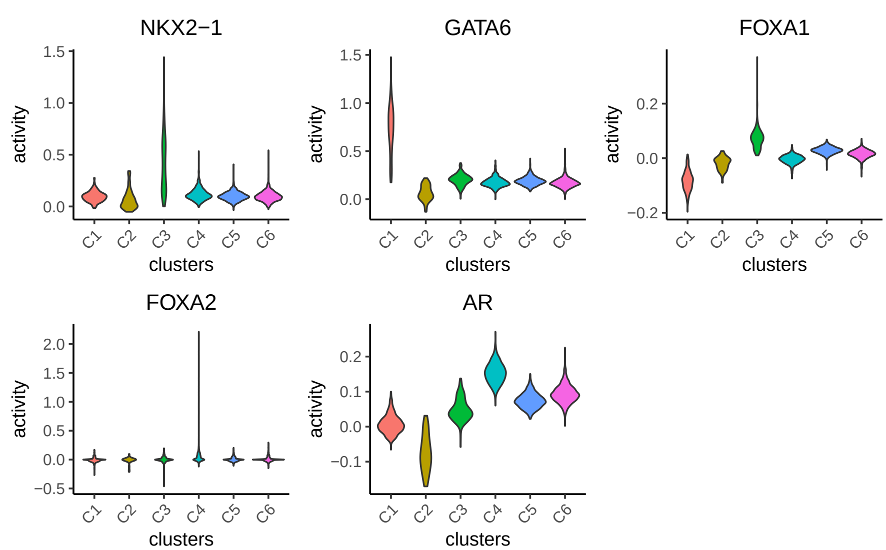
Then visualize the most differential TFs by clusters

```
plotBubble(activity_matrix = score.combine,
           tf = markers.sig$tf,
           clusters = GeneExpressionMatrix$Clusters)
```



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.

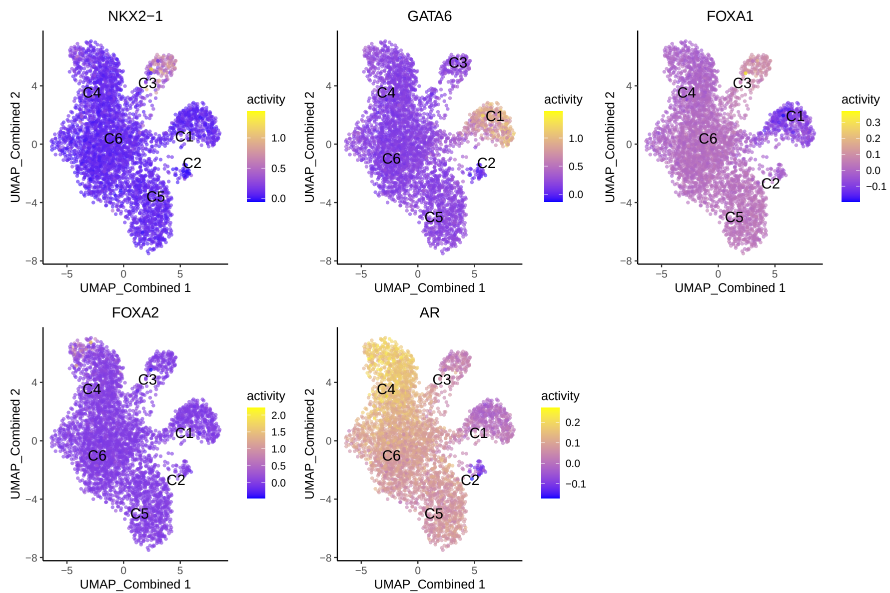
```
plotActivityViolin(activity_matrix = score.combine,
                   tf = c("NKX2-1","GATA6","FOXA1","FOXA2", "AR", "SOX2"),
                   clusters = GeneExpressionMatrix$Clusters)
#> SOX2 not found in activity matrix. Excluded from plots
```



Visualize the known differential TFs by UMAP

```
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = score.combine,
                tf = c("NKX2-1","GATA6","FOXA1","FOXA2", "AR", "SOX2"),
                dimtype = "UMAP_Combined",
                label = "Clusters",
```

