prostate cancer archr tutorial

Xiaosai Yao

3 June 2023

Package

epiregulon 1.0.25

Contents

1 Introduction

This tutorial walks through an example of TF activity inference in unpaired scATAC-seq/scRNAseq of parental LNCaP cells treated with DMSO, Enzalutamide and Enza resistant cells. The dataset was taken from Taavitsainen et al GSE168667 and GSE168668.

2 Installation

Epiregulon is currently available on R/dev

Alternatively, you could install from gitlab

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

library(epiregulon)
```

3 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. RNA-seq integration. a. For unpaired scATAC-seq, use addGeneIntegrationMatrix. See chapter 8 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

3.1 Load ArchR project

Copy this ArchR project into your own directory

```
archR_project_path <- "/gstore/project/lineage/prostate/GSE168667/OUTPUT/multiome/"
proj <- loadArchRProject(path = archR_project_path, showLogo = FALSE)
```

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

```
getAvailableMatrices(proj)
#> [1] "GeneIntegrationMatrix" "GeneScoreMatrix"       "MotifMatrix"
#> [4] "PeakMatrix"            "TileMatrix"
```

We will use the joint reducedDims - "LSI_Combined" and joint embeddings - "UMAP_Combined"

```
head(getReducedDims(proj, reducedDims = "iLSI_Combined")[,1:5])
#>                                  LSI1       LSI2       LSI3        LSI4
#> SRR13927735#TTATGTCTCCAGGTAT-1 -2.713935 -0.3677949 -0.4484238 -0.30645138
```

```
#> SRR13927735#TATTGCTCATCAGAAA-1 -2.642781 -0.2767556 -0.9142714 -0.19675812
#> SRR13927735#TTCGATTGTAGGGTTG-1 -2.322865 -0.1543080 -1.4106049 -0.08891276
#> SRR13927735#CATTCATTCGGATGTT-1 -2.572976 -0.1917188 -1.0464294 -0.12660121
#> SRR13927735#ACGTTAGGTCAACTGT-1 -2.478552 -0.1776639 -1.1037295 -0.22976613
#> SRR13927735#AAATGCCCAGCAATGG-1 -2.595352 -0.3803464 -0.7770309 -0.52431765
#>                                        LSI5
#> SRR13927735#TTATGTCTCCAGGTAT-1 -0.046845365
#> SRR13927735#TATTGCTCATCAGAAA-1  0.075746940
#> SRR13927735#TTCGATTGTAGGGTTG-1  0.019873276
#> SRR13927735#CATTCATTCGGATGTT-1  0.009947438
#> SRR13927735#ACGTTAGGTCAACTGT-1 -0.150097539
#> SRR13927735#AAATGCCCAGCAATGG-1 -0.243074591
head(getEmbedding(proj, embedding = "UMAP_Combined"))
#>                                iLSI_Combined#UMAP_Dimension_1
#> SRR13927735#TTATGTCTCCAGGTAT-1                      -9.622903
#> SRR13927735#TATTGCTCATCAGAAA-1                      -9.360211
#> SRR13927735#TTCGATTGTAGGGTTG-1                      -8.617347
#> SRR13927735#CATTCATTCGGATGTT-1                      -9.285448
#> SRR13927735#ACGTTAGGTCAACTGT-1                      -8.809260
#> SRR13927735#AAATGCCCAGCAATGG-1                      -9.261216
#>                                iLSI_Combined#UMAP_Dimension_2
#> SRR13927735#TTATGTCTCCAGGTAT-1                     -0.2908237
#> SRR13927735#TATTGCTCATCAGAAA-1                     -0.2892935
#> SRR13927735#TTCGATTGTAGGGTTG-1                     -0.2154103
#> SRR13927735#CATTCATTCGGATGTT-1                     -0.3267481
#> SRR13927735#ACGTTAGGTCAACTGT-1                     -0.2168703
#> SRR13927735#AAATGCCCAGCAATGG-1                      0.3200356
```

3.2 Retrieve matrices from ArchR project

Retrieve gene expression and peak matrix from the ArchR project

```
GeneExpressionMatrix <- getMatrixFromProject(
    ArchRProj = proj,
    useMatrix = "GeneIntegrationMatrix",
    useSeqnames = NULL,
    verbose = TRUE,
    binarize = FALSE,
    threads = 1,
    logFile = "x"
)
#> 2023-06-03 16:46:10.090058 : Organizing colData, 2.556 mins elapsed.
#> 2023-06-03 16:46:10.890214 : Organizing rowData, 2.569 mins elapsed.
#> 2023-06-03 16:46:10.898103 : Organizing rowRanges, 2.57 mins elapsed.
#> 2023-06-03 16:46:10.913467 : Organizing Assays (1 of 1), 2.57 mins elapsed.
#> 2023-06-03 16:46:23.427703 : Constructing SummarizedExperiment, 2.778 mins elapsed.
#> 2023-06-03 16:46:24.998601 : Finished Matrix Creation, 2.805 mins elapsed.

PeakMatrix <- getMatrixFromProject(
```

```
    ArchRProj = proj,
    useMatrix = "PeakMatrix",
    useSeqnames = NULL,
    verbose = TRUE,
    binarize = FALSE,
    threads = 1,
    logFile = "x"
)
#> 2023-06-03 16:47:41.916129 : Organizing colData, 1.278 mins elapsed.
#> 2023-06-03 16:47:42.89514 : Organizing rowData, 1.295 mins elapsed.
#> 2023-06-03 16:47:42.91496 : Organizing rowRanges, 1.295 mins elapsed.
#> 2023-06-03 16:47:42.941403 : Organizing Assays (1 of 1), 1.295 mins elapsed.
#> 2023-06-03 16:47:48.723018 : Constructing SummarizedExperiment, 1.392 mins elapsed.
#> 2023-06-03 16:48:10.944167 : Finished Matrix Creation, 1.762 mins elapsed.
```

If we extract the gene expression from matrix, it will be in the form of RangedSummarizedExperiment. We can make use of `ArchRMatrix2SCE` to convert gene expression matrix to SingleCellExperiment object. It's also important to note that gene expression from ArchR is library size normalized. We further transform it to logcounts by adding a pseudocount of 1 and taking log2.

