TreeCorTreat: Tree-based correlation screen for phenotype-associated transcriptomic features and cell types

Boyang Zhang Zhicheng Ji Hongkai Ji

Overview

Single-cell RNA-seq experiments with multiple samples are increasingly used to discover cell types and their molecular features that may influence samples' phenotype (e.g. disease). However, analyzing and visualizing the complex cell type-phenotype association remains nontrivial. TreeCorTreat is an open source R package that tackles this problem by using a tree-based correlation screen to analyze and visualize the association between phenotype and transcriptomic features and cell types at multiple cell type resolution levels. With TreeCorTreat, one can conveniently explore and compare different feature types, phenotypic traits, analysis protocols and datasets, and evaluate the impacts of potential confounders.

TreeCorTreat takes a gene expression matrix (raw count), cell-level metadata and sample-level metadata as input. It provides a whole pipeline to integrate data across samples, identify cell clusters and their hierarchical structure, evaluate the association between sample phenotype and cell type at different resolution levels in terms of both cell type proportion and gene expression, and summarize and visualize the results in a tree structured TreeCorTreat plot. This pipeline consists of six functional modules:

- Module 1: Data integration
- Module 2: Define cell types at multiple resolutions
- Module 3: Identify association between cell type proportion and sample phenotype
- Module 4: Identify association between global gene expression and sample phenotype
- Module 5: Identify differentially expressed genes
- Module 6: Visualization via TreeCorTreat plot

The modular structure provides users with the flexibility to skip certain analysis steps and replace them by users' own data or analysis functions.

For more details, please check our paper descrbing the **TreeCorTreat** package: - Tree-based Correlation Screen and Visualization for Exploring Phenotype-Cell Type Association in Multiple Sample Single-Cell RNA-Sequencing Experiments. [bioRxiv link]

Input data and data preparation

The input for TreeCorTreat consists of three components: a gene expression matrix E (G×C, with rows representing G genes and columns representing C cells), cell-level metadata M (C×K, where $K \geq 2$) and sample-level metadata L (S×J, where $J \geq 2$). To avoid redundancy and save storage space, we split metadata into cell-level and sample-level metadata. Cell-level metadata primarily annotates cell-level information and must contain at least 2 columns for each cell: cell barcode and corresponding sample identifier (ID). An optional third column can be added to provide cell-type annotation. Cell barcode is used to couple cell-level metadata M and gene expression matrix E. Sample-level metadata documents a sample's phenotype(s) of interests (e.g. clinical outcome) and other related covariates (e.g. age and sex). The first column contains unique sample IDs, and the remaining columns contain phenotypes and covariates. The cell-level metadata and sample-level metadata can be linked via unique sample IDs.

Here, we include 8 single-cell RNA-seq PBMC samples downloaded from **ArrayExpress: E-MTAB-9357** as an example to illustrate TreeCorTreat pipeline.

```
# load in TreeCorTreat
options(warn = -1)
suppressMessages(library(TreeCorTreat))
suppressMessages(library(ggplot2))
suppressMessages(library(dplyr))
suppressMessages(library(tidyr))

data(raw_data)
str(raw_data)
```

```
## List of 3
##  $ sample_meta:'data.frame': 8 obs. of  5 variables:
##   ..$ sample  : chr [1:8] "HD-17-Su" "HD-19-Su" "HD-23-Su" "HD-30-Su" ...
##   ..$ study   : chr [1:8] "Su" "Su" "Su" "Su" ...
##   ..$ age     : num [1:8] 52 79 26 35 33 63 88 84
##   ..$ sex     : chr [1:8] "F" "M" "F" "F" ...
##   ..$ severity: chr [1:8] "HD" "HD" "HD" "HD" ...
##  $ cell_meta  :'data.frame': 16572 obs. of  2 variables:
##   ..$ barcode: chr [1:16572] "HD-17-Su:345974" "HD-17-Su:345975" "HD-17-Su:345976" "HD-17-Su:345977"
##   ..$ sample : chr [1:16572] "HD-17-Su" "HD-17-Su" "HD-17-Su" "HD-17-Su" ...
##  $ count      :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. ..@ i       : int [1:17088466] 10 77 81 419 683 873 1496 2208 2218 2229 ...
##   .. ..@ p       : int [1:16573] 0 796 1378 2418 3480 4540 5982 7295 8011 8927 ...
##   .. ..@ Dim     : int [1:2] 19668 16572
##   .. ..@ Dimnames:List of 2
##   .. .. ..$ : chr [1:19668] "A1BG" "A1BG-AS1" "A2M" "A2M-AS1" ...
##   .. .. ..$ : chr [1:16572] "HD-17-Su:345974" "HD-17-Su:345975" "HD-17-Su:345976" "HD-17-Su:345977"
##   .. ..@ x       : num [1:17088466] 1 1 1 7 1 1 1 1 1 1 ...
##   .. ..@ factors : list()
```

raw_data is a list that contains three elements: sample-level metadata, cell-level metadata and gene expression matrix (raw count). There are 8 samples (4 healthy donors (HD) and 4 Severe (Se) COVID-19 samples), 16572 cells and 19668 genes.

Module 1: Data integration

Harmony algorithm is applied to integrate cells from different samples and embed them into a common low-dimensional space. We adapt the `RunHarmony` and `BuildClusterTree` functions from **Seurat v3** and wrap the following steps into a standalone **`treecor_harmony`** function with tunable parameters: library size normalization, integration feature selection, Principal Component Analysis (PCA), Harmony integration, unsupervised louvain clustering, Uniform Manifold Approximation and Projection (UMAP), hierarchical clustering of louvain clusters, and differentially expression analysis to identify cell type marker genes to facilitate annotation.

Specifically, for library size normalization, gene counts for a given cell are divided by total counts for that cell, multiplied by a scaling factor (10^4) and applied a natural log transformation. Features for integrating samples are obtained by choosing highly variable genes (HVGs) based on variance-mean relationship within each individual sample (using `SelectIntegrationFeatures` function) and ranking the features based on the number of samples where they are identified as HVGs. By default, top 2000 HVGs are chosen and fed into downstream PCA procedure. Harmony is carried out based on the top 20 PCs and the corrected harmony embedding is obtained. Top 20 harmony coordinates are then used for downstream louvain clustering and UMAP analyses (using `FindClusters` and `RunUMAP` functions in Seurat).

```
# integration
set.seed(12345)
integration <- treecor_harmony(count = raw_data[["count"]], sample_meta = raw_data[["sample_meta"]],
    output_dir = getwd())
```

This step is the most time-consuming and RAM intensive module in our evaluation. It may take up to days to run using a large dataset ($>10^5$ cells). Instead of using this default integration pipeline, users can also use their own integration results in a Seurat object or provide cell cluster labels in an additional **'celltype'** column in the cell-level metadata and skip the integration step.

The integrated Seurat object will be stored in local directory and filtered gene expression matrix or cell-level metadata can be extracted by `access_data_seurat`, which will be used for downstream analyses.

```
# list integration result and data extracted from Seurat
# object
integrated_data <- access_data_seurat(seurat_obj = integration,
    output_dir = getwd())
```

In our example, no cell has been filtered out and we have stored updated cell-level metadata (with additional columns including 'celltype' and UMAP coordinates) as **integrated_cellmeta**.

```
data(integrated_cellmeta)
head(integrated_cellmeta)
```

```
##           barcode   sample celltype    UMAP_1     UMAP_2
## 1 HD-17-Su:345974 HD-17-Su        3 -3.393676 -7.0051720
## 2 HD-17-Su:345975 HD-17-Su        2 -5.090366  6.9038575
## 3 HD-17-Su:345976 HD-17-Su        2 -6.407369  6.9599665
## 4 HD-17-Su:345977 HD-17-Su        5 -6.225506  3.5054925
## 5 HD-17-Su:345978 HD-17-Su        1  8.189426  0.8951219
## 6 HD-17-Su:345979 HD-17-Su        9  9.156926 -4.1967079
```