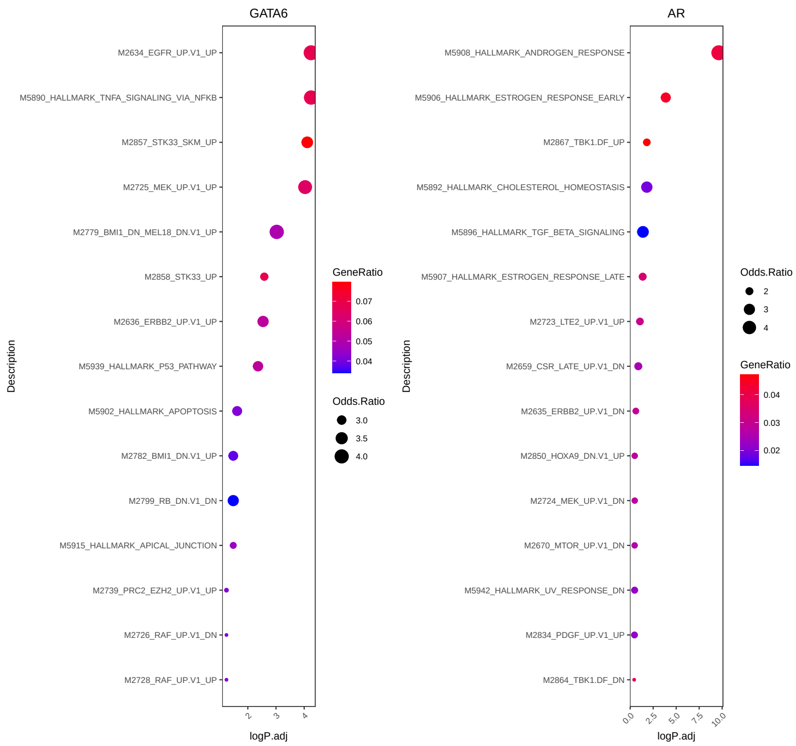
```
                point_size=1,
                ncol = 3)
#> SOX2 not found in activity matrix. Excluded from plots
```



4.10 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.

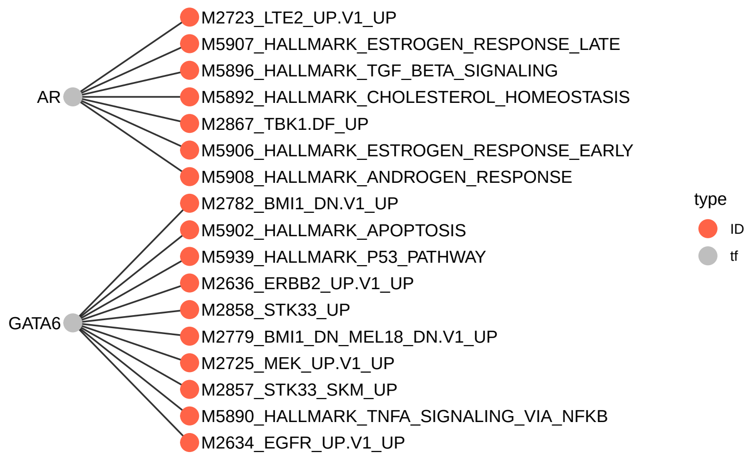
```
#retrieve genesets
H <- EnrichmentBrowser::getGenesets(org = "hsa", db = "msigdb", cat = "H",
                                    gene.id.type = "SYMBOL" )
#> Using cached version from 2022-11-11 22:43:20
C6 <- EnrichmentBrowser::getGenesets(org = "hsa", db = "msigdb", cat = "C6",
                                     gene.id.type = "SYMBOL" )
#> Using cached version from 2022-11-11 22:43:25

#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(TF = c("GATA6","AR"),
                               regulon = regulon.w,
                               corr = "weight",
                               corr_cutoff = 0.1,
                               genesets = gs.list)

#plot results
enrichPlot(results = enrichresults, )
```