```
GeneExpressionMatrix <- ArchRMatrix2SCE(GeneExpressionMatrix)
rownames(GeneExpressionMatrix) <- rowData(GeneExpressionMatrix)$name
assay(GeneExpressionMatrix) <- as(log2(assay(GeneExpressionMatrix)+1), "CsparseMatrix")
```

We rename the assay name of the PeakMatrix as counts

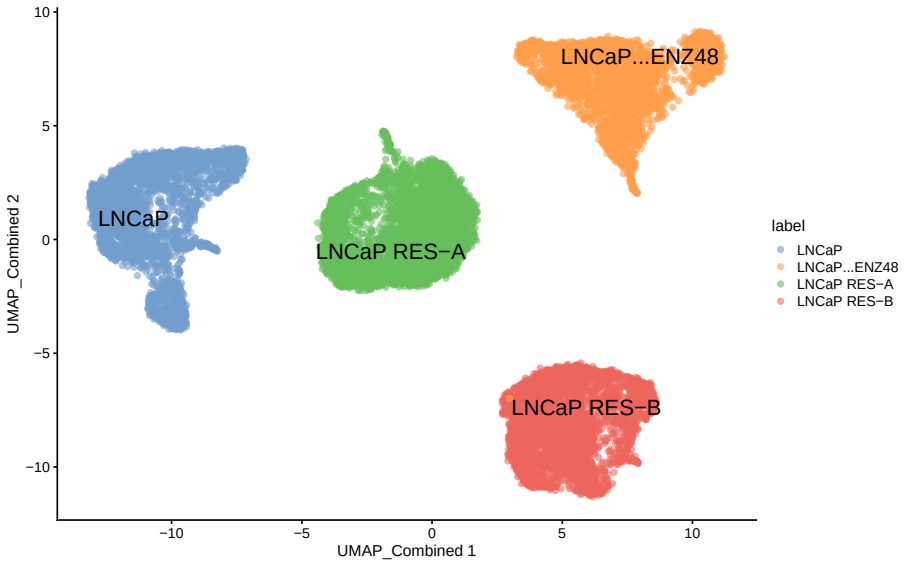
```
assayNames(PeakMatrix)  <-  "counts"
```

Transfer embeddings from ArchR project to singleCellExperiment

```
reducedDim(GeneExpressionMatrix, "UMAP_Combined") <- getEmbedding(ArchRProj = proj,
                                                                  embedding = "UMAP_Combined",
                                                                  returnDF = TRUE)[colnames(GeneExpressionMat



# add cell label
GeneExpressionMatrix$label <- GeneExpressionMatrix$Cells
GeneExpressionMatrix$label[GeneExpressionMatrix$Treatment == "enzalutamide 48h"] <- "LNCaP-ENZ48"
GeneExpressionMatrix$label <- factor(GeneExpressionMatrix$label,
                                     levels = c("LNCaP", "LNCaP-ENZ48", "LNCaP RES-A", "LNCaP RES-B"))
```

Visualize singleCellExperiment by UMAP

```
scater::plotReducedDim(GeneExpressionMatrix,
                       dimred = "UMAP_Combined",
                       text_by = "label",
                       colour_by = "label")
```



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(): 2023-04-24
#> 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 the `addPeak2GeneLinks` function from the ArchR package. The user would need to supply a path to an ArchR project already containing peak and gene matrices, as well as Latent semantic indexing (LSI) dimensionality reduction.

```
# path to ArchR project
p2g <- calculateP2G(ArchR_path = archR_project_path,
                    useDim = "iLSI_Combined",
                    useMatrix = "GeneIntegrationMatrix",
                    threads = 1)
#> Setting ArchRLogging = FALSE
#> Using ArchR to compute peak to gene links...
#> 2023-06-03 16:49:34.879248 : Getting Available Matrices, 0 mins elapsed.
#> 2023-06-03 16:49:37.125622 : Filtered Low Prediction Score Cells (0 of 15522, 0), 0.004 mins elapsed.
#> 2023-06-03 16:49:37.832338 : Computing KNN, 0.016 mins elapsed.
#> 2023-06-03 16:49:40.674425 : Identifying Non-Overlapping KNN pairs, 0.063 mins elapsed.
#> 2023-06-03 16:49:43.32916 : Identified 498 Groupings!, 0.107 mins elapsed.
#> 2023-06-03 16:49:43.389784 : Getting Group RNA Matrix, 0.108 mins elapsed.
#> 2023-06-03 16:53:20.361747 : Getting Group ATAC Matrix, 3.725 mins elapsed.
#> 2023-06-03 16:56:58.083631 : Normalizing Group Matrices, 7.353 mins elapsed.
#> 2023-06-03 16:57:06.65751 : Finding Peak Gene Pairings, 7.496 mins elapsed.
#> 2023-06-03 16:57:07.284271 : Computing Correlations, 7.507 mins elapsed.
#> 2023-06-03 16:57:16.743506 : Completed Peak2Gene Correlations!, 7.664 mins elapsed.
head(p2g)
#> DataFrame with 6 rows and 8 columns
#>     idxATAC      chr     start       end    idxRNA      target Correlation
#>   <integer> <factor> <integer> <integer> <integer> <character>   <numeric>
#> 1        15     chr1    912762    913262         7       NOC2L    0.546722
#> 2        15     chr1    912762    913262         8      KLHL17    0.516539
#> 3        25     chr1    920261    920761         7       NOC2L    0.649425
#> 4        25     chr1    920261    920761         8      KLHL17    0.637711
#> 5        32     chr1    927728    928228         7       NOC2L    0.610240
#> 6        32     chr1    927728    928228         8      KLHL17    0.550093
#>    distance
#>   <numeric>
#> 1     46297
#> 2     47575
#> 3     38798
#> 4     40076
#> 5     31331
#> 6     32609
```

4.3 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!
```