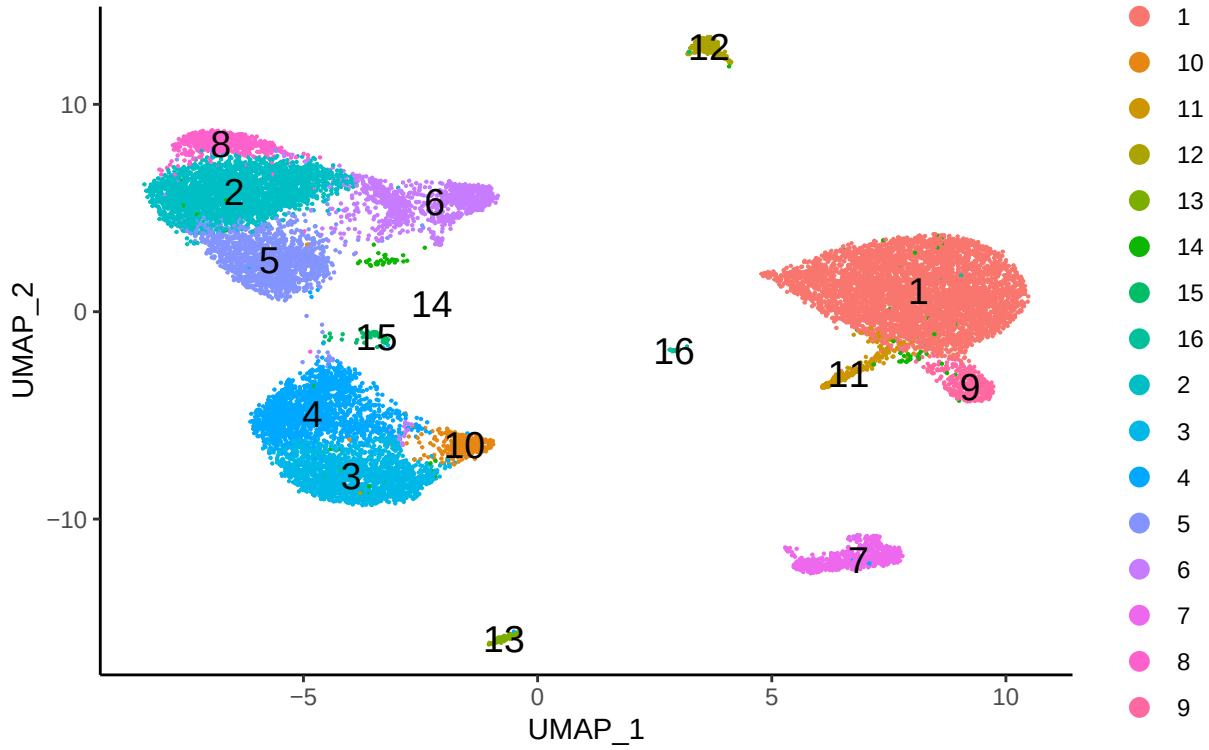
```
# data for downstream analysis
sample_meta <- raw_data[["sample_meta"]]
count <- raw_data[["count"]]
cell_meta <- integrated_cellmeta
```

Module 2: Define cell types at multiple resolutions

The previous steps defined cell clusters either by louvain clustering or based on user-provided cell-level metadata (i.e. the optional column 3 in the cell metadata). Users can add text labels to annotate the cell type of each cell cluster based on top differentially expressed genes or known cell type marker genes. Here, we overlay several gene markers (e.g. CD3D, CD19 and CD68) on UMAP to roughly annotate cell clusters.

```
# cell clusters
label_text <- integrated_cellmeta %>%
    group_by(celltype) %>%
    summarise(UMAP_1 = median(UMAP_1), UMAP_2 = median(UMAP_2))

ggplot(integrated_cellmeta, aes(x = UMAP_1, y = UMAP_2, color = celltype)) +
    geom_point(size = 0.1) + geom_text(data = label_text, aes(x = UMAP_1,
    y = UMAP_2, label = celltype), color = "black", size = 5) +
    theme_classic() + guides(color = guide_legend(override.aes = list(size = 3)))
```

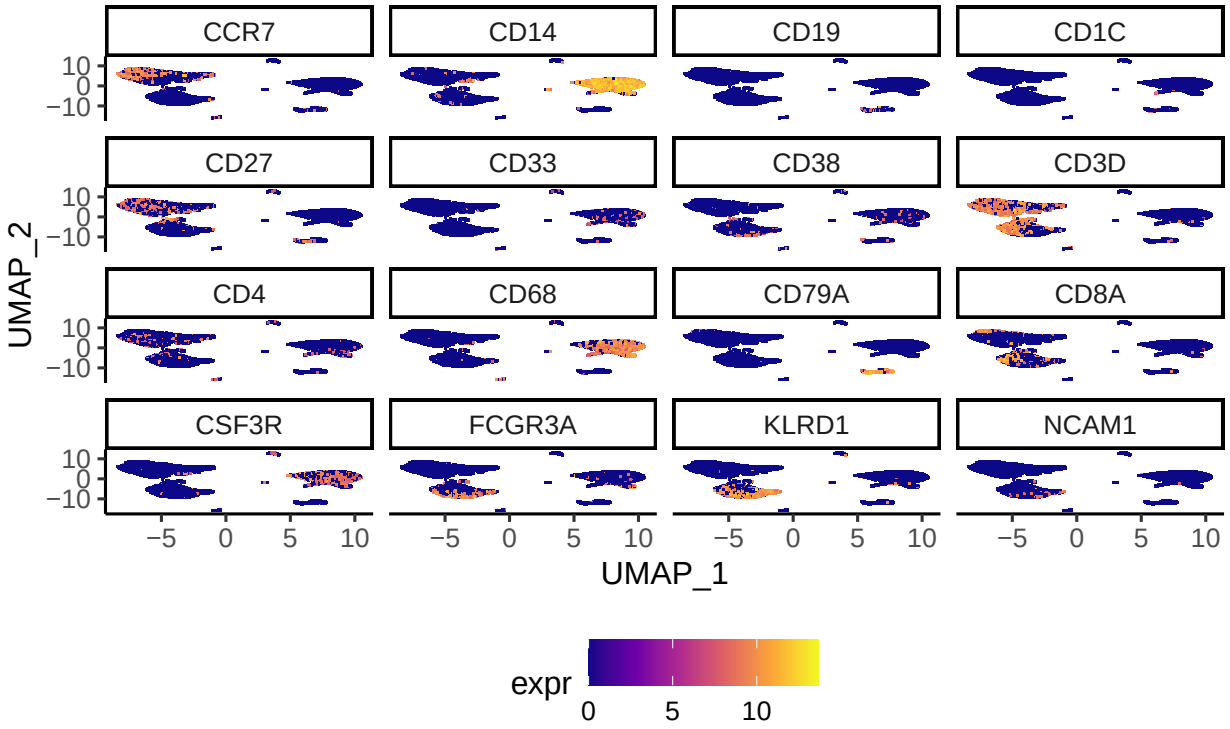



```
# gene markers
genes <- c("CD3D", "CD14", "CD19", "NCAM1", "CD4", "CD8A", "FCGR3A",
    "CD1C", "CD68", "CD79A", "CSF3R", "CD33", "CCR7", "CD38",
    "CD27", "KLRD1")

rc <- Matrix::colSums(count)
sub_count <- count[genes, ] %>%
    as.matrix
sub_norm <- log2(t(t(sub_count)/rc * 1e+06 + 1)) %>%
    as.matrix

df_marker <- t(sub_norm) %>%
    data.frame %>%
    mutate(barcode = rownames(.)) %>%
    gather(gene, expr, -barcode) %>%
    inner_join(cell_meta %>%
        select(barcode, UMAP_1, UMAP_2))

ggplot(df_marker, aes(x = UMAP_1, y = UMAP_2, col = expr)) +
    geom_point(size = 0.01, shape = ".") + scale_colour_viridis_c(option = "C",
    direction = 1) + facet_wrap(~gene) + theme_classic(base_size = 12) +
    theme(legend.position = "bottom")
```