4.11 Network analysis

We can visualize the genesets as a network

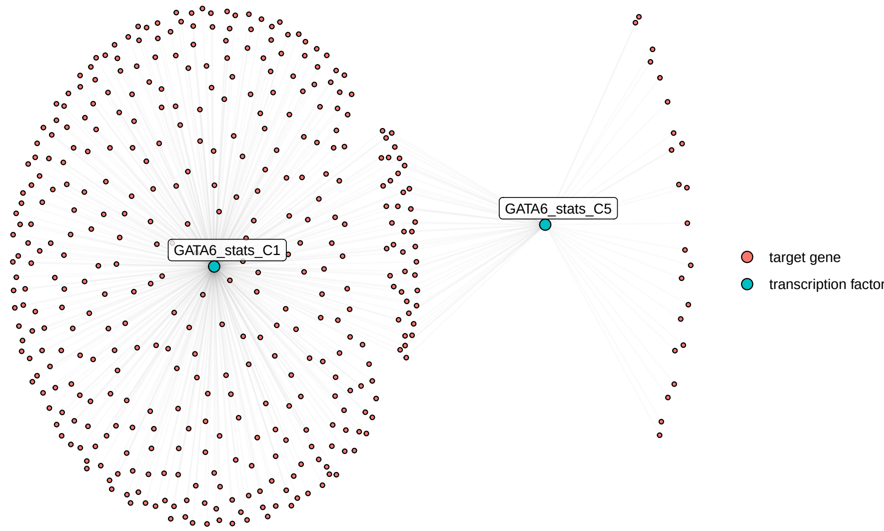
```
plotGseaNetwork(tf = names(enrichresults),
                enrichresults = enrichresults,
                p.adj_cutoff = 0.1,
                ntop_pathways = 10)
```



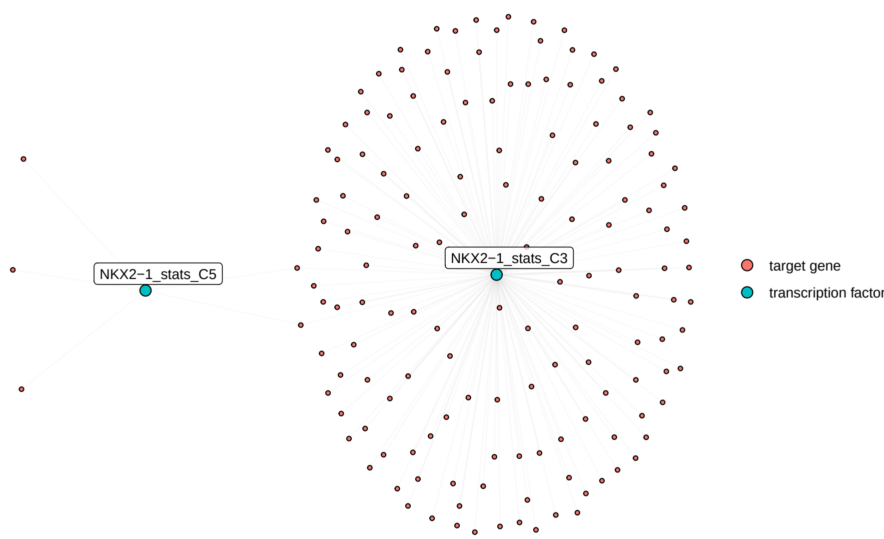
4.12 Differential networks

We are interested in understanding the differential networks between two conditions and determining which transcription factors account for the differences in the topology of networks. The pruned regulons with cluster-specific test statistics computed by `pruneRegulon` can be used to generate cluster-specific networks based on user-defined cutoffs and to visualize differential networks for transcription factors of interest. In this dataset, the GATA6 gene was only expressed in cluster 1 (C1) and NKX2-1 was only expressed in cluster 3 (C3). If we visualize the target genes of GATA6, we can see that C1 has many more target genes of GATA6 compared to C5, a cluster that does not express GATA6. Similarly, NKX2-1 target genes are confined to C3 which is the only cluster that exogenously expresses NKX2-1.

```
plotDiffNetwork(pruned.regulon,
                        cutoff = 1,
                        tf = c("GATA6"),
                        groups  = c("stats_C1","stats_C5"),
                        layout = "stress")
```



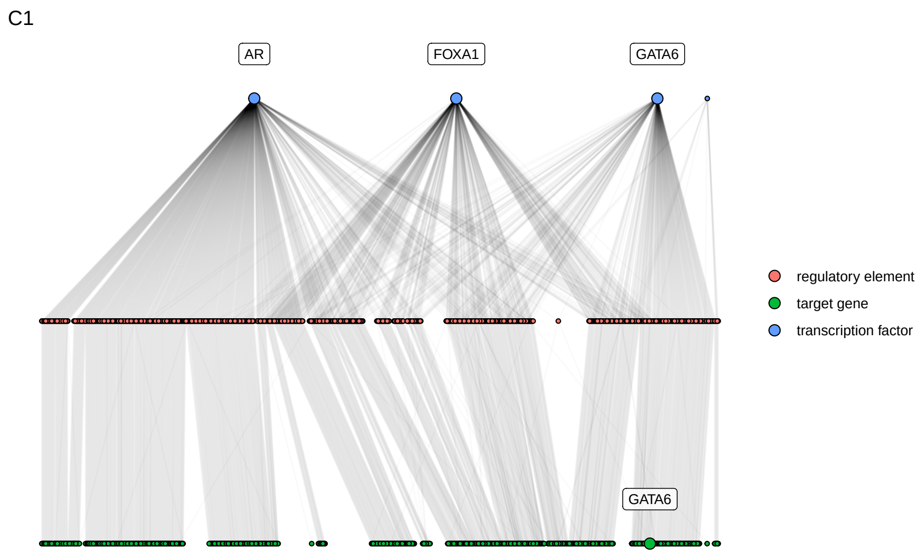

```
plotDiffNetwork(pruned.regulon,
                        cutoff = 1,
                        tf = c("NKX2-1"),
                        groups  = c("stats_C3","stats_C5"),
                        layout = "stress")
```