4.4 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)
#> DataFrame with 6 rows and 10 columns
#>     idxATAC      chr     start       end    idxRNA      target     corr
#>   <integer> <factor> <integer> <integer> <integer> <character> <matrix>
#> 1        15     chr1    912762    913262         8      KLHL17 0.516539
#> 2        15     chr1    912762    913262         8      KLHL17 0.516539
#> 3        15     chr1    912762    913262         8      KLHL17 0.516539
#> 4        15     chr1    912762    913262         8      KLHL17 0.516539
#> 5        15     chr1    912762    913262         8      KLHL17 0.516539
#> 6        15     chr1    912762    913262         8      KLHL17 0.516539
#>    distance     idxTF          tf
#>   <numeric> <integer> <character>
#> 1     47575        10        AGO1
#> 2     47575        22    AML1-ETO
#> 3     47575        32      ARID4A
#> 4     47575        33      ARID4B
#> 5     47575        34      ARID5B
#> 6     47575        80        BCOR
```

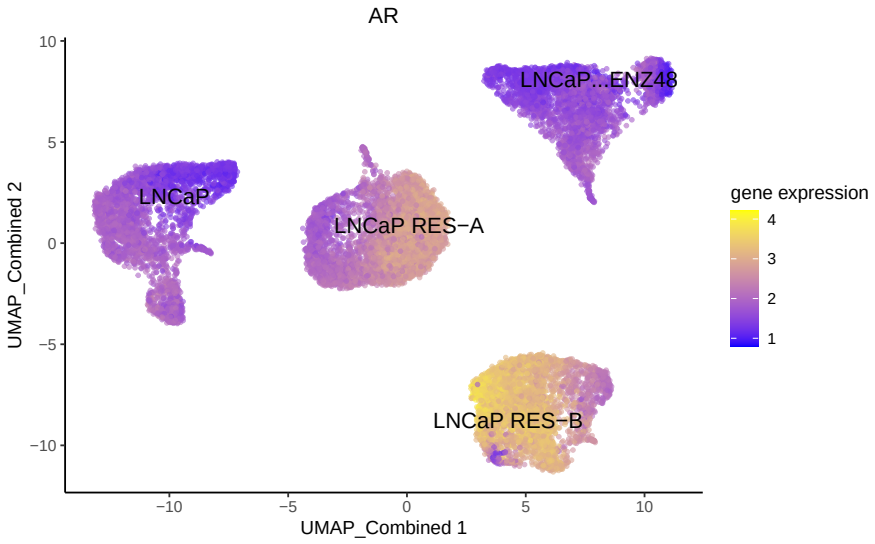
Epiregulon computes weights using either correlation, linear regression, mutual information, log fold change or wilcoxon rank sum test. The choice of methods depends on the datasets. Correlation works best when increased TF activity results from increased TF expression, such in the case of normal development. The user has a choice between computing the correlation of TF expression vs target gene expression by setting `method = "corr"`, or the product of TF expression and chromatin accessibility at TF-bound regulatory elements vs target gene expression by setting `method = "corr"` and `"tf_re.merge = TRUE`.

In the case of drug treatment, however, the activity of TF is suppressed often not by downregulation of the TF gene expression, but by direct interference of the TF protein function. In this dataset, the drug enzalutamide blocks the ligand binding domain of the

androgen receptor and prevents it from binding to the chromatin. As a result, while the AR gene expression stays the same, the chromatin accessibility of AR, as computed by chromVar in the ArchR package, is greatly reduced by 48 hour treatment of enzalutamide.

First, we visualize the AR expression and observed that enzalutamide did not decrease AR expression.

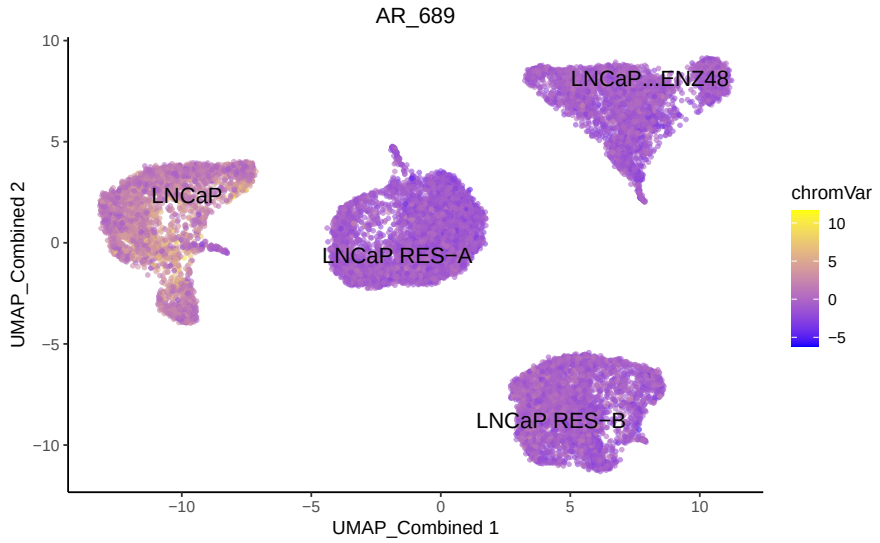
```
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = assay(GeneExpressionMatrix),
                tf = "AR",
                dimtype = "UMAP_Combined",
                label = "label",
                point_size = 1,
                legend.label = "gene expression")
```



Then we extract the chromVarMatrix from ArchR project and then visualize the chromatin accessibility at AR bound sites. We can see that 48 hour of enzalutamide treatment reduced chromatin accessibility at AR bound sites

```
chromVarMatrix <- getMatrixFromProject(
    ArchRProj = proj,
    useMatrix = "MotifMatrix",
    useSeqnames = NULL,
    verbose = TRUE,
    binarize = FALSE,
    threads = 1
)
#> 2023-06-03 17:00:11.91522 : Organizing colData, 0.299 mins elapsed.
#> 2023-06-03 17:00:12.073419 : Organizing rowData, 0.301 mins elapsed.
#> 2023-06-03 17:00:12.07607 : Organizing rowRanges, 0.301 mins elapsed.
#> 2023-06-03 17:00:12.081709 : Organizing Assays (1 of 2), 0.301 mins elapsed.
#> 2023-06-03 17:00:12.63229 : Organizing Assays (2 of 2), 0.31 mins elapsed.
#> 2023-06-03 17:00:13.202221 : Constructing SummarizedExperiment, 0.32 mins elapsed.
#> 2023-06-03 17:00:15.89814 : Finished Matrix Creation, 0.365 mins elapsed.
```

```
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = assay(chromVarMatrix, "z"),
                tf = "AR_689",
                dimtype = "UMAP_Combined",
                label = "label",
                point_size = 1,
                legend.label = "chromVar")
```



Therefore, we consider the choice of the `wilcoxon` test which compare target gene expression in cells meeting both the TF expression and accessibility cutoffs vs cells failing either the TF expression or/and accessibility cutoffs. We also compare the output of `wilcoxon` vs `corr`.

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

regulon.w.corr.re <- addWeights(regulon = pruned.regulon,
```

```
                            expMatrix = GeneExpressionMatrix,
                            exp_assay = "logcounts",
                            peakMatrix = PeakMatrix,
                            peak_assay = "counts",
                            clusters = GeneExpressionMatrix$Sample,
                            method = "corr",
                            tf_re.merge = TRUE)
#> adding weights using corr...
#> calculating average expression across clusters...
#> computing weights...
```

4.5 Calculate TF activity

Finally, the activities for a specific TF in each cell are computed by averaging the weighted expressions of target genes linked to the TF.

