Multiome tutorial - MultiAssayExperiment

Xiaosai Yao

2 August 2023

Package

epiregulon 1.0.28

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)
#> Warning: replacing previous import 'GenomicRanges::union' by 'igraph::union'
#> when loading 'epiregulon'
```

Alternatively, you could install from github

```
devtools::install_github(repo ='xiaosaiyao/epiregulon')

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

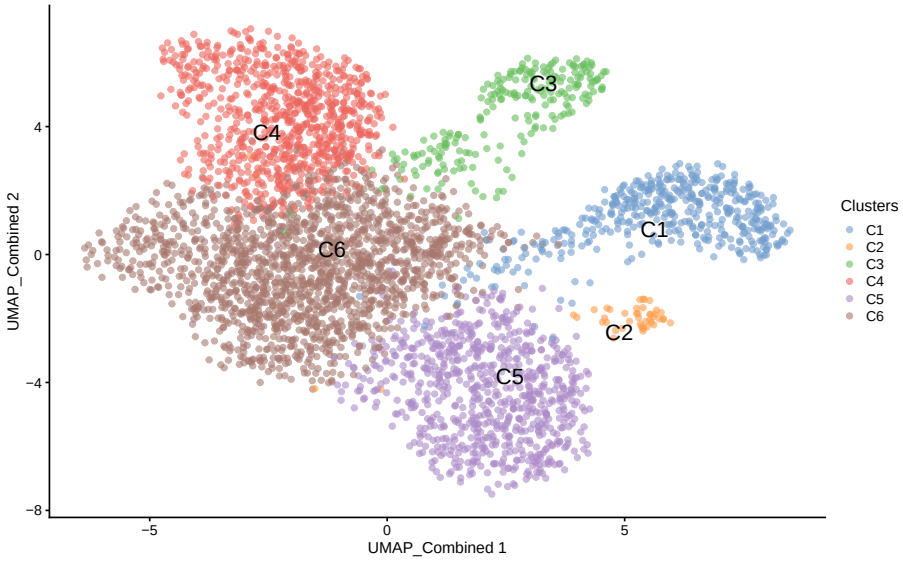
Multiome data can now be conveniently processed by `initiate.archr` and then `gp.sa.archr` to obtain peak matrices. Finally, the archR project can be uploaded into DatasetDB as a MultiAssayExperiment object using `maw.archr::importArchr` or `maw.archr::create.mae.with.multiple.sces.from.archr`

```
# load the MAE object
library(scMultiome)
#> Loading required package: AnnotationHub
#> Loading required package: BiocFileCache
#> Loading required package: dbplyr
#>
#> Attaching package: 'AnnotationHub'
#> The following object is masked from 'package:Biobase':
#>
#>     cache
#> Loading required package: ExperimentHub
#> Loading required package: MultiAssayExperiment

mae <- scMultiome::reprogramSeq()
#> snapshotDate(): 2023-08-02
```

```
#> see ?scMultiome and browseVignettes('scMultiome') for documentation
#> loading from cache

# peak matrix
PeakMatrix <- mae[["PeakMatrix"]]

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

assay(GeneExpressionMatrix,"normalizedCounts") <- assay(GeneExpressionMatrix,2)
assay(GeneExpressionMatrix,"logcounts") <- log2(assay(GeneExpressionMatrix,"normalizedCounts")+1)


# 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")
#> snapshotDate(): 2023-08-02
#> see ?scMultiome and browseVignettes('scMultiome') for documentation
#> loading from cache
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.

```
set.seed(1010)
p2g <- calculateP2G(peakMatrix = PeakMatrix,
                    expMatrix = GeneExpressionMatrix,
                    reducedDim = reducedDimMatrix,
                    exp_assay = "normalizedCounts",
                    peak_assay = "counts")
#> Using epiregulon to compute peak to gene links...
#> performing k means clustering to form metacells
#> Computing correlation
```

```
p2g
#> DataFrame with 25693 rows and 8 columns
#>         idxATAC         chr     start       end    idxRNA     target
#>       <integer> <character> <integer> <integer> <integer>    <array>
#> 1             1        chr1    817121    817621        19     FAM41C
#> 2             6        chr1    869650    870150        14 AL669831.2
#> 3            10        chr1    920987    921487        19     FAM41C
#> 4            22        chr1    960317    960817        19     FAM41C
#> 5            22        chr1    960317    960817        28      PERM1
#> ...         ...         ...       ...       ...       ...        ...
#> 25689    126586        chrX 155071227 155071727     36422      MTCP1
#> 25690    126590        chrX 155228844 155229344     36426      CLIC2
#> 25691    126592        chrX 155334445 155334945     36426      CLIC2
#> 25692    126596        chrX 155820104 155820604     36436      VAMP7
#> 25693    126599        chrX 155897986 155898486     36436      VAMP7
#>       Correlation  distance
#>          <matrix> <integer>
#> 1        0.505457     50578
#> 2        0.614009    108540
#> 3        0.662441     50587
#> 4        0.635411     89917
#> 5        0.614227     18210
#> ...           ...       ...
#> 25689    0.504839         0
#> 25690    0.830640    103268
#> 25691    0.529991         0
#> 25692    0.629908     58739
#> 25693    0.505036     16442
```

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
#> 1       1     2     5-mC
#> 2       1    22 AML1-ETO
#> 3       1    25       AR
#> 4       1    49     ATF1
#> 5       1    50     ATF2
#> 6       1    51     ATF3
```