We roughly categorize 16 cell clusters into 3 large cell-types based on gene markers: B cells (CD19), T cells (CD3D) and Monocytes (CD14).

```
# new celltype annotation
new_label <- data.frame(old = sort(factor(unique(cell_meta$celltype,
    levels = 1:16)))) %>%
    mutate(large_celltype = ifelse(old %in% c(1, 9, 11, 12, 13,
        16), "Mono", ifelse(old %in% c(2:6, 8, 10, 14, 15), "T",
        "B")), new = paste0(large_celltype, "_c", old)) %>%
    select(-large_celltype)
new_label
```

```
##    old      new
## 1    1  Mono_c1
## 2   10    T_c10
## 3   11 Mono_c11
## 4   12 Mono_c12
## 5   13 Mono_c13
## 6   14    T_c14
## 7   15    T_c15
## 8   16 Mono_c16
## 9    2     T_c2
## 10   3     T_c3
## 11   4     T_c4
## 12   5     T_c5
## 13   6     T_c6
## 14   7     B_c7
## 15   8     T_c8
## 16   9  Mono_c9
```

Modify cell type annotation (optional)

Users can modify the cell type label via `modify_label` function to update cell type annotations in `cell_meta`:

```
# modify cell type names in `cell_meta`
cell_meta <- modify_label(new_label, hierarchy_list = NULL, cell_meta)$cell_meta
```

```
## Modify label in cell_meta
```

```
head(cell_meta[, c("barcode", "sample", "celltype")])
```

```
##           barcode   sample celltype
## 1 HD-17-Su:345974 HD-17-Su     T_c3
## 2 HD-17-Su:345975 HD-17-Su     T_c2
## 3 HD-17-Su:345976 HD-17-Su     T_c2
## 4 HD-17-Su:345977 HD-17-Su     T_c5
## 5 HD-17-Su:345978 HD-17-Su  Mono_c1
## 6 HD-17-Su:345979 HD-17-Su  Mono_c9
```

Construct hierarchical tree structure

To facilitate association analyses at multiple resolutions, cell clusters are further clustered hierarchically. The tree can be derived using either a data-driven approach or a knowledge-based approach.

The tree can be either provided by users based on prior knowledge (knowledge-based) or derived from the data using hierarchical clustering (data-driven). For data-driven approach, a phylogenetic tree is built on PC space by applying `treecor_harmony` function (or `BuildClusterTree` from Seurat R package).

Data-driven approach

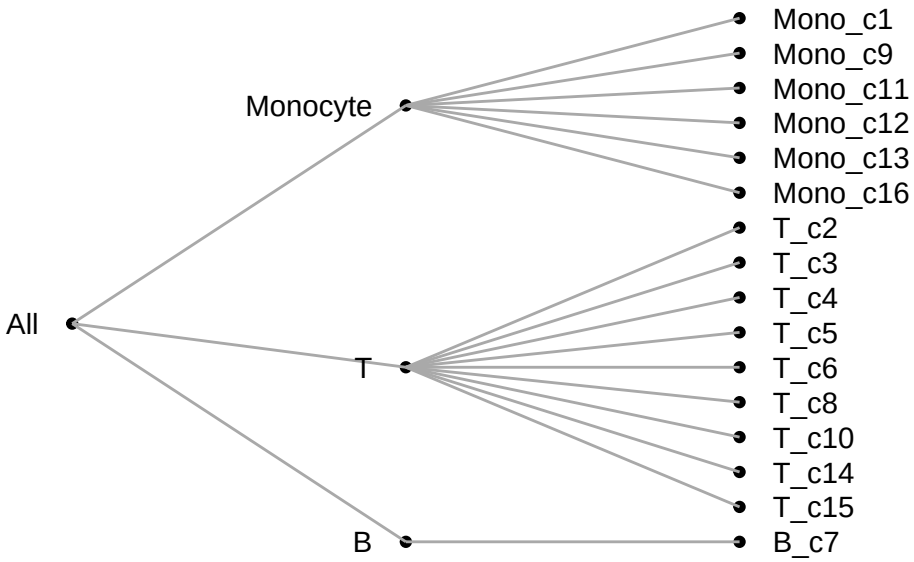
In the data-driven approach, the tree is generated by the hierarchical clustering of the louvain clusters by applying `treecor_harmony` function (or `BuildClusterTree` from Seurat R package). Specifically, PC scores are first averaged across cells within each cell cluster and a hierarchical tree is constructed. The data-driven approach could provide an unbiased way to infer underlying tree structure, but it can be challenging to annotate every intermediate tree nodes.

Knowledge-based approach

In the knowledge-based approach, users can specify the tree based on their prior knowledge by providing a string to describe the parent-children relationship of clusters at different granularity levels. The leaf nodes in the tree correspond to cell clusters obtained from either the louvain clustering in the integration step or the 'celltype' column from cell metadata.

```
# specify string
input_string <- "@All(@B(B_c7),@T(T_c15,T_c14,T_c10,T_c8,T_c6,T_c5,T_c4,T_c3,T_c2),@Monocyte(Mono_c16,M

# extract hierarchy from string
hierarchy_structure <- extract_hrchy_string(input_string, special_character = "@",
    plot = T)
```



For demonstration purpose, we will use knowledge-based hierarchical structure in subsequent analysis.

Module 3: Identify association between cell type proportion and sample phenotype

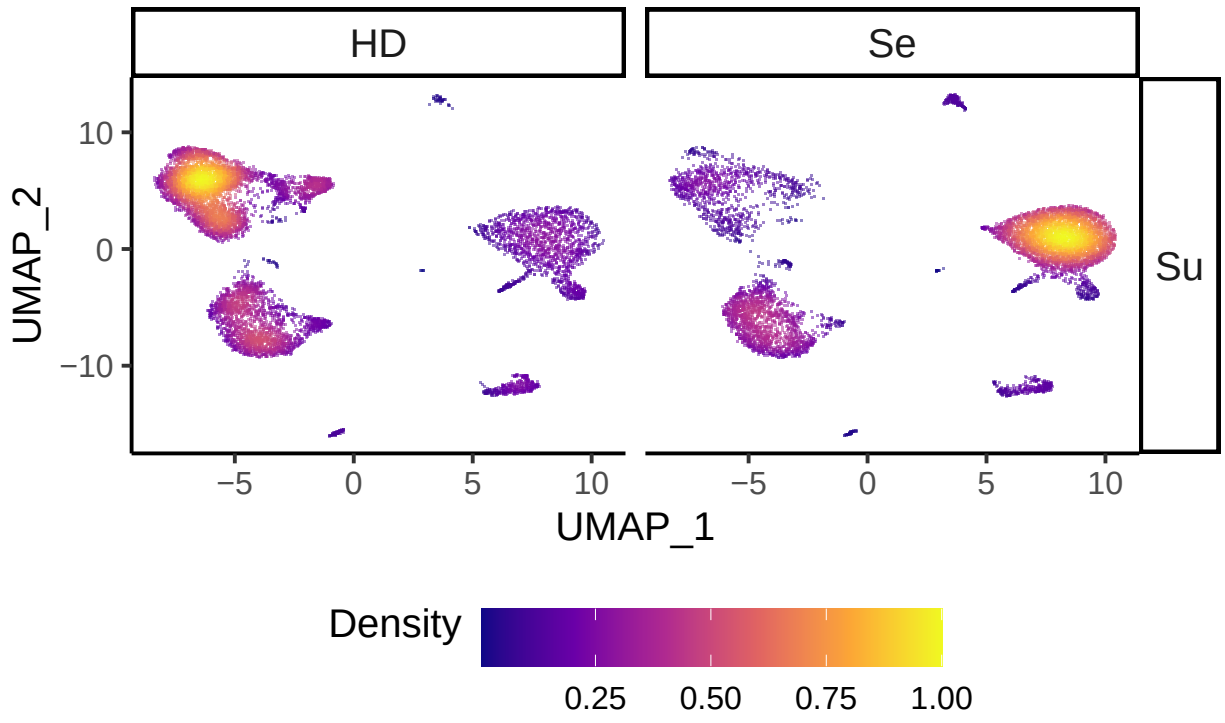
Analyses of cell type proportions are wrapped into a standalone function **`treecor_ctprop`**. Here we first define the feature for each tree node and then evaluate the association between the feature and the sample phenotype. For a leaf node cell cluster, the feature is defined as the proportion of cells in a sample that fall into that node. For an intermediate parent node or root node, users can choose to define the feature in one of **Aggregate** (default setting), **Concatenate leaf nodes** or **Concatenate immediate children**. Once the feature for each node is defined, we will go through each tree node to examine the correlation (or other summary statistic) between its feature and the sample phenotype. The way to compute the summary statistic depends on whether the phenotype is univariate or multivariate:

- Univariate phenotype/Multivariate phenotype analyzed separately:
- Pearson correlation (default) with correlation sign
- Spearman correlation with correlation sign
- Canonical correlation
- Chi-squared statistic (via likelihood ratio test between a full model (with phenotype as explanatory variables) and a reduced model (intercept-only)) with coefficient sign
- Multivariate phenotype analyzed jointly:
- Canonical correlation
- Chi-squared statistic (via likelihood ratio test)

The permutation-based p-value can be obtained and adjusted p-value is computed using Benjamini & Yekutieli procedure.