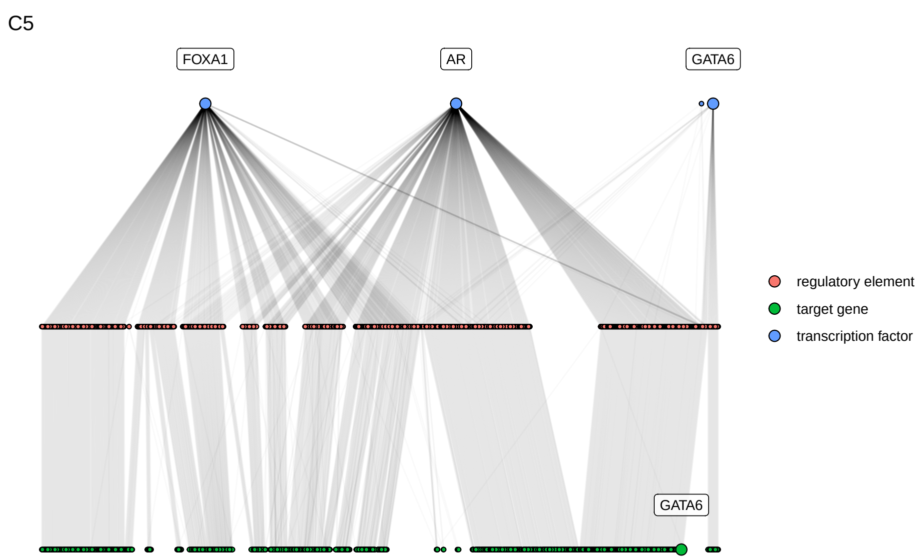
We can also visualize how transcription factors relate to other transcription factors in each cluster.

```
C1_network <- buildGraph(pruned.regulon[pruned.regulon$stats_C1>1,], weights = "stats_C1")
C5_network <- buildGraph(pruned.regulon[pruned.regulon$stats_C5>1,], weights = "stats_C5")

plotEpiregulonNetwork(C1_network,
                        layout = "sugiyama",
```

```
                         tfs_to_highlight = c("GATA6","FOXA1","AR")) + ggplot2::ggtitle ("C1")
```

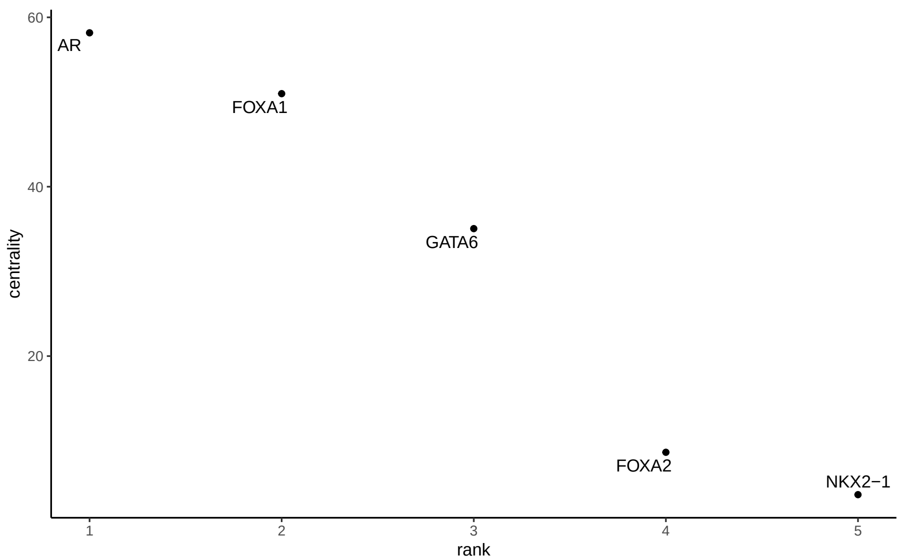



```
plotEpiregulonNetwork(C5_network,
                         layout = "sugiyama",
                         tfs_to_highlight = c("GATA6","FOXA1","AR")) + ggplot2::ggtitle ("C5")
```