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)
regulon
#> DataFrame with 3187407 rows and 11 columns
#>           idxATAC         chr     start       end    idxRNA      target
#>         <integer> <character> <integer> <integer> <integer> <character>
#> 1               1        chr1    817121    817621        19      FAM41C
#> 2               1        chr1    817121    817621        19      FAM41C
#> 3               1        chr1    817121    817621        19      FAM41C
#> 4               1        chr1    817121    817621        19      FAM41C
#> 5               1        chr1    817121    817621        19      FAM41C
#> ...           ...         ...       ...       ...       ...         ...
#> 3187403    126599        chrX 155897986 155898486     36436       VAMP7
#> 3187404    126599        chrX 155897986 155898486     36436       VAMP7
#> 3187405    126599        chrX 155897986 155898486     36436       VAMP7
#> 3187406    126599        chrX 155897986 155898486     36436       VAMP7
#> 3187407    126599        chrX 155897986 155898486     36436       VAMP7
#>                all  distance     idxTF          tf     corr
#>         <numeric> <integer> <integer> <character> <matrix>
#> 1        0.505457     50578         2        5-mC
#> 2        0.505457     50578        22    AML1-ETO
#> 3        0.505457     50578        25          AR
#> 4        0.505457     50578        49        ATF1
#> 5        0.505457     50578        50        ATF2
#> ...           ...       ...       ...         ...      ...
#> 3187403  0.505036     16442       436      HOXB13
#> 3187404  0.505036     16442       669       NANOG
#> 3187405  0.505036     16442       762     ONECUT2
#> 3187406  0.505036     16442      1044       SUMO2
#> 3187407  0.505036     16442      1116        TLE3
```

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 = "normalizedCounts",
                               peakMatrix = PeakMatrix,
                               peak_assay = "counts",
                               test = "chi.sq",
                               regulon,
                               clusters = GeneExpressionMatrix$Clusters,
                               prune_value = "pval",
                               regulon_cutoff = 0.05
                               )

pruned.regulon
```

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  = "normalizedCounts",
                        peakMatrix = PeakMatrix,
                        peak_assay = "counts",
                        clusters = GeneExpressionMatrix$Clusters,
                        block_factor = NULL,
                        tf_re.merge = TRUE,
                        method = "corr")

regulon.w
```

4.7 (Optional) Annotate with TF motifs

So far the gene regulatory network was constructed from TF ChIP-seq exclusively. Some users would prefer to further annotate the regulatory elements with the presence of motifs. We provide an option to annotate peaks with motifs from the Cisbp database. If no motifs are present for this particular factor (as in the case of co-factors or chromatin modifiers), we return NA. If motifs are available for a factor and the RE contains a motif, we return 1. If motifs are available and the RE does not contain a motif, we return 0.

If the user has already performed motif annotation with ArchR, we could also retrieve the results directly. See `?addMotifScore`

```
regulon.w.motif <- addMotifScore(regulon = regulon.w,
                                 peaks = rowRanges(PeakMatrix),
                                 species = "human",
                                 genome = "hg38")
#> annotating peaks with motifs
#>
#>
#> Attaching package: 'Biostrings'
#> The following object is masked from 'package:base':
#>
#>     strsplit
#>
#> Attaching package: 'rtracklayer'
#> The following object is masked from 'package:AnnotationHub':
#>
#>     hubUrl

# if desired, set weight to 0 if no motif is found
regulon.w.motif$weight[regulon.w.motif$motif == 0] <- 0
```

4.8 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 $\{1,2,\ldots,n\}$ $weights_i$ is the weight of TF and target i

```
score.combine <- calculateActivity(expMatrix = GeneExpressionMatrix,
                                   regulon = regulon.w,
                                   mode = "weight",
                                   method = "weightedMean",
                                   exp_assay = "normalizedCounts",
                                   normalize = FALSE)
```

```
#> calculating TF activity from regulon using weightedmean
#> aggregating regulons...
#> creating weight matrix...
#> calculating activity scores...
#> normalize by the number of targets...
```

4.9 Perform differential activity

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

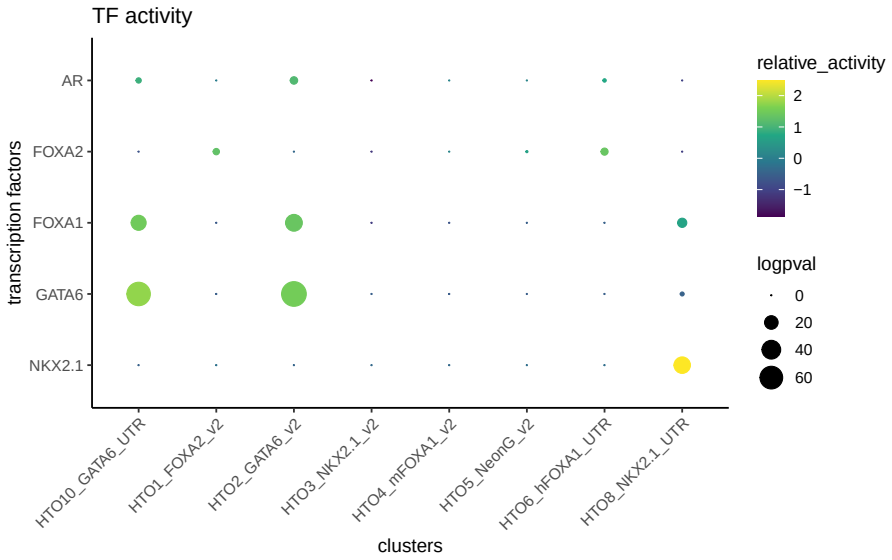
Take the top TFs