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

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

We calculate three different activities corresponding to the different weighted regulons

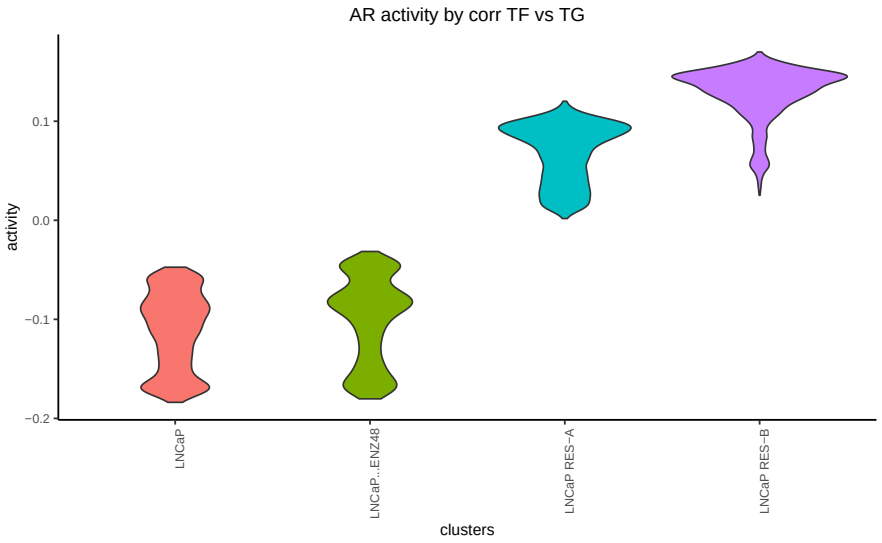
```
score.combine.wilcox <- calculateActivity(expMatrix = GeneExpressionMatrix,
                                    regulon = regulon.w.wilcox,
                                    normalize = TRUE,
                                    mode = "weight",
                                    method = "weightedMean")
#> calculating TF activity from regulon using weightedmean
#> aggregating regulons...
#> creating weight matrix...
#> calculating activity scores...
#> normalize by mean...
#> normalize by the number of targets...
score.combine.corr <- calculateActivity(expMatrix = GeneExpressionMatrix,
                                    regulon = regulon.w.corr,
                                    normalize = TRUE,
                                    mode = "weight",
                                    method = "weightedMean")
#> calculating TF activity from regulon using weightedmean
#> aggregating regulons...
#> creating weight matrix...
#> calculating activity scores...
#> normalize by mean...
#> normalize by the number of targets...
score.combine.corr.re <- calculateActivity(expMatrix = GeneExpressionMatrix,
                                    regulon = regulon.w.corr.re,
                                    normalize = TRUE,
                                    mode = "weight",
```

```
                                    method = "weightedMean")
#> calculating TF activity from regulon using weightedmean
#> aggregating regulons...
#> creating weight matrix...
#> calculating activity scores...
#> normalize by mean...
#> normalize by the number of targets...
```

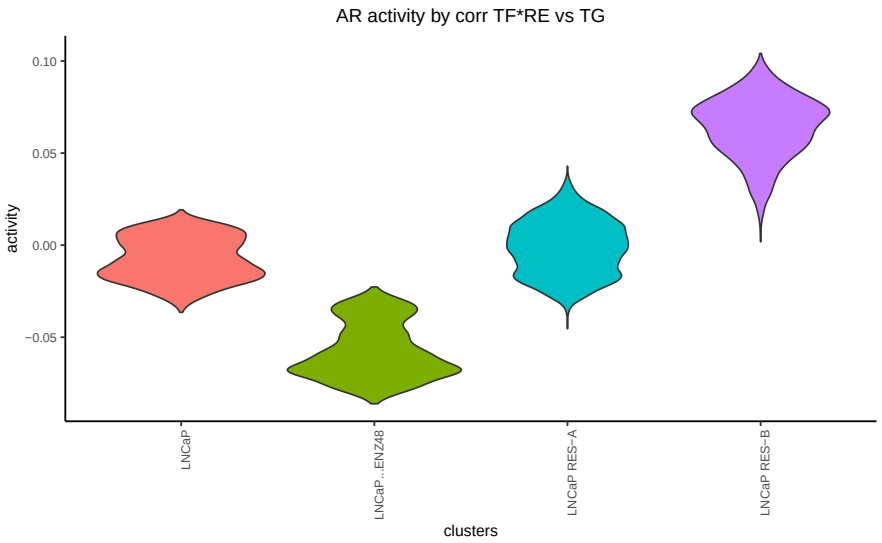
We visualize the different activities side by side

```
plotActivityViolin(activity_matrix = score.combine.wilcox,
                   tf = c( "AR"),
                   clusters = GeneExpressionMatrix$label) + ggtitle ("AR activity by wilcoxon")
```


```
plotActivityViolin(activity_matrix = score.combine.corr,
                   tf = c( "AR"),
                   clusters = GeneExpressionMatrix$label) + ggtitle ("AR activity by corr TF vs TG")
```




```
plotActivityViolin(activity_matrix = score.combine.corr.re,
                   tf = c( "AR"),
                   clusters = GeneExpressionMatrix$label) + ggtitle ("AR activity by corr TF*RE vs TG")
```