To systematically examine the differential network topology between two clusters, we perform an edge subtraction between two graphs, using weights computed by `pruneRegulon`. We then calculate the degree centrality of the weighted differential graphs and if desired, normalize the differential centrality against the total number of edges. The default normalization function is `sqrt` as it preserves both the difference in the number of edges (but scaled by sqrt) and the differences in the weights. If the user only wants to examine the differences in the averaged weights, the `FUN` argument can be changed to `identity`. Finally, we rank the transcription factors by (normalized) differential centrality.

```
# rank by differential centrality
C1_network <- buildGraph(pruned.regulon, weights = "stats_C1")
C5_network <- buildGraph(pruned.regulon, weights = "stats_C5")

diff_graph <- buildDiffGraph(C1_network, C5_network)
diff_graph <- addCentrality(diff_graph)
diff_graph <- normalizeCentrality(diff_graph)
rank_table <- rankTfs(diff_graph)

library(ggplot2)
ggplot(rank_table, aes(x = rank, y = centrality)) +
    geom_point() +
    ggrepel::geom_text_repel(data = head(rank_table, 10), aes(label = tf)) +
    theme_classic()
```



5 Session Info

```
sessionInfo()
#> R version 4.2.0 (2022-04-22)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Ubuntu 18.04.6 LTS
#>
#> Matrix products: default
#> BLAS:   /usr/local/lib/R/lib/libRblas.so
#> LAPACK: /usr/local/lib/R/lib/libRlapack.so
#>
#> Random number generation:
#>  RNG:     L'Ecuyer-CMRG
#>  Normal:  Inversion
```

```
#>  Sample:  Rejection
#>
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C         LC_TIME=C            LC_COLLATE=C
#>  [5] LC_MONETARY=C        LC_MESSAGES=C        LC_PAPER=C           LC_NAME=C
#>  [9] LC_ADDRESS=C         LC_TELEPHONE=C       LC_MEASUREMENT=C     LC_IDENTIFICATION=C
#>
#> attached base packages:
#>  [1] parallel  grid      stats4    stats     graphics  grDevices utils     datasets
#>  [9] methods   base
#>
#> other attached packages:
#>  [1] nabor_0.5.0                 rhdf5_2.42.0                Rcpp_1.0.9
#>  [4] Matrix_1.5-3                data.table_1.14.4           stringr_1.4.0
#>  [7] plyr_1.8.8                  magrittr_2.0.3              gtable_0.3.1
#> [10] gtools_3.9.3                gridExtra_2.3               ArchR_1.0.2
#> [13] ggplot2_3.4.0               msigdbr_7.5.1               epiregulon_1.0.16
#> [16] SingleCellExperiment_1.20.0 SummarizedExperiment_1.29.1 Biobase_2.58.0
#> [19] GenomicRanges_1.50.1        GenomeInfoDb_1.34.3         IRanges_2.32.0
#> [22] S4Vectors_0.36.0            BiocGenerics_0.44.0         MatrixGenerics_1.10.0
#> [25] matrixStats_0.62.0          BiocStyle_2.26.0
#>
#> loaded via a namespace (and not attached):
#>   [1] utf8_1.2.2                  tidyselect_1.2.0            RSQLite_2.2.18
#>   [4] AnnotationDbi_1.60.0        BiocParallel_1.32.1         scatterpie_0.1.8
#>   [7] munsell_0.5.0               ScaledMatrix_1.6.0          base64url_1.4
#>  [10] codetools_0.2-18            statmod_1.4.37              scran_1.26.0
#>  [13] withr_2.5.0                 colorspace_2.0-3            GOSemSim_2.24.0
#>  [16] genomitory_2.1.6            filelock_1.0.2              knitr_1.40
#>  [19] rstudioapi_0.13             DOSE_3.23.3                 labeling_0.4.2
#>  [22] KEGGgraph_1.58.0            GenomeInfoDbData_1.2.9      polyclip_1.10-4
#>  [25] bit64_4.0.5                 farver_2.1.1                downloader_0.4
#>  [28] treeio_1.22.0               vctrs_0.5.0                 generics_0.1.3