```
markers.sig <- getSigGenes(markers, topgenes = 5 )
#> Using a logFC cutoff of 0.2 for class HTO10_GATA6_UTR
#> Using a logFC cutoff of 0 for class HTO1_FOXA2_v2
#> Using a logFC cutoff of 0.2 for class HTO2_GATA6_v2
#> Using a logFC cutoff of 0 for class HTO3_NKX2.1_v2
#> Using a logFC cutoff of 0 for class HTO4_mFOXA1_v2
#> Using a logFC cutoff of 0 for class HTO5_NeonG_v2
#> Using a logFC cutoff of 0 for class HTO6_hFOXA1_UTR
#> Using a logFC cutoff of 0 for class HTO8_NKX2.1_UTR
```

4.10 Visualize the results

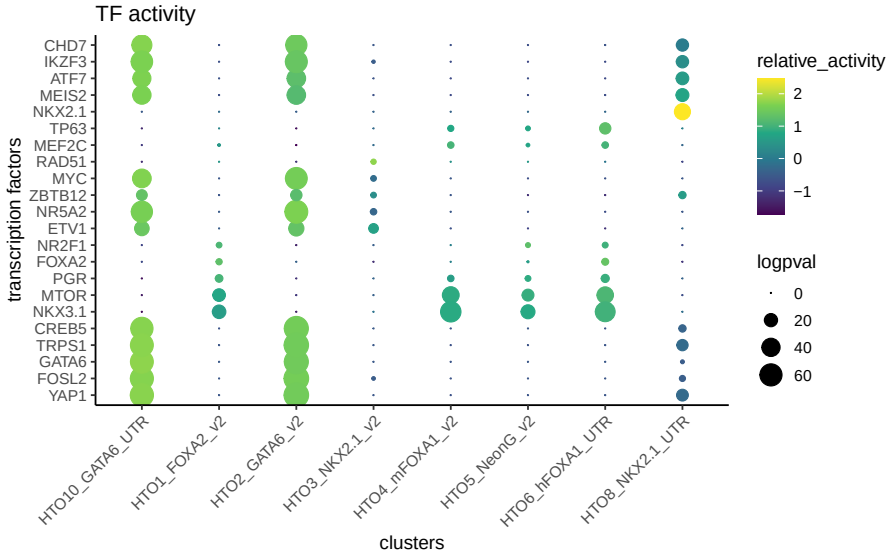
First visualize the known differential TFs by bubble plot

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



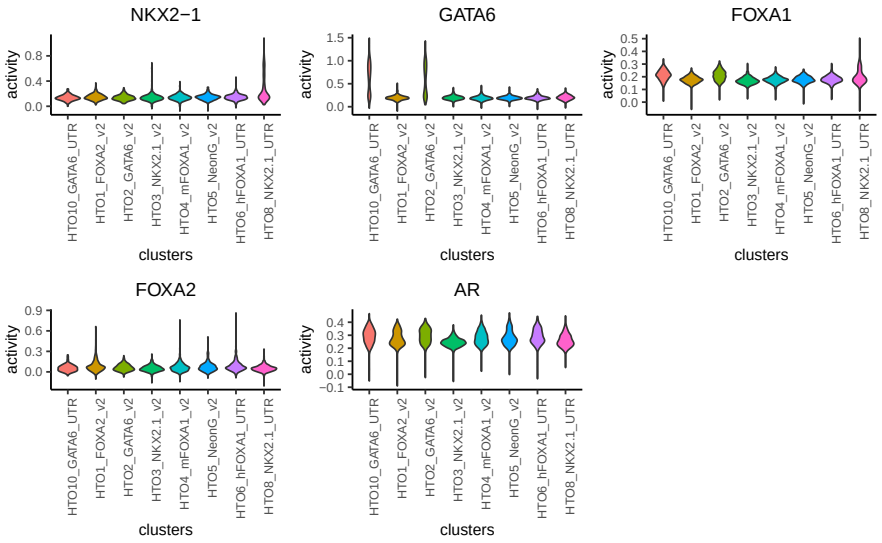
Then visualize the most differential TFs by clusters

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



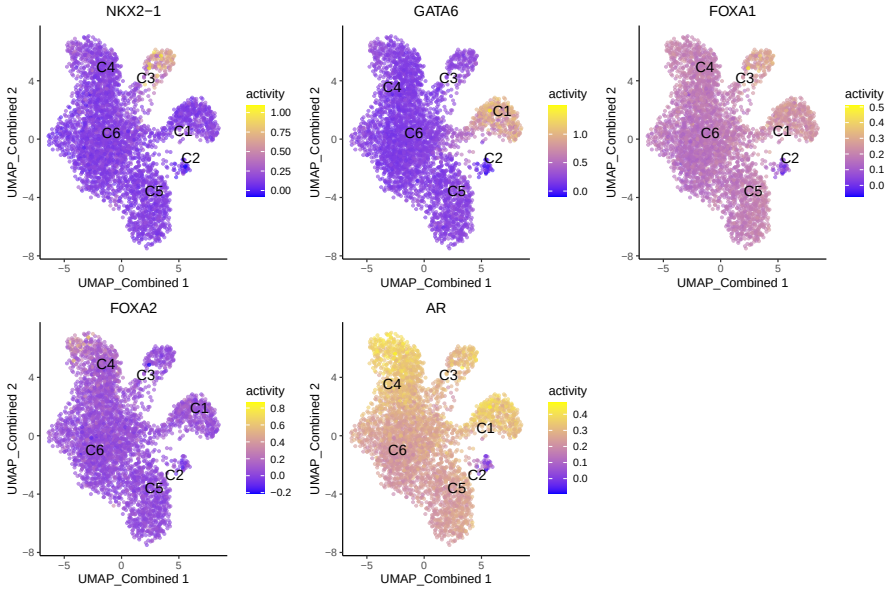
Visualize the known differential TFs by violin plot. Note there is no activity calculated for SOX2 because the expression of SOX2 is 0 in all cells.

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



Visualize the known differential TFs by UMAP

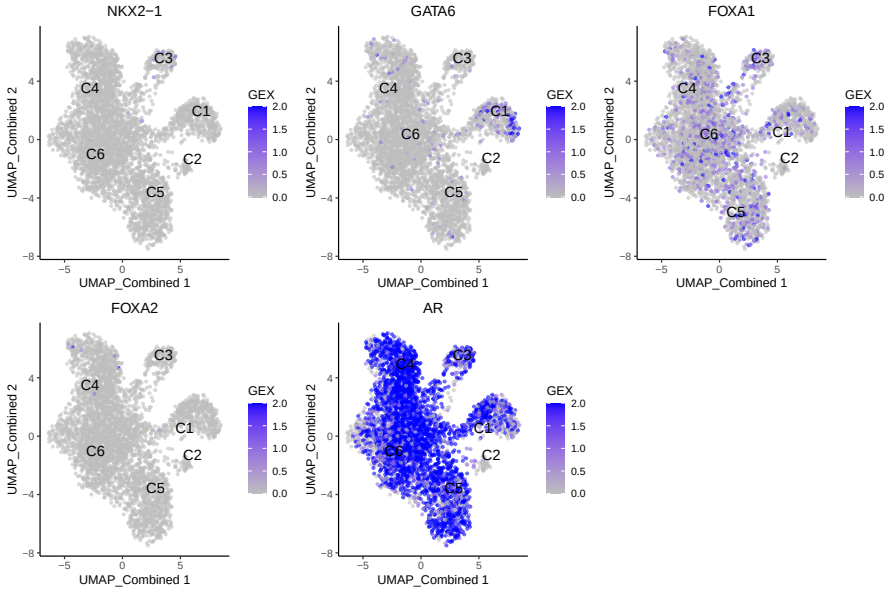
```
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = score.combine,
                tf = c("NKX2-1","GATA6","FOXA1","FOXA2", "AR"),
                dimtype = "UMAP_Combined",
                label = "Clusters",
                point_size = 1,
                ncol = 3)
```



In contrast, the gene expression of the TFs is very sparse

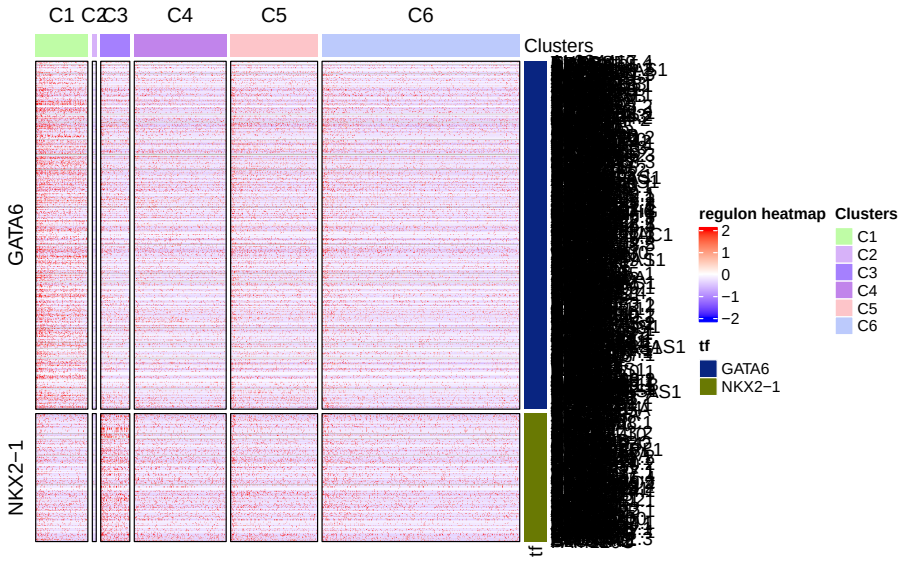
```
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = counts(GeneExpressionMatrix),
```