4.6 Perform differential activity

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

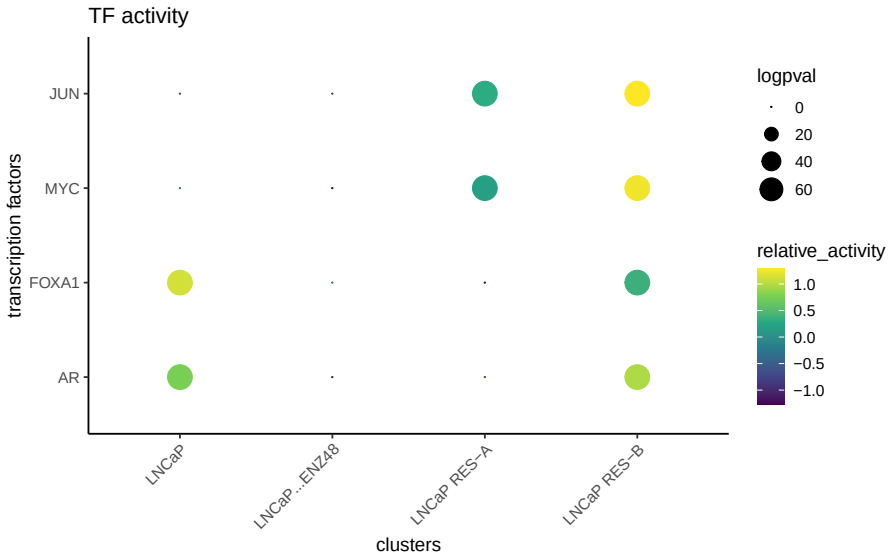
Take the top TFs

```
markers.sig <- getSigGenes(markers, topgenes = 8 )
#> Using a logFC cutoff of 0 for class LNCaP
#> Using a logFC cutoff of 0 for class LNCaP-ENZ48
#> Using a logFC cutoff of 0 for class LNCaP RES-A
#> Using a logFC cutoff of 0 for class LNCaP RES-B
```

4.7 Visualize the results

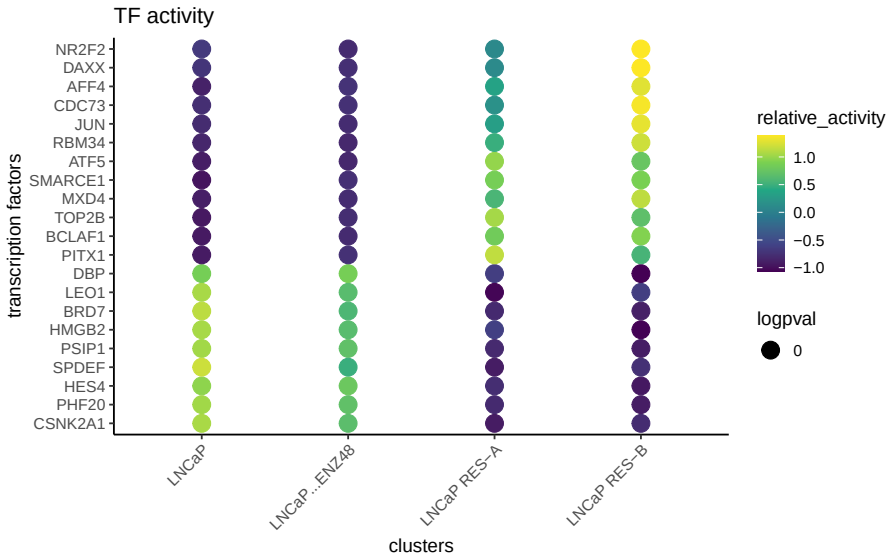
First visualize the known differential TFs by bubble plot

```
plotBubble(activity_matrix = score.combine.wilcox,
           tf = c("AR","FOXA1", "MYC","JUN"),
           clusters = GeneExpressionMatrix$label)
```



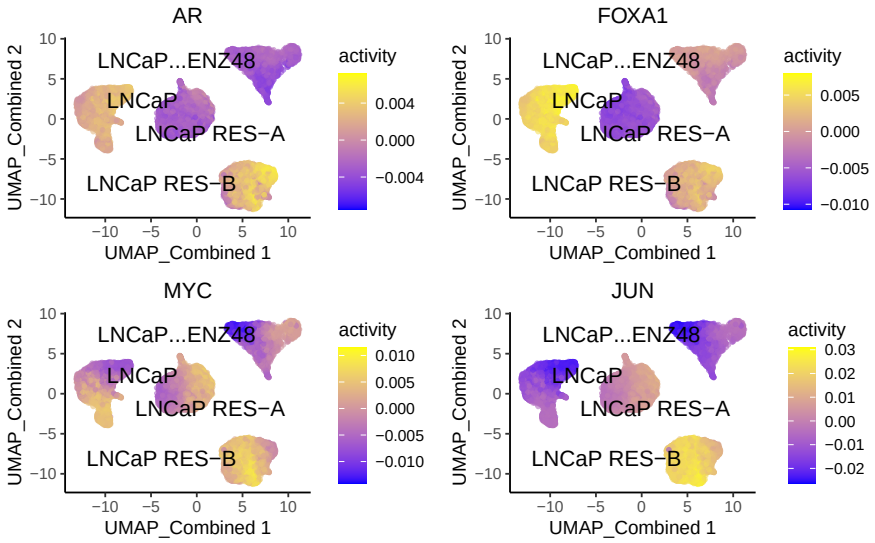
Then visualize the most differential TFs by clusters

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



Visualize the known differential TFs by UMAP

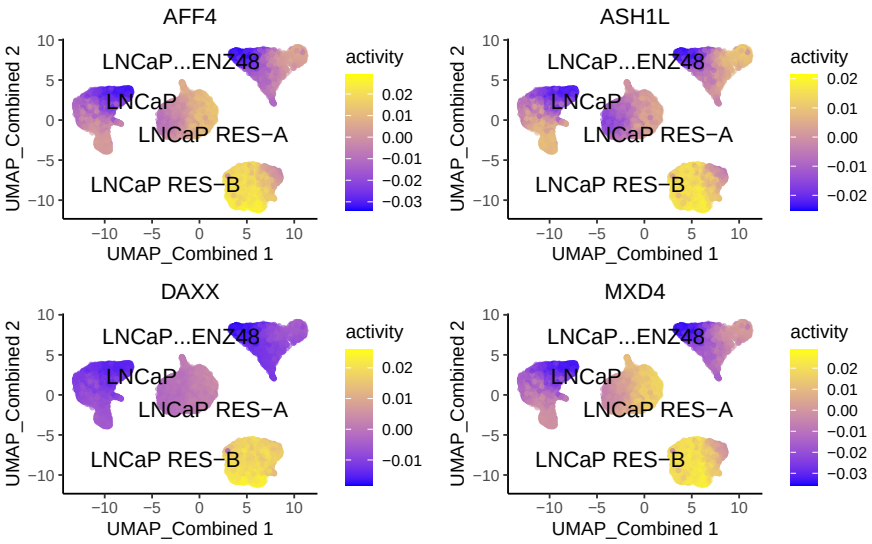
```
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = score.combine.wilcox,
                tf = c( "AR", "FOXA1", "MYC", "JUN"),
                dimtype = "UMAP_Combined",
                label = "label",
                point_size = 1,
                ncol = 2,
                nrow = 2)
```