We can first use a kernel density plot to visualize cell density stratified by disease severity.

```
# density plot on UMAP embeddings
treecor_celldensityplot(cell_meta, sample_meta, row_variable = "study",
    col_variable = "severity", row_combined = F)
```



Next, we can assess the association between cell type proportion and disease severity using `treecor_ctprop()` pipeline:

```
# cell type prop pipeline (default)
res_ctprop_full <- treecor_ctprop(hierarchy_structure, cell_meta,
    sample_meta, response_variable = "severity", method = "aggregate",
    analysis_type = "pearson", num_permutations = 100)
names(res_ctprop_full)
```

```
## [1] "canonical_corr" "pc_ls"
```

The first element is a table of computed summary statistic (e.g. pearson correlation) along with p-value and adjusted p-value. The second element is a list of PC matrices for each cell type. However, in default setting (aggregate), proportion is aggregated as a vector (1-dimension) and thus we do not conduct PCA.

```
# extract Pearson correlation
res_ctprop <- res_ctprop_full[[1]] %>%
    mutate(severity.absolute_cor = abs(severity.pearson))
head(res_ctprop)
```

```
##   id severity.method severity.analysis_type severity.pearson severity.p
## 1  1       aggregate                pearson               NA         NA
## 2  2       aggregate                pearson        0.1147058 0.82828283
## 3  3       aggregate                pearson       -0.7614817 0.08080808
## 4  4       aggregate                pearson        0.7603030 0.08080808
## 5  5       aggregate                pearson        0.1147058 0.82828283
## 6  6       aggregate                pearson        0.5039696 0.11111111
##   severity.adjp severity.direction severity.p.sign severity.adjp.sign     x y
## 1            NA               <NA>            <NA>               <NA>  6.25 2
## 2     1.0000000                  +              ns                 ns  0.00 1
## 3     0.7781478                  -              ns                 ns  5.00 1
```

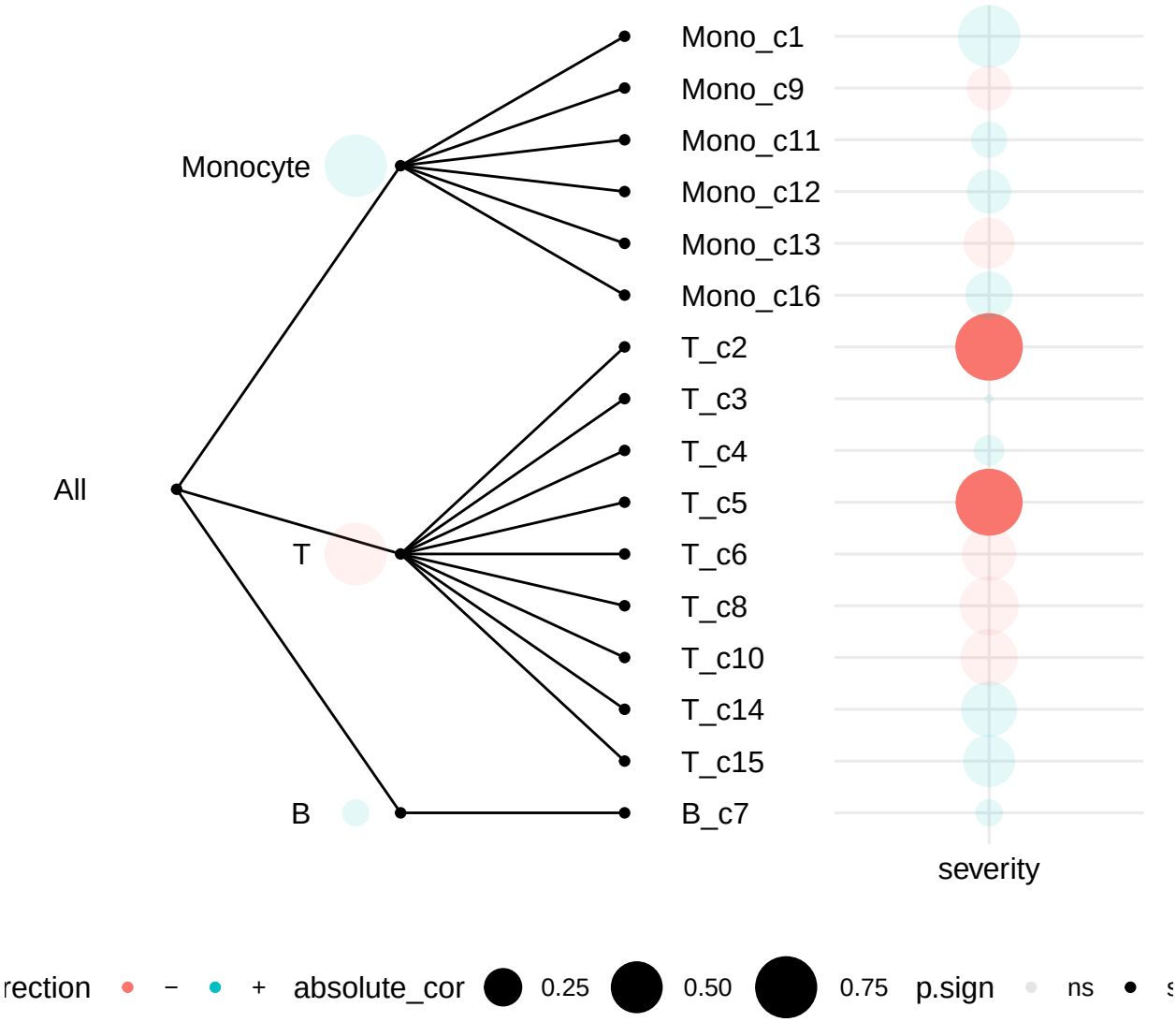
```
## 4     0.7781478                  +             ns                 ns 12.50 1
## 5     1.0000000                  +             ns                 ns  0.00 0
## 6     0.8321858                  +             ns                 ns  1.00 0
##      label  leaf severity.absolute_cor
## 1      All FALSE                    NA
## 2        B FALSE             0.1147058
## 3        T FALSE             0.7614817
## 4 Monocyte FALSE             0.7603030
## 5     B_c7  TRUE             0.1147058
## 6    T_c15  TRUE             0.5039696
```

```
colnames(res_ctprop)
```

```
##  [1] "id"                    "severity.method"       "severity.analysis_type"
##  [4] "severity.pearson"      "severity.p"            "severity.adjp"
##  [7] "severity.direction"    "severity.p.sign"       "severity.adjp.sign"
## [10] "x"                     "y"                     "label"
## [13] "leaf"                  "severity.absolute_cor"
```

Then we can use **TreeCorTreat plot** to visualize the result. Users can specify variables for different aesthetic (e.g. color, size, alpha). We will discuss TreeCorTreat plot in details later.

```
# visualize
treecortreatplot(hierarchy_structure, annotated_df = res_ctprop,
    response_variable = "severity", color_variable = "direction",
    size_variable = "absolute_cor", alpha_variable = "p.sign",
    font_size = 12, nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2)
```



Module 4: Identify association between global gene expression and sample phenotype