```
                tf = c("NKX2-1","GATA6","FOXA1","FOXA2", "AR"),
                dimtype = "UMAP_Combined",
                label = "Clusters",
                point_size = 1,
                ncol = 3,
                limit = c(0,2),
                colors = c("grey","blue"),
                legend.label = "GEX")
```

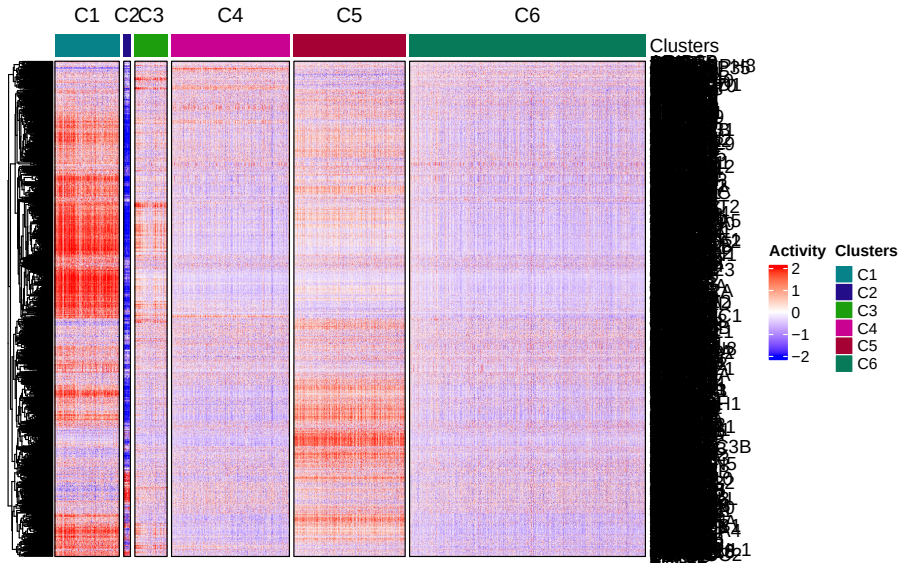


Visualize the gene expression of the regulons by heatmap