Visualize the newly discovered differential TFs by UMAP

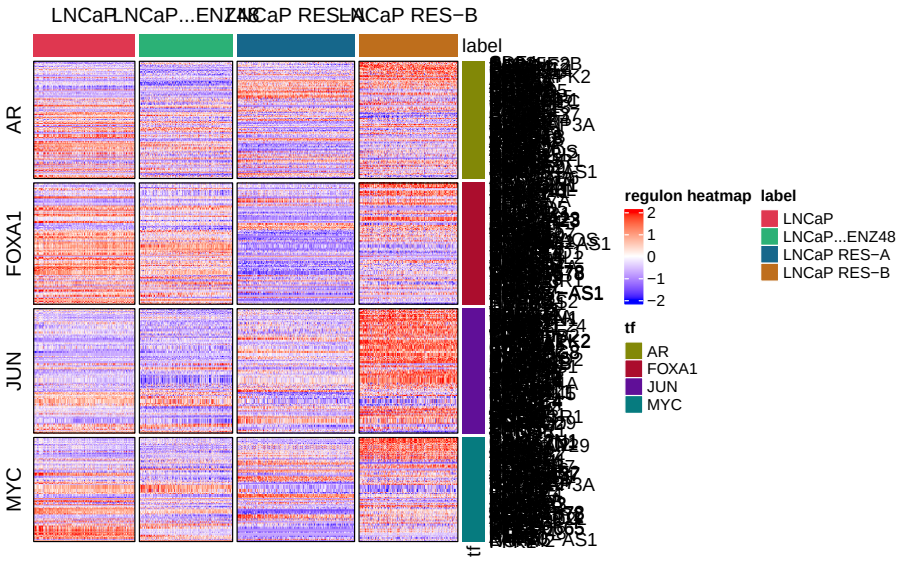
```
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = score.combine.wilcox,
```

```
                tf = c("AFF4","ASH1L","DAXX","MXD4"),
                dimtype = "UMAP_Combined",
                label = "label",
                point_size = 1,
                ncol = 2,
                nrow = 2)
```

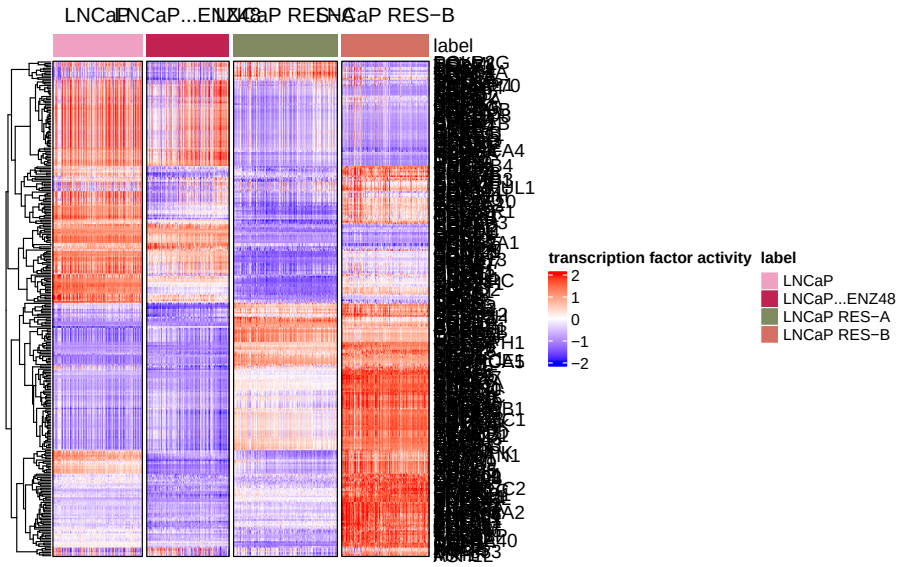


Visualize regulons by heatmap

```
rowData(GeneExpressionMatrix) <- NULL

plotHeatmapRegulon(sce=GeneExpressionMatrix,
                   tfs= c( "AR", "FOXA1", "MYC", "JUN"),
                   regulon=regulon.w.wilcox,
                   regulon_cutoff=0.1,
                   downsample=1000,
                   cell_attributes="label",
                   col_gap="label",
                   exprs_values="logcounts",
                   name="regulon heatmap")
```



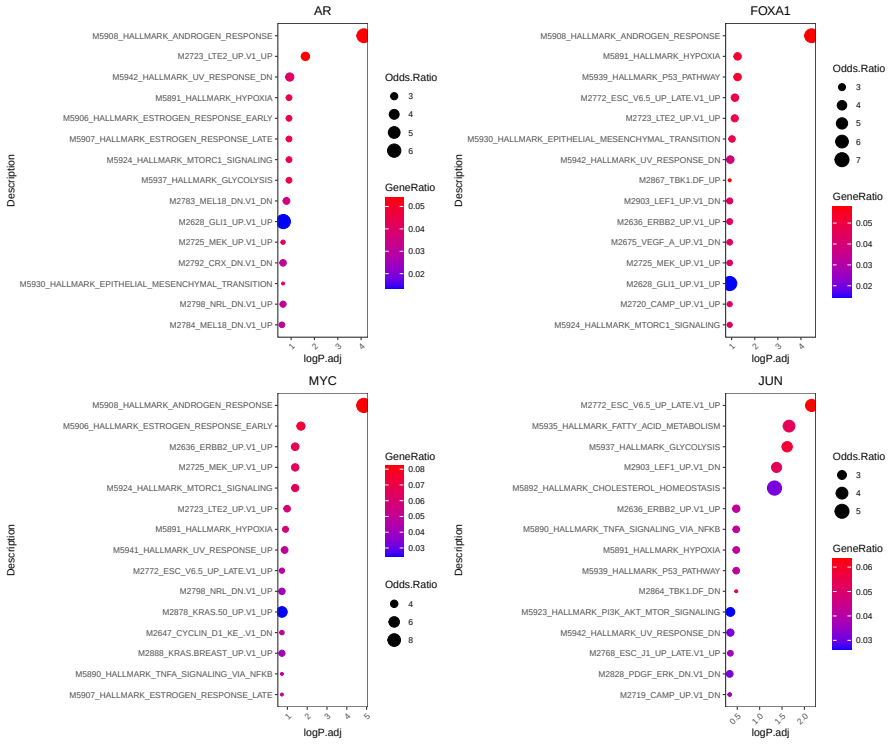

```
plotHeatmapActivity(activity=score.combine.wilcox,
                    sce=GeneExpressionMatrix,
                    tfs=rownames(score.combine.wilcox),
                    downsample=1000,
                    cell_attributes="label",
                    col_gap="label",
                    name = "transcription factor activity")
```



4.8 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.