Analyses of global gene expression are wrapped into a standalone function **`treecor_expr`**. Here we also first define the feature for each tree node and then evaluate the global gene expression-sample phenotype association. For a leaf node, we first pool all cells in the node to obtain a pseudobulk profile of the node. We then use these pseudobulk profiles from all samples to select highly variable genes using locally weighted scatterplot smoothing (LOWESS) procedure in R. The pseudobulk profile of the selected highly variable genes in each sample is used as the feature vector of the node. This feature vector is multivariate. For non-leaf nodes, users can choose to define the feature in one of **Aggregate** (default setting), **Concatenate leaf nodes** or **Concatenate immediate children**. Once the feature for each node is defined, we will go through each tree node to examine the correlation (or other summary statistic) between its feature and the sample phenotype. Users can choose either canonical correlation or F-statistic (Pillai-Bartlett, comparing a full model (with phenotype as explanatory variables) with a reduced model (intercept-only)).

```
# gene expression pipeline
res_expr_full <- treecor_expr(count, hierarchy_structure, cell_meta,
    sample_meta, response_variable = "severity", method = "aggregate",
    analysis_type = "cancor", num_permutations = 100)
names(res_expr_full)
```

```
## [1] "canonical_corr" "pc_ls"
```

The first element is a table of computed summary statistic (e.g. canonical correlation) along with p-value and adjusted p-value. The second element is a list of PC matrices for each cell type.

```
# extract canonical correlation
res_expr <- res_expr_full[[1]]
head(res_expr)
```

```
##   id severity.cancor severity.p severity.adjp severity.direction
## 1  1       0.8942939 0.03030303     0.4360897                  +
## 2  2       0.6428883 0.15151515     0.9911129                  +
## 3  3       0.9297493 0.03030303     0.4360897                  +
## 4  4       0.8324347 0.03030303     0.4360897                  +
## 5  5       0.6428883 0.15151515     0.9911129                  +
## 6  6       0.8807827 0.04040404     0.4845441                  +
##   severity.p.sign severity.adjp.sign     x y    label  leaf
## 1             sig                 ns  6.25 2      All FALSE
## 2              ns                 ns  0.00 1        B FALSE
## 3             sig                 ns  5.00 1        T FALSE
## 4             sig                 ns 12.50 1 Monocyte FALSE
## 5              ns                 ns  0.00 0     B_c7  TRUE
## 6             sig                 ns  1.00 0    T_c15  TRUE
```

```
# extract PCA list
pc_expr <- res_expr_full[[2]]
names(pc_expr)
```

```
##  [1] "All"      "B"        "T"        "Monocyte" "B_c7"     "T_c15"
##  [7] "T_c14"    "T_c10"    "T_c8"     "T_c6"     "T_c5"     "T_c4"
## [13] "T_c3"     "T_c2"     "Mono_c16" "Mono_c13" "Mono_c12" "Mono_c11"
## [19] "Mono_c9"  "Mono_c1"
```

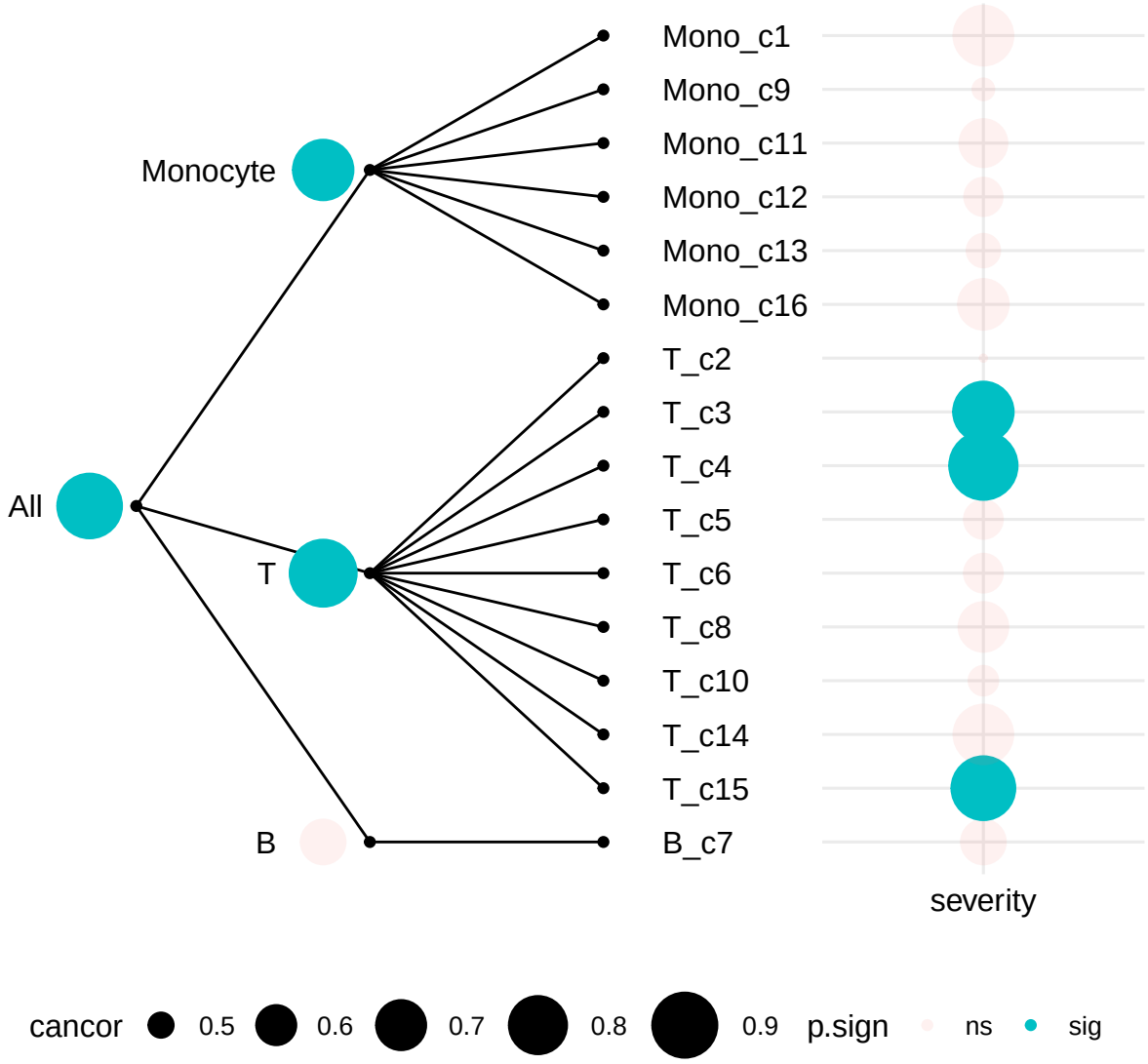
```
# extract PCA matrix corresponding to 'T' tree node
pc_expr[["T"]]
```

```
##                    PC1        PC2
## HD-17-Su    -44.684639   2.823131
## HD-19-Su    -51.774059   4.264000
## HD-23-Su    -51.352355 -23.472410
## HD-30-Su    -39.098933 -37.230307
## Se-137.1-Su  73.681369  47.520847
## Se-178.1-Su  11.966558  20.281313
## Se-180.1-Su   6.457736  53.349695
## Se-181.1-Su  94.804324 -67.536270
```

Similarly, we can use TreeCorTreat plot to visualize the result:

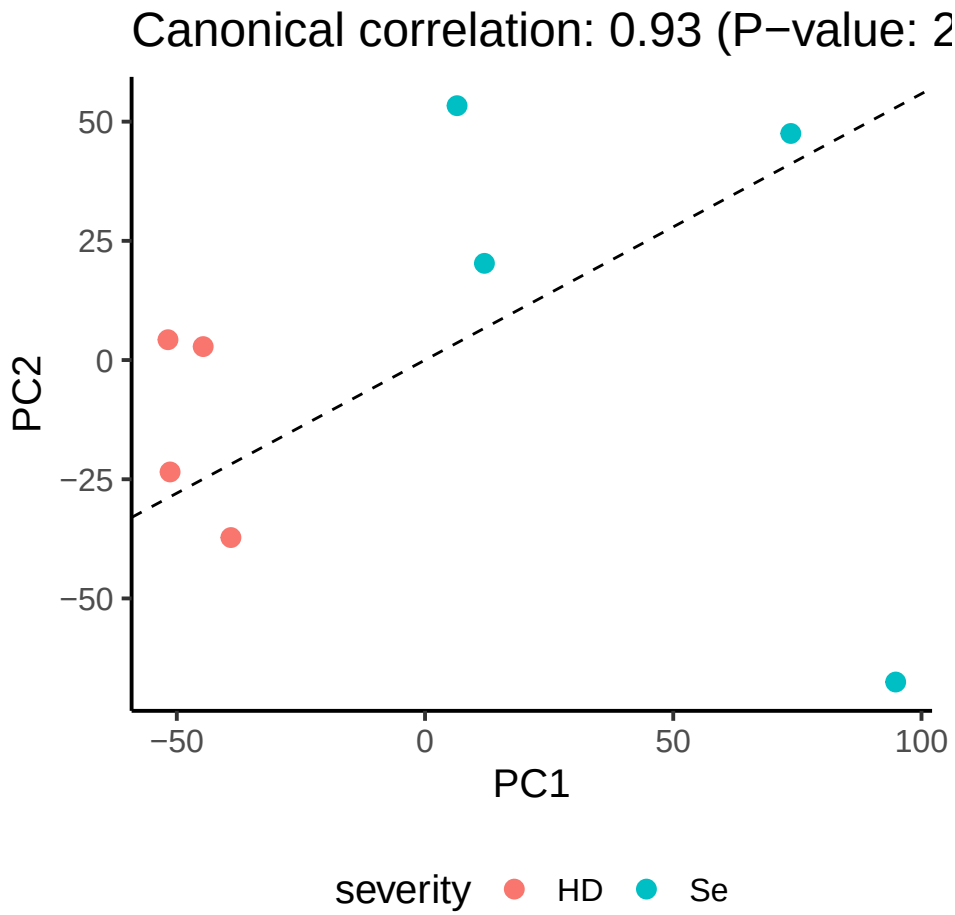
```
# visualize
treecortreatplot(hierarchy_structure, annotated_df = res_expr,
    response_variable = "severity", color_variable = "p.sign",
```

```
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2)
```



In addition to TreeCorTreat plot, users can apply `treecor_samplepcaplot` to visualize the low-dimensional embedding of samples for a given cell type (e.g. T cell category).

```
# PCA plot (using `T` node)
treecor_samplepcaplot(sample_meta, pca_matrix = pc_expr[["T"]],
    response_variable = "severity", point_size = 3)
```



We can observe that samples' severity progressively changes from healthy to severe along the direction indicated by the dashed line, which corresponds to an optimal axis inferred from CCA that maximizes the correlation between the severity and samples' coordinates in the embedded space.

Module 5: Identify differentially expressed genes

The analysis of differentially expressed genes (DEGs) is wrapped into a function `treecor_deg()`. Here we first compute the pseudobulk gene expression profile for each tree node using the 'Aggregate' approach as before. In other words, for each node all cells from its descendants are pooled and aggregated into a pseudobulk profile in each sample. Pseudobulk profiles are normalized by library size. Limma was then used to conduct differential expression analysis:

- Univariate phenotype/Multivariate phenotype analyzed separately: Fit a limma model by regressing gene expression against phenotype with covariate adjustment. DEGs passing a user-specified false discovery rate (FDR) cutoff (default cutoff = 0.05) will be reported for each node and saved into csv files along with log fold change, t-statistics, p-value and FDR.
- Multivariate phenotype analyzed jointly: Multiple traits are combined into a univariate variable using either a data-driven approach (PC1) or user-specified weights (linear combination of multiple traits using weights). Then the aggregated trait will be analyzed as a univariate phnenotype.

```
# DEG pipeline
res_deg <- treecor_deg(count, hierarchy_structure, cell_meta,
    sample_meta, response_variable = "severity")  # can change to TRUE
```

```
names(res_deg)
```

```
## [1] "dge.summary"   "dge.ls"        "pseudobulk.ls"
```

```
# summary of number of DEGs
head(res_deg$dge.summary)
```

```
##      label severity.num_deg     x y id  leaf
## 1      All                0  6.25 2  1 FALSE
## 2        B                0  0.00 1  2 FALSE
## 3        T              993  5.00 1  3 FALSE
## 4 Monocyte                2 12.50 1  4 FALSE
## 5     B_c7                0  0.00 0  5  TRUE
## 6    T_c15              456  1.00 0  6  TRUE
```

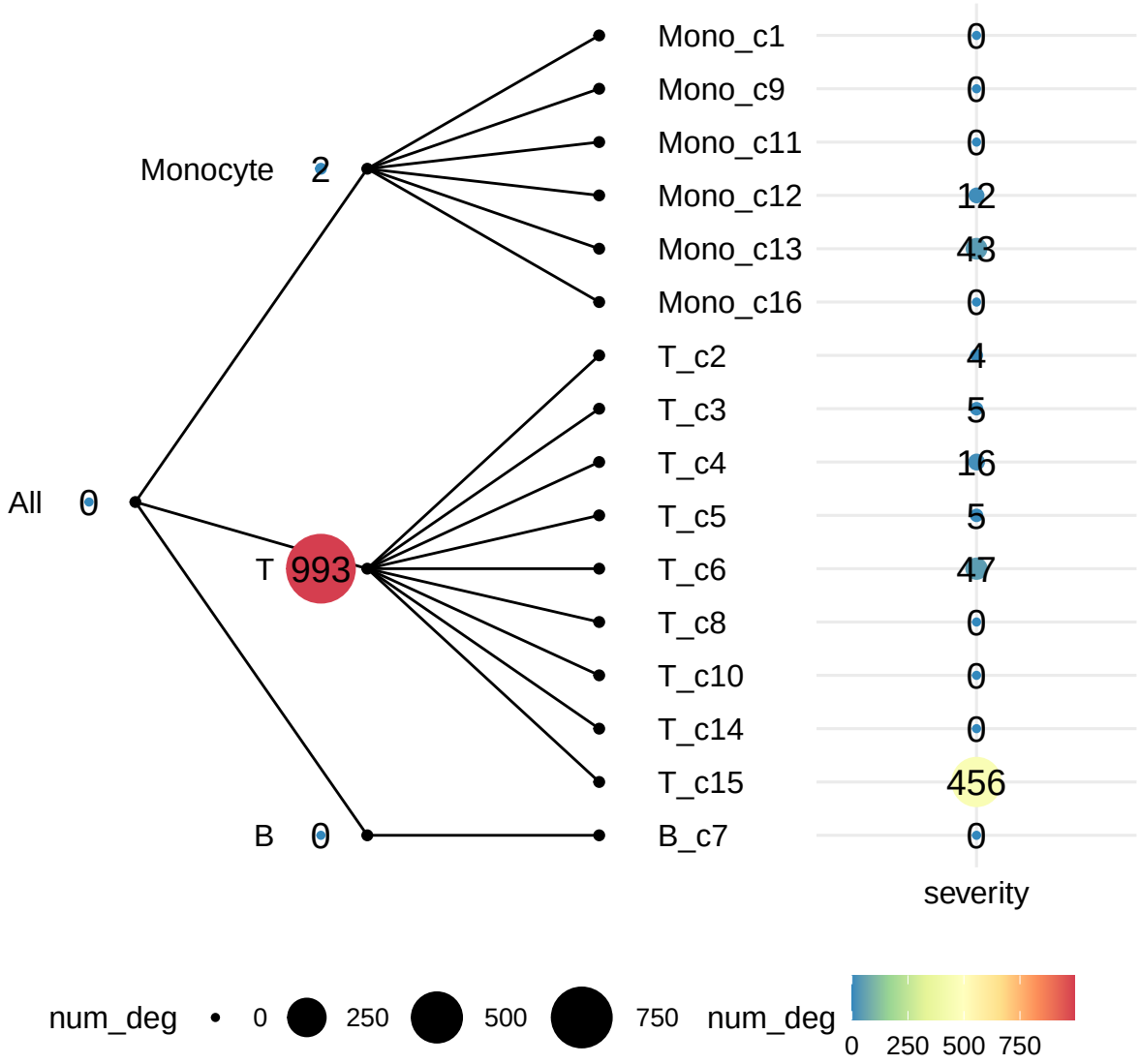
```
# extract DEGs for a specific tree node using cell type
# name
severity.dge.ls <- res_deg$dge.ls[["severity"]]
head(severity.dge.ls[["T"]])
```

```
##          severitySe.logFC severitySe.t severitySe.p severitySe.fdr
## FADS1            2.221299    13.884431 2.351480e-07    0.004165177
## JCHAIN           3.036823    12.451043 5.964215e-07    0.005282207
## INPPL1           1.755148    10.250902 3.058248e-06    0.017470015
## APOBEC3H         2.250345     9.908317 4.052855e-06    0.017470015
## GPR68            1.693644     9.515158 5.657245e-06    0.017470015
## CORO1C           1.813891     9.220954 7.316652e-06    0.017470015
```

```
# extract sample-level pseudobulk for a specific tree node
# using cell type name
head(res_deg$pseudobulk.ls[["T"]])
```