```
plotHeatmapRegulon(sce=GeneExpressionMatrix,
                   tfs=c("GATA6","NKX2-1"),
                   regulon=regulon.w,
                   regulon_cutoff=0.1,
                   downsample=1000,
                   cell_attributes="Clusters",
                   col_gap="Clusters",
                   exprs_values="counts",
                   name="regulon heatmap")
```



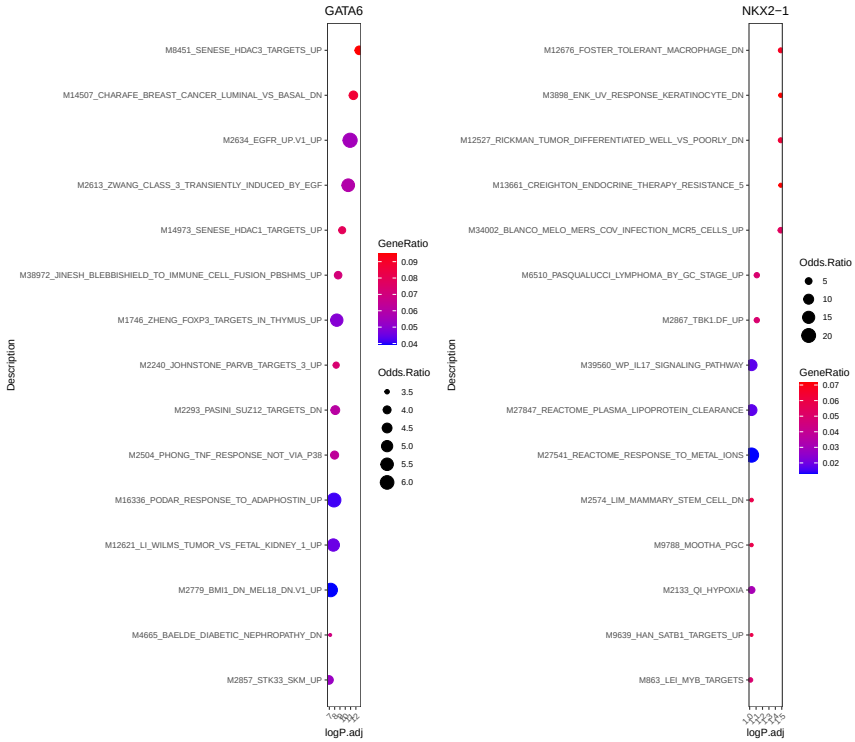

```
plotHeatmapActivity(activity=score.combine,
                    sce=GeneExpressionMatrix,
                    tfs=rownames(score.combine),
                    downsample=5000,
                    cell_attributes="Clusters",
                    col_gap="Clusters",
                    name = "Activity")
```



4.11 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.

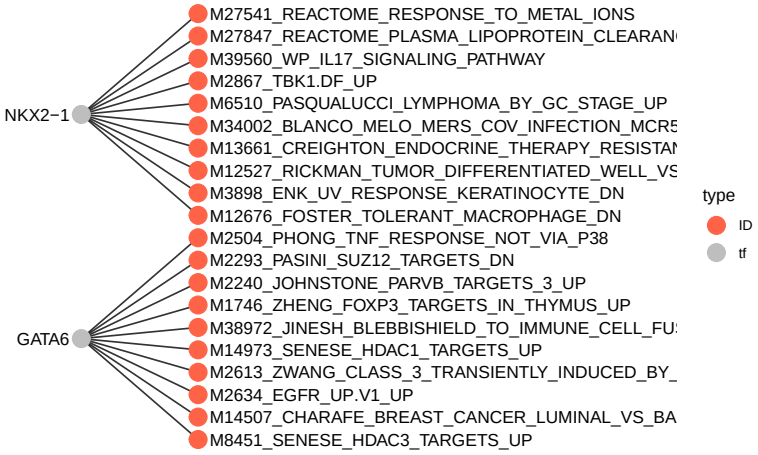
```
        #retrieve genesets
        H <- EnrichmentBrowser::getGenesets(org = "hsa",
                                            db = "msigdb",
                                            cat = "H",
                                            gene.id.type = "SYMBOL" )
        C2 <- EnrichmentBrowser::getGenesets(org = "hsa",
                                             db = "msigdb",
                                             cat = "C2",
                                             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, C2, C6)
        gs.list <- do.call(rbind,lapply(names(gs), function(x)
          {data.frame(gs=x, genes=gs[[x]])}))

        enrichresults <- regulonEnrich(TF = c("GATA6","NKX2-1"),
                                       regulon = regulon.w,
                                       weight = "weight",
                                       weight_cutoff = 0,
                                       genesets = gs.list)
        #> GATA6
        #> NKX2-1

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



4.12 Network analysis

We can visualize the genesets as a network

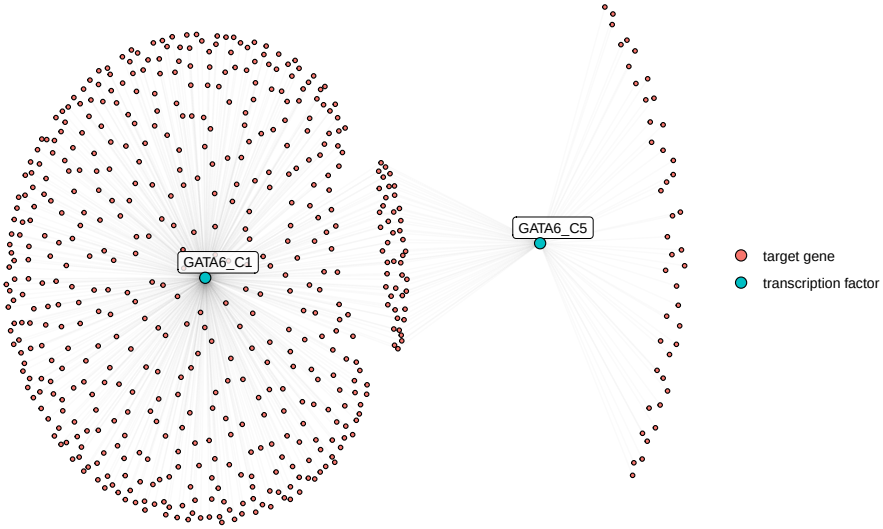
```
plotGseaNetwork(tf = names(enrichresults),
                enrichresults = enrichresults,
                p.adj_cutoff = 0.1,
                ntop_pathways = 10)
```