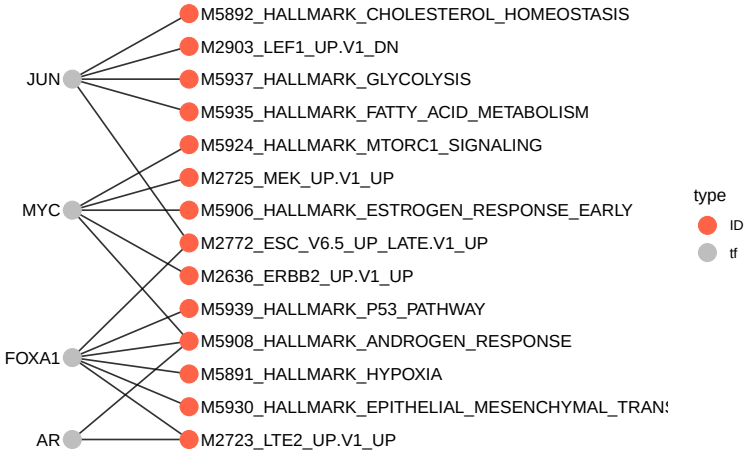
```
#retrieve genesets
H <- EnrichmentBrowser::getGenesets(org = "hsa",
                                    db = "msigdb",
                                    cat = "H",
                                    gene.id.type = "SYMBOL",
                                    cache = FALSE)
C6 <- EnrichmentBrowser::getGenesets(org = "hsa",
                                     db = "msigdb",
                                     cat = "C6",
                                     gene.id.type = "SYMBOL",
                                     cache = FALSE)

#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("AR", "FOXA1", "MYC", "JUN"),
                               regulon = regulon.w.wilcox,
                               weight = "weight",
                               weight_cutoff = 0.1,
                               genesets = gs.list)
#> AR
#>
#> FOXA1
#> MYC
#> JUN

#plot results
enrichPlot(results = enrichresults, ncol = 2)
```



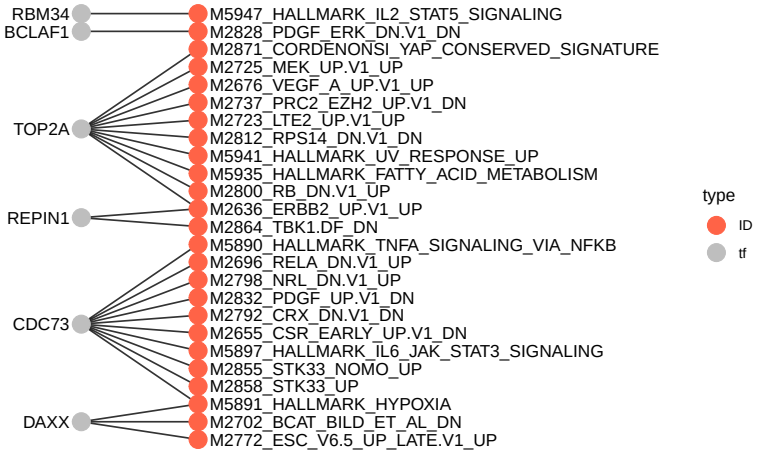
4.9 Network analysis

We can visualize the genesets as a network

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




```
enrichresults <- regulonEnrich(TF = c("AFF4","ASH1L","DAXX","MXD4","CDC73","REPIN1",
                                      "BCLAF1","RBM34","CERS6","TOP2B","ATF5","PITX1",
                                      "EWSR1","TOP2A"),
                               regulon = regulon.w.wilcox,
                               weight = "weight",
                               weight_cutoff = 0.1,
                               genesets = gs.list)
#> AFF4
#> ASH1L
#> DAXX
#> MXD4
#> CDC73
#> REPIN1
#> BCLAF1
#> RBM34
#> CERS6
#> TOP2B
#> ATF5
#> PITX1
#> EWSR1
#> TOP2A
plotGseaNetwork(tf = names(enrichresults),
                enrichresults = enrichresults,
                p.adj_cutoff = 0.1,
                ntop_pathways = 10)
```