#>  [31] gson_0.0.9                  xfun_0.31                   BiocFileCache_2.6.0
#>  [34] R6_2.5.1                    graphlayouts_0.8.3          ggbeeswarm_0.6.0
#>  [37] rsvd_1.0.5                  gp.version_1.5.0            locfit_1.5-9.6
#>  [40] artificer.matrix_1.3.7      gridGraphics_0.5-1          bitops_1.0-7
#>  [43] rhdf5filters_1.10.0         cachem_1.0.6                fgsea_1.24.0
#>  [46] DelayedArray_0.24.0         assertthat_0.2.1            scales_1.2.1
#>  [49] ggraph_2.1.0                enrichplot_1.18.0           beeswarm_0.4.0
#>  [52] beachmat_2.14.0             Cairo_1.6-0                 metacommons_1.9.0
#>  [55] tidygraph_1.2.2             rlang_1.0.6                 splines_4.2.0
#>  [58] lazyeval_0.2.2              dsdb.schemas_0.99.1         checkmate_2.1.0
#>  [61] gp.auth_1.7.0               BiocManager_1.30.19         yaml_2.3.5
#>  [64] reshape2_1.4.4              backports_1.4.1             rsconnect_0.8.28
#>  [67] qvalue_2.30.0               clusterProfiler_4.6.0       tools_4.2.0
#>  [70] bookdown_0.30               ggplotify_0.1.0             RColorBrewer_1.1-3
#>  [73] artificer.base_1.3.19       MultiAssayExperiment_1.24.0 sparseMatrixStats_1.10.0
#>  [76] zlibbioc_1.44.0             purrr_0.3.5                 RCurl_1.98-1.9
#>  [79] artificer.ranges_1.3.4      viridis_0.6.2               cowplot_1.1.1
#>  [82] ShadowArray_1.7.1           ggrepel_0.9.2               cluster_2.1.3
```

```
#>  [85] tinytex_0.42            patchwork_1.1.2            evaluate_0.18
#>  [88] GSVA_1.46.0             xtable_1.8-4               HDO.db_0.99.0
#>  [91] XML_3.99-0.12           artificer.schemas_0.99.2   artificer.sce_1.3.4
#>  [94] compiler_4.2.0          scater_1.26.0              tibble_3.1.8
#>  [97] shadowtext_0.1.2        crayon_1.5.1               gp.cache_1.7.1
#> [100] htmltools_0.5.3         ggfun_0.0.8                aplot_0.1.8
#> [103] tidyr_1.2.1             ArtifactDB_1.9.5           DBI_1.1.3
#> [106] tweenr_2.0.2            dbplyr_2.2.1               MASS_7.3-58.1
#> [109] rappdirs_0.3.3          babelgene_22.9             cli_3.4.1
#> [112] artificer.mae_1.3.4     genomitory.schemas_0.99.0  metapod_1.6.0
#> [115] igraph_1.3.5            pkgconfig_2.0.3            getPass_0.2-2
#> [118] dsdb.plus_1.3.2         scuttle_1.8.0              ggtree_3.6.2
#> [121] annotate_1.76.0         vipor_0.4.5                dqrng_0.3.0
#> [124] XVector_0.38.0          yulab.utils_0.0.5          digest_0.6.29
#> [127] graph_1.76.0            Biostrings_2.66.0          rmarkdown_2.18
#> [130] fastmatch_1.1-3         tidytree_0.4.1             artificer.se_1.3.4
#> [133] edgeR_3.40.0            DelayedMatrixStats_1.20.0  GSEABase_1.60.0
#> [136] curl_4.3.2              nlme_3.1-160               lifecycle_1.0.3
#> [139] jsonlite_1.8.3          Rhdf5lib_1.20.0            dsassembly_1.7.6
#> [142] BiocNeighbors_1.16.0    viridisLite_0.4.1          limma_3.54.0
#> [145] fansi_1.0.3             pillar_1.8.1               lattice_0.20-45
#> [148] KEGGREST_1.38.0         fastmap_1.1.0              httr_1.4.3
#> [151] GO.db_3.16.0            glue_1.6.2                 png_0.1-7
#> [154] bluster_1.8.0           bit_4.0.4                  Rgraphviz_2.42.0
#> [157] ggforce_0.4.1           stringi_1.7.6              HDF5Array_1.26.0
#> [160] BiocBaseUtils_1.1.0     EnrichmentBrowser_2.28.0   blob_1.2.3
#> [163] BiocSingular_1.14.0     memoise_2.0.1              dplyr_1.0.10
#> [166] ape_5.6-2               irlba_2.3.5.1
```