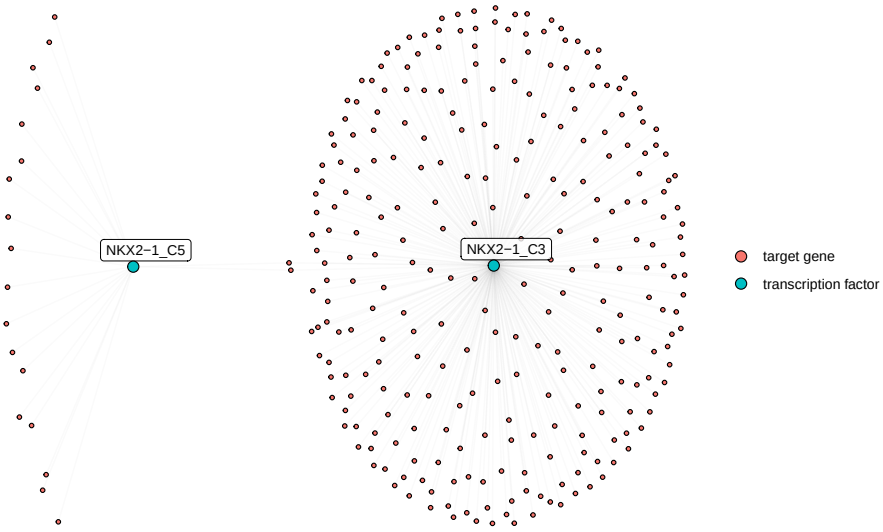
4.13 Differential networks

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"),
                weight = "stats",
                clusters = c("C1","C5"),
                layout = "stress")
#> Building graph using weight as edge weights
```



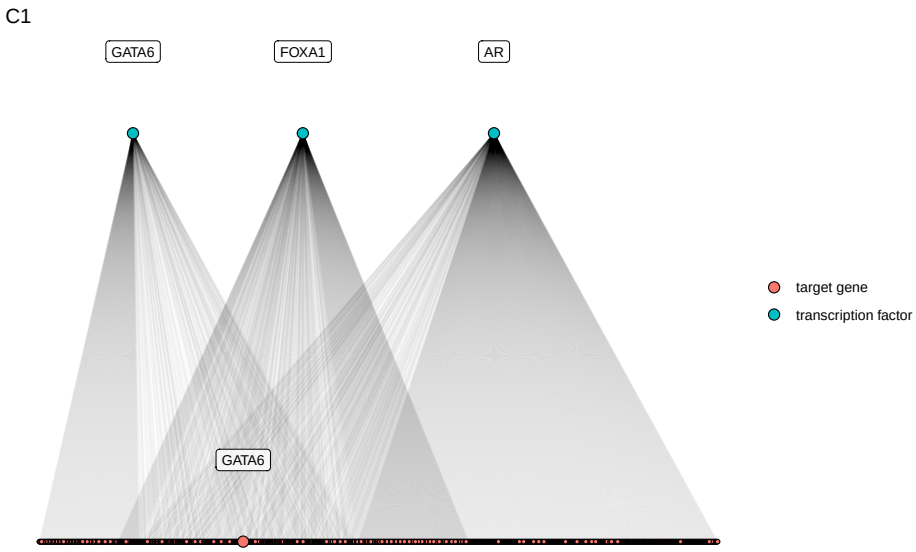

```
plotDiffNetwork(pruned.regulon,
                cutoff = 1,
                tf = c("NKX2-1"),
                weight = "stats",
                clusters = c("C3","C5"),
                layout = "stress")
#> Building graph using weight as edge weights
```



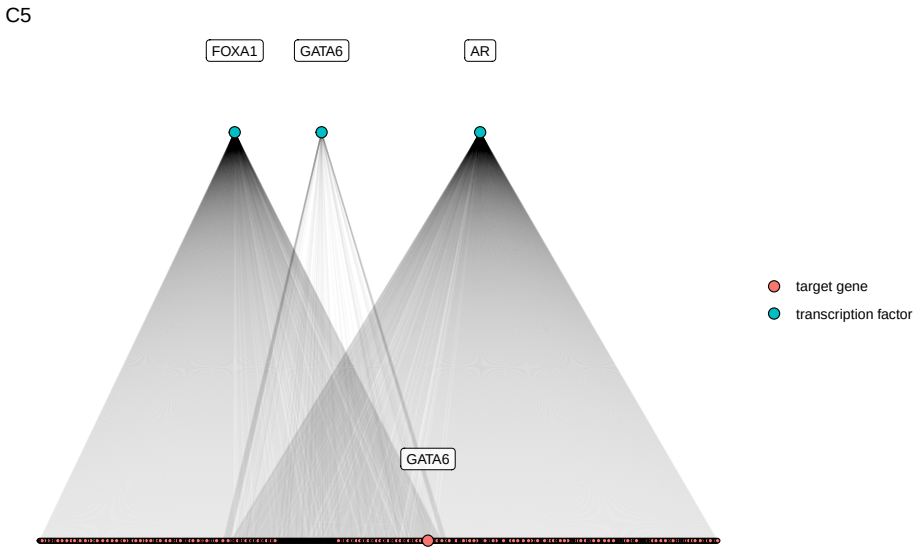
We can also visualize how transcription factors relate to other transcription factors in each cluster.