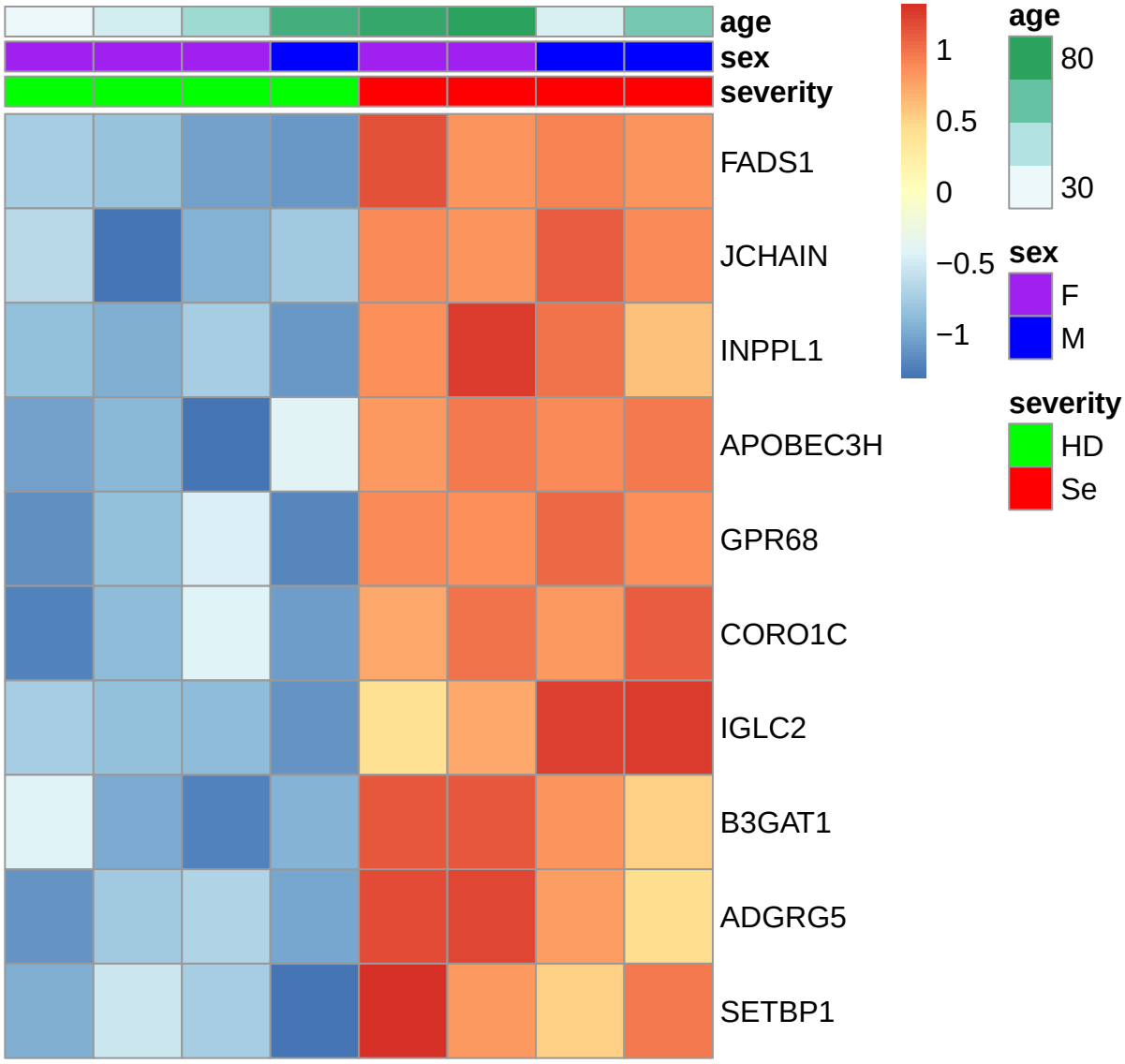
```
##          HD-17-Su HD-19-Su  HD-23-Su HD-30-Su Se-137.1-Su Se-178.1-Su
## A1BG     5.493462 5.506674 5.6802740 5.876471   4.7791551   5.3461643
## A1BG-AS1 2.652148 1.854222 2.3063366 2.420313   2.9822341   2.2278498
## A2M      0.000000 0.000000 0.1968054 0.000000   0.6555113   0.3599841
## A2M-AS1  1.814409 1.413839 1.2995822 1.852688   3.5050187   2.9638223
## A4GALT   0.000000 0.000000 0.3699580 0.000000   0.0000000   0.0000000
## AAAS     4.653329 4.235223 4.1063058 4.665398   5.1026623   4.7283047
##          Se-180.1-Su Se-181.1-Su
## A1BG        5.516765    4.809994
## A1BG-AS1    0.000000    2.002053
## A2M         0.000000    1.001369
## A2M-AS1     3.578594    3.809896
## A4GALT      0.000000    0.000000
## AAAS        4.315328    4.250520
```

```
# visualize
treecortreatplot(hierarchy_structure, annotated_df = res_deg$dge.summary,
    response_variable = "severity", color_variable = "num_deg",
    size_variable = "num_deg", alpha_variable = NULL, annotate_number = T,
    annotate_number_column = "num_deg", font_size = 12, nonleaf_label_pos = 0.4,
    nonleaf_point_gap = 0.2)
```