5 Session Info

```
sessionInfo()
#> R Under development (unstable) (2022-11-21 r83371)
#> 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
#>  [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] parallel  grid      stats4    stats     graphics  grDevices utils
#>  [8] datasets  methods   base
#>
#> other attached packages:
```

```
#>  [1] org.Hs.eg.db_3.17.0          AnnotationDbi_1.61.2
#>  [3] msigdbr_7.5.1                nabor_0.5.0
#>  [5] scMultiome_0.99.15           MultiAssayExperiment_1.25.11
#>  [7] ExperimentHub_2.7.1          AnnotationHub_3.7.4
#>  [9] BiocFileCache_2.7.2          dbplyr_2.3.2
#> [11] rhdf5_2.43.5                 RcppArmadillo_0.12.2.0.0
#> [13] Rcpp_1.0.10                  Matrix_1.5-3
#> [15] sparseMatrixStats_1.11.1     data.table_1.14.8
#> [17] stringr_1.5.0                plyr_1.8.8
#> [19] magrittr_2.0.3               ggplot2_3.4.2
#> [21] gtable_0.3.3                 gtools_3.9.4
#> [23] gridExtra_2.3                devtools_2.4.5
#> [25] usethis_2.1.6                ArchR_1.0.3
#> [27] BiocStyle_2.27.2             epiregulon_1.0.25
#> [29] SingleCellExperiment_1.21.1  SummarizedExperiment_1.29.1
#> [31] Biobase_2.59.0               GenomicRanges_1.51.4
#> [33] GenomeInfoDb_1.35.17         IRanges_2.33.1
#> [35] S4Vectors_0.37.7             BiocGenerics_0.45.3
#> [37] MatrixGenerics_1.11.1        matrixStats_0.63.0
#>
#> loaded via a namespace (and not attached):
#>   [1] fs_1.6.2                          GSVA_1.47.3
#>   [3] bitops_1.0-7                      enrichplot_1.19.2
#>   [5] HDO.db_0.99.1                     httr_1.4.5
#>   [7] RColorBrewer_1.1-3                doParallel_1.0.17
#>   [9] Rgraphviz_2.43.0                  profvis_0.3.7
#>  [11] tools_4.3.0                       backports_1.4.1
#>  [13] utf8_1.2.2                        R6_2.5.1
#>  [15] HDF5Array_1.27.0                  lazyeval_0.2.2
#>  [17] rhdf5filters_1.11.2               GetoptLong_1.0.5
#>  [19] urlchecker_1.0.1                  withr_2.5.0
#>  [21] prettyunits_1.1.1                 cli_3.6.1
#>  [23] Cairo_1.6-0                       scatterpie_0.1.9
#>  [25] labeling_0.4.2                    KEGGgraph_1.59.3
#>  [27] yulab.utils_0.0.6                 Rsamtools_2.15.3
#>  [29] gson_0.1.0                        DOSE_3.25.0
#>  [31] R.utils_2.12.2                    scater_1.27.9
#>  [33] sessioninfo_1.2.2                 BSgenome_1.67.4
#>  [35] limma_3.55.10                     rstudioapi_0.14
#>  [37] RSQLite_2.3.1                     gridGraphics_0.5-1
#>  [39] generics_0.1.3                    shape_1.4.6
#>  [41] BiocIO_1.9.2                      dplyr_1.1.2
#>  [43] GO.db_3.17.0                      ggbeeswarm_0.7.1
#>  [45] fansi_1.0.3                       R.methodsS3_1.8.2
#>  [47] lifecycle_1.0.3                   yaml_2.3.6
#>  [49] edgeR_3.41.9                      qvalue_2.31.1
#>  [51] blob_1.2.4                        promises_1.2.0.1
#>  [53] dqrng_0.3.0                       crayon_1.5.2
#>  [55] miniUI_0.1.1.1                    lattice_0.20-45
#>  [57] beachmat_2.15.2                   cowplot_1.1.1
#>  [59] annotate_1.77.0                   KEGGREST_1.39.0
```

```
#>  [61] magick_2.7.4                        pillar_1.9.0
#>  [63] knitr_1.42                          ComplexHeatmap_2.15.3
#>  [65] metapod_1.7.0                       fgsea_1.25.2
#>  [67] rjson_0.2.21                        codetools_0.2-18
#>  [69] fastmatch_1.1-3                     glue_1.6.2
#>  [71] ggfun_0.0.9                         downloader_0.4
#>  [73] remotes_2.4.2                       treeio_1.23.1
#>  [75] vctrs_0.6.2                         png_0.1-8
#>  [77] cachem_1.0.6                        xfun_0.39
#>  [79] mime_0.12                           tidygraph_1.2.3
#>  [81] iterators_1.0.14                    statmod_1.5.0
#>  [83] bluster_1.9.1                       interactiveDisplayBase_1.37.0
#>  [85] ellipsis_0.3.2                      nlme_3.1-162
#>  [87] ggtree_3.7.2                        bit64_4.0.5
#>  [89] filelock_1.0.2                      irlba_2.3.5.1
#>  [91] vipor_0.4.5                         colorspace_2.1-0
#>  [93] DBI_1.1.3                           tidyselect_1.2.0
#>  [95] processx_3.8.0                      bit_4.0.5
#>  [97] compiler_4.3.0                      curl_4.3.3
#>  [99] AUCell_1.21.2                       graph_1.77.3
#> [101] BiocNeighbors_1.17.1                BSgenome.Mmusculus.UCSC.mm10_1.4.3
#> [103] DelayedArray_0.25.0                 shadowtext_0.1.2
#> [105] bookdown_0.33                       rtracklayer_1.59.1
#> [107] checkmate_2.1.0                     scales_1.2.1
#> [109] callr_3.7.3                         rappdirs_0.3.3
#> [111] digest_0.6.30                       rmarkdown_2.21
#> [113] BSgenome.Hsapiens.UCSC.hg19_1.4.3   XVector_0.39.0
#> [115] htmltools_0.5.5                     pkgconfig_2.0.3
#> [117] fastmap_1.1.0                       rlang_1.1.0
#> [119] GlobalOptions_0.1.2                 htmlwidgets_1.6.2
#> [121] shiny_1.7.4                         DelayedMatrixStats_1.21.0
#> [123] farver_2.1.1                        jsonlite_1.8.4
#> [125] BiocParallel_1.33.12                GOSemSim_2.25.0
#> [127] R.oo_1.25.0                         BiocSingular_1.15.0
#> [129] RCurl_1.98-1.12                     ggplotify_0.1.0
#> [131] scuttle_1.9.4                       GenomeInfoDbData_1.2.10
#> [133] patchwork_1.1.2                     Rhdf5lib_1.21.1
#> [135] munsell_0.5.0                       ape_5.7-1
#> [137] babelgene_22.9                      viridis_0.6.2
#> [139] EnrichmentBrowser_2.29.2            stringi_1.7.8
#> [141] ggraph_2.1.0                        MASS_7.3-58.1
#> [143] zlibbioc_1.45.0                     pkgbuild_1.4.0
#> [145] ggrepel_0.9.3                       graphlayouts_0.8.4
#> [147] splines_4.3.0                       Biostrings_2.67.2
#> [149] circlize_0.4.15                     locfit_1.5-9.7
#> [151] BSgenome.Hsapiens.UCSC.hg38_1.4.5   ps_1.7.2
#> [153] igraph_1.4.2                        reshape2_1.4.4
#> [155] ScaledMatrix_1.7.1                  pkgload_1.3.2
#> [157] BiocVersion_3.17.1                  XML_3.99-0.14
#> [159] evaluate_0.20                       scran_1.27.3
#> [161] BiocManager_1.30.20                 tweenr_2.0.2
```

```
#> [163] foreach_1.5.2                    httpuv_1.6.9
#> [165] polyclip_1.10-4                  tidyr_1.3.0
#> [167] purrr_1.0.1                      clue_0.3-64
#> [169] ggforce_0.4.1                    rsvd_1.0.5
#> [171] xtable_1.8-4                     restfulr_0.0.15
#> [173] tidytree_0.4.2                   later_1.3.0
#> [175] viridisLite_0.4.1                tibble_3.2.1
#> [177] aplot_0.1.10                     clusterProfiler_4.7.1
#> [179] memoise_2.0.1                    beeswarm_0.4.0
#> [181] GenomicAlignments_1.35.1         cluster_2.1.4
#> [183] GSEABase_1.61.2
```