```
selected <- which(pruned.regulon$stats[,"C1"]> 1 &
                  pruned.regulon$tf %in% c("GATA6","FOXA1","AR"))
C1_network <- buildGraph(pruned.regulon[selected,],
```

```
                          weights = "stats",
                          cluster = "C1")
#> Building graph using stats as edge weights
selected <- which(pruned.regulon$stats[,"C5"]> 1 &
                  pruned.regulon$tf %in% c("GATA6","FOXA1","AR"))
C5_network <- buildGraph(pruned.regulon[selected,],
                          weights = "stats",
                          cluster = "C5")
#> Building graph using stats as edge weights
plotEpiregulonNetwork(C1_network,
                      layout = "sugiyama",
                      tfs_to_highlight = c("GATA6","FOXA1","AR")) +
  ggplot2::ggtitle ("C1")
```

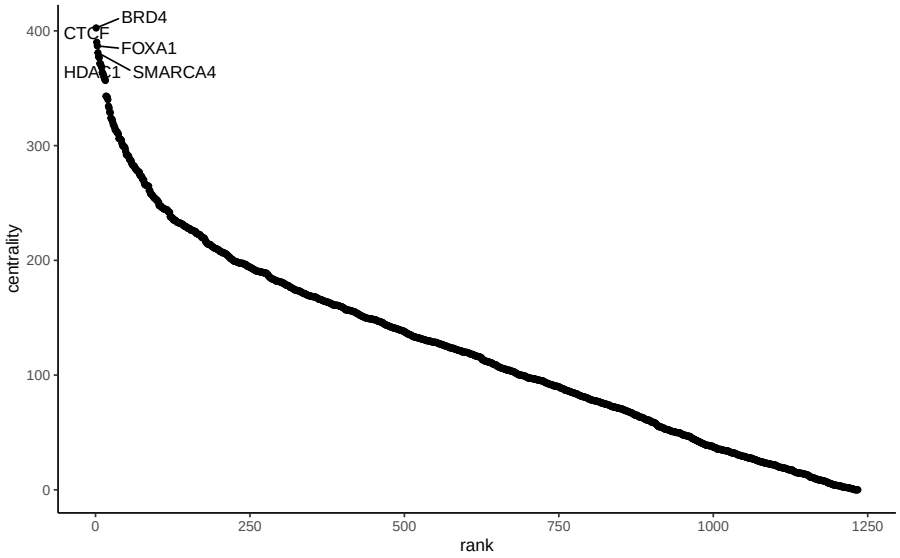



```
plotEpiregulonNetwork(C5_network,
                      layout = "sugiyama",
                      tfs_to_highlight = c("GATA6","FOXA1","AR")) +
  ggplot2::ggtitle ("C5")
```



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", cluster="C1")
#> Replacement of na values for weights with 0
#> Building graph using stats as edge weights
C5_network <- buildGraph(pruned.regulon, weights = "stats", cluster="C5")
#> Replacement of na values for weights with 0
#> Building graph using stats as edge weights

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, 5), aes(label = tf)) +
    theme_classic()
```



5 Session Info

```
sessionInfo()
#> R version 4.3.0 (2023-04-21)
#> 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;  LAPACK version 3.11.0
#>
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C         LC_TIME=C
#>  [4] LC_COLLATE=C         LC_MONETARY=C        LC_MESSAGES=C
#>  [7] LC_PAPER=C           LC_NAME=C            LC_ADDRESS=C
#> [10] LC_TELEPHONE=C       LC_MEASUREMENT=C     LC_IDENTIFICATION=C
#>
#> time zone: Etc/UTC
#> tzcode source: system (glibc)
#>
#> attached base packages:
#> [1] stats4    stats     graphics  grDevices utils     datasets  methods
#> [8] base
#>
#> other attached packages:
#>  [1] ggplot2_3.4.2                    org.Hs.eg.db_3.17.0
#>  [3] AnnotationDbi_1.63.2             msigdbr_7.5.1
#>  [5] BSgenome.Hsapiens.UCSC.hg38_1.4.5 BSgenome_1.69.0
#>  [7] rtracklayer_1.61.0               Biostrings_2.69.2
#>  [9] XVector_0.41.1                   scMultiome_1.1.0
```

```
#> [11] MultiAssayExperiment_1.27.0       ExperimentHub_2.9.1
#> [13] AnnotationHub_3.9.1               BiocFileCache_2.9.1
#> [15] dbplyr_2.3.3                      epiregulon_1.0.28
#> [17] SingleCellExperiment_1.23.0       SummarizedExperiment_1.31.1
#> [19] Biobase_2.61.0                    GenomicRanges_1.53.1
#> [21] GenomeInfoDb_1.37.2               IRanges_2.35.2
#> [23] S4Vectors_0.39.1                  BiocGenerics_0.47.0
#> [25] MatrixGenerics_1.13.1             matrixStats_1.0.0
#> [27] BiocStyle_2.29.1
#>
#> loaded via a namespace (and not attached):
#>   [1] fs_1.6.2                          GSVA_1.49.4
#>   [3] bitops_1.0-7                      enrichplot_1.21.1
#>   [5] DirichletMultinomial_1.43.0       TFBSTools_1.39.0
#>   [7] HDO.db_0.99.1                     httr_1.4.6
#>   [9] RColorBrewer_1.1-3                doParallel_1.0.17
#>  [11] Rgraphviz_2.45.0                  tools_4.3.0
#>  [13] backports_1.4.1                   utf8_1.2.3
#>  [15] R6_2.5.1                          HDF5Array_1.29.3
#>  [17] lazyeval_0.2.2                    rhdf5filters_1.13.5
#>  [19] GetoptLong_1.0.5                  withr_2.5.0
#>  [21] gridExtra_2.3                     cli_3.6.1
#>  [23] Cairo_1.6-0                       scatterpie_0.2.1
#>  [25] labeling_0.4.2                    KEGGgraph_1.61.0
#>  [27] readr_2.1.4                       yulab.utils_0.0.6
#>  [29] Rsamtools_2.17.0                  gson_0.1.0
#>  [31] DOSE_3.27.2                       R.utils_2.12.2
#>  [33] scater_1.29.0                     limma_3.57.6
#>  [35] rstudioapi_0.14                   RSQLite_2.3.1
#>  [37] gridGraphics_0.5-1                generics_0.1.3
#>  [39] shape_1.4.6                       BiocIO_1.11.0
#>  [41] gtools_3.9.4                      dplyr_1.1.2
#>  [43] GO.db_3.17.0                      Matrix_1.6-0
#>  [45] ggbeeswarm_0.7.2                  fansi_1.0.4
#>  [47] abind_1.4-5                       R.methodsS3_1.8.2
#>  [49] lifecycle_1.0.3                   yaml_2.3.7
#>  [51] edgeR_3.43.7                      qvalue_2.33.0
#>  [53] rhdf5_2.45.1                      SparseArray_1.1.11
#>  [55] grid_4.3.0                        blob_1.2.4
#>  [57] promises_1.2.0.1                  dqrng_0.3.0
#>  [59] crayon_1.5.2                      lattice_0.21-8
#>  [61] beachmat_2.17.14                  cowplot_1.1.1
#>  [63] annotate_1.79.0                   KEGGREST_1.41.0
#>  [65] magick_2.7.4                      pillar_1.9.0
#>  [67] knitr_1.43                        ComplexHeatmap_2.17.0
#>  [69] metapod_1.9.0                     fgsea_1.27.0
#>  [71] rjson_0.2.21                      codetools_0.2-19
#>  [73] fastmatch_1.1-3                   glue_1.6.2
#>  [75] ggfun_0.1.1                       downloader_0.4
#>  [77] data.table_1.14.8                 treeio_1.25.2
#>  [79] vctrs_0.6.2                       png_0.1-8
```

```
#>  [81] gtable_0.3.3                       poweRlaw_0.70.6
#>  [83] cachem_1.0.8                       xfun_0.39
#>  [85] S4Arrays_1.1.5                     mime_0.12
#>  [87] tidygraph_1.2.3                    pracma_2.4.2
#>  [89] iterators_1.0.14                   statmod_1.5.0
#>  [91] bluster_1.11.3                     interactiveDisplayBase_1.39.0
#>  [93] ellipsis_0.3.2                     nlme_3.1-162
#>  [95] ArchR_1.0.3                        ggtree_3.9.0
#>  [97] bit64_4.0.5                        filelock_1.0.2
#>  [99] irlba_2.3.5.1                      vipor_0.4.5
#> [101] colorspace_2.1-0                   seqLogo_1.67.0
#> [103] DBI_1.1.3                          tidyselect_1.2.0
#> [105] bit_4.0.5                          compiler_4.3.0
#> [107] curl_5.0.0                         graph_1.79.0
#> [109] BiocNeighbors_1.19.0               BSgenome.Mmusculus.UCSC.mm10_1.4.3
#> [111] DelayedArray_0.27.10               shadowtext_0.1.2
#> [113] bookdown_0.34                      checkmate_2.2.0
#> [115] scales_1.2.1                       caTools_1.18.2
#> [117] rappdirs_0.3.3                     stringr_1.5.0
#> [119] digest_0.6.31                      motifmatchr_1.23.0
#> [121] rmarkdown_2.23                     BSgenome.Hsapiens.UCSC.hg19_1.4.3
#> [123] htmltools_0.5.5                    pkgconfig_2.0.3
#> [125] sparseMatrixStats_1.13.0           fastmap_1.1.1
#> [127] rlang_1.1.1                        GlobalOptions_0.1.2
#> [129] shiny_1.7.4.1                      DelayedMatrixStats_1.23.0
#> [131] farver_2.1.1                       jsonlite_1.8.7
#> [133] BiocParallel_1.35.3                GOSemSim_2.27.2
#> [135] R.oo_1.25.0                        BiocSingular_1.17.1
#> [137] RCurl_1.98-1.12                    magrittr_2.0.3
#> [139] ggplotify_0.1.1                    scuttle_1.11.0
#> [141] GenomeInfoDbData_1.2.10            patchwork_1.1.2
#> [143] Rhdf5lib_1.23.0                    munsell_0.5.0
#> [145] Rcpp_1.0.11                        ape_5.7-1
#> [147] babelgene_22.9                     viridis_0.6.4
#> [149] EnrichmentBrowser_2.31.5           stringi_1.7.12
#> [151] ggraph_2.1.0                       MASS_7.3-60
#> [153] zlibbioc_1.47.0                    plyr_1.8.8
#> [155] HPO.db_0.99.2                      parallel_4.3.0
#> [157] ggrepel_0.9.3                      CNEr_1.37.0
#> [159] graphlayouts_1.0.0                 splines_4.3.0
#> [161] hms_1.1.3                          circlize_0.4.15
#> [163] locfit_1.5-9.8                     igraph_1.5.0.1
#> [165] reshape2_1.4.4                     ScaledMatrix_1.9.1
#> [167] TFMPvalue_0.0.9                    BiocVersion_3.18.0
#> [169] XML_3.99-0.14                      evaluate_0.21
#> [171] scran_1.29.0                       BiocManager_1.30.21.1
#> [173] tweenr_2.0.2                       tzdb_0.4.0
#> [175] foreach_1.5.2                      httpuv_1.6.11
#> [177] polyclip_1.10-4                    tidyr_1.3.0
#> [179] purrr_1.0.1                        clue_0.3-64
#> [181] ggforce_0.4.1                      rsvd_1.0.5
```

```
#> [183] xtable_1.8-4                    restfulr_0.0.15
#> [185] tidytree_0.4.4                  MPO.db_0.99.7
#> [187] later_1.3.1                     viridisLite_0.4.2
#> [189] tibble_3.2.1                    aplot_0.1.10
#> [191] clusterProfiler_4.9.2           memoise_2.0.1
#> [193] beeswarm_0.4.0                  GenomicAlignments_1.37.0
#> [195] cluster_2.1.4                   GSEABase_1.63.0
```