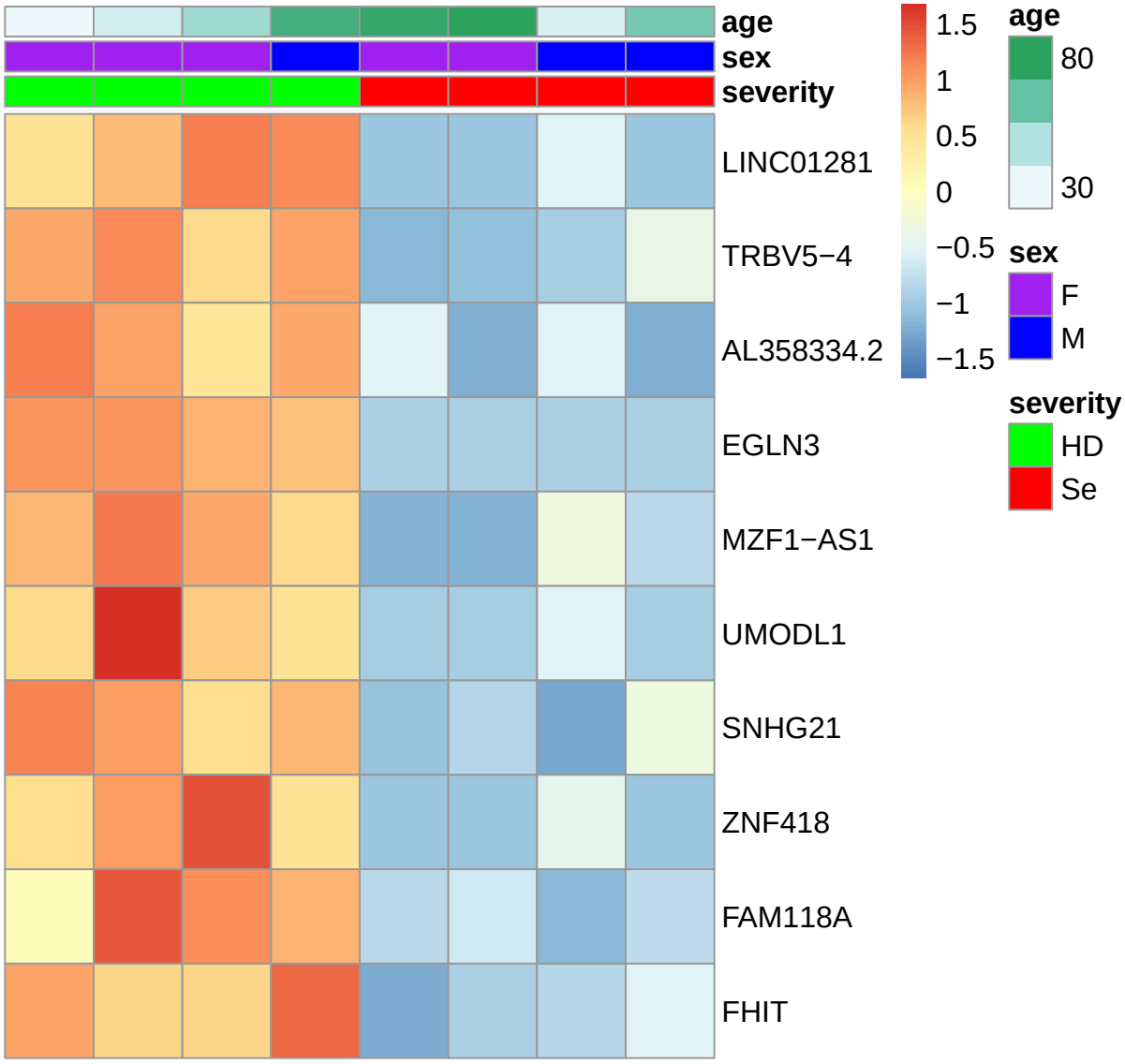


We can then use `treecor_degheatmap` to look at gene expression pattern for top n DEGs:

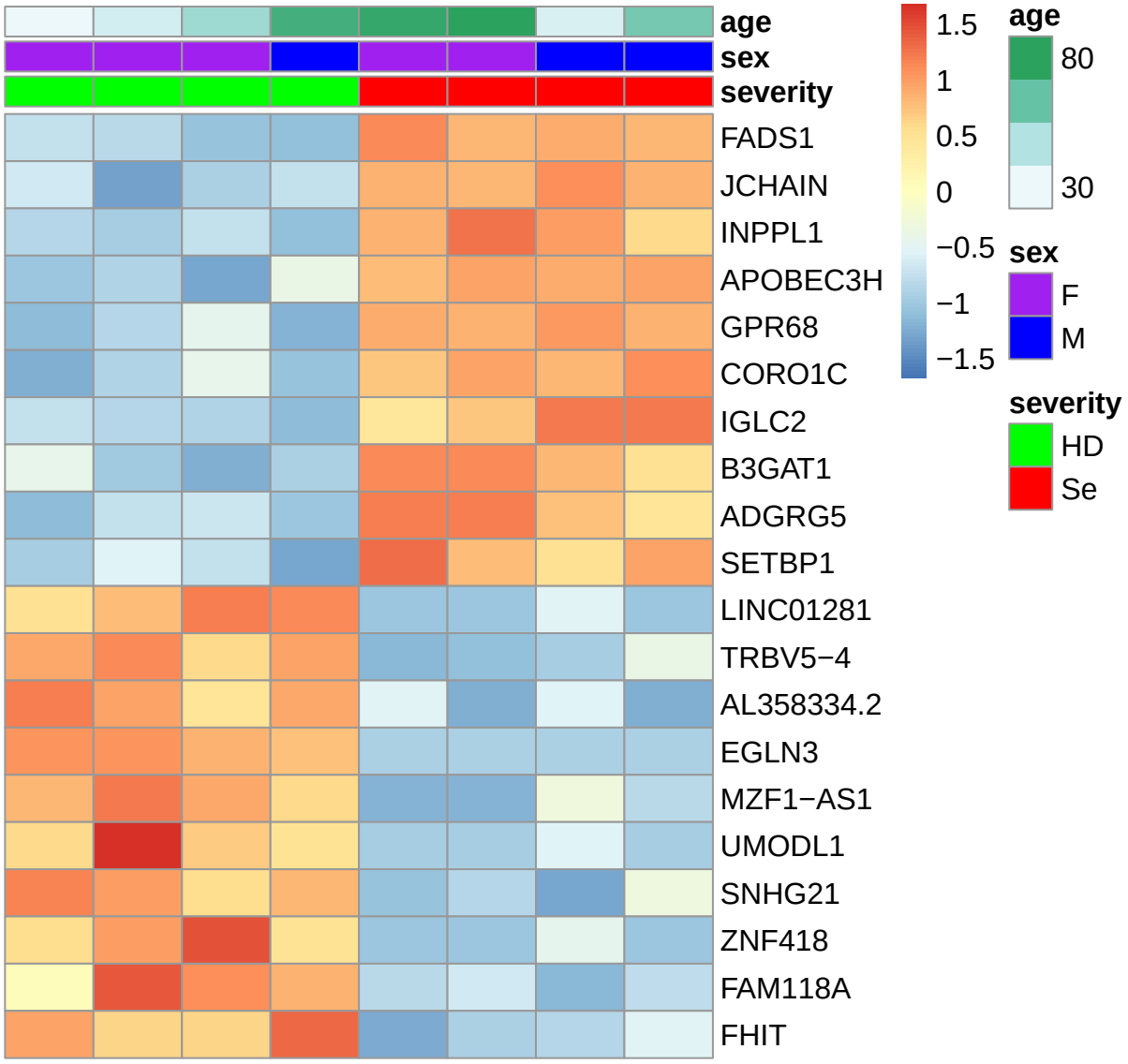
```
# heatmap for top 10 DEGs with positive log fold change
# (enriched in Severe)
treecor_degheatmap(sample_meta, pseudobulk = res_deg$pseudobulk.ls[["T"]],
    deg_result = res_deg$dge.ls$severity[["T"]], top_n = 10,
    deg_logFC = "positive", annotation_col = c("severity", "sex",
        "age"), annotation_colors = list(severity = c(HD = "green",
        Se = "red"), sex = c(F = "purple", M = "blue")))
```




```
# heatmap for top 10 DEGs with negative log fold change
# (enriched in Healthy)
treecor_degheatmap(sample_meta, pseudobulk = res_deg$pseudobulk.ls[["T"]],
    deg_result = res_deg$dge.ls$severity[["T"]], top_n = 10,
    deg_logFC = "negative", annotation_col = c("severity", "sex",
        "age"), annotation_colors = list(severity = c(HD = "green",
        Se = "red"), sex = c(F = "purple", M = "blue")))
```




```
# Combined two plots above
treecor_degheatmap(sample_meta, pseudobulk = res_deg$pseudobulk.ls[["T"]],
    deg_result = res_deg$dge.ls$severity[["T"]], top_n = 10,
    deg_logFC = "both", annotation_col = c("severity", "sex",
        "age"), annotation_colors = list(severity = c(HD = "green",
        Se = "red"), sex = c(F = "purple", M = "blue")))
```



Module 6: Visualization

TreeCorTreat provides a variety of functions to visualize both the intermediate and final results. For example, we can use `treecor_celldensityplot` to show cell type proportions for different sample groups (Module 3). `treecor_samplepcaplot` projects samples to a shared low-dimensional embedding and find optimal axis (Module 4).`treecor_degheatmap` shows top differentially expressed genes of a given cell type (Module 5). In particular, in order to summarize the final results, we developed a versatile function `treecortreatplot` to display the multi-resolution cell type-phenotype association in a format we call **'TreeCorTreat plot'**. The plot consists of a dendrogram showing the cell types' hierarchical structure and information layers showing analysis results about each tree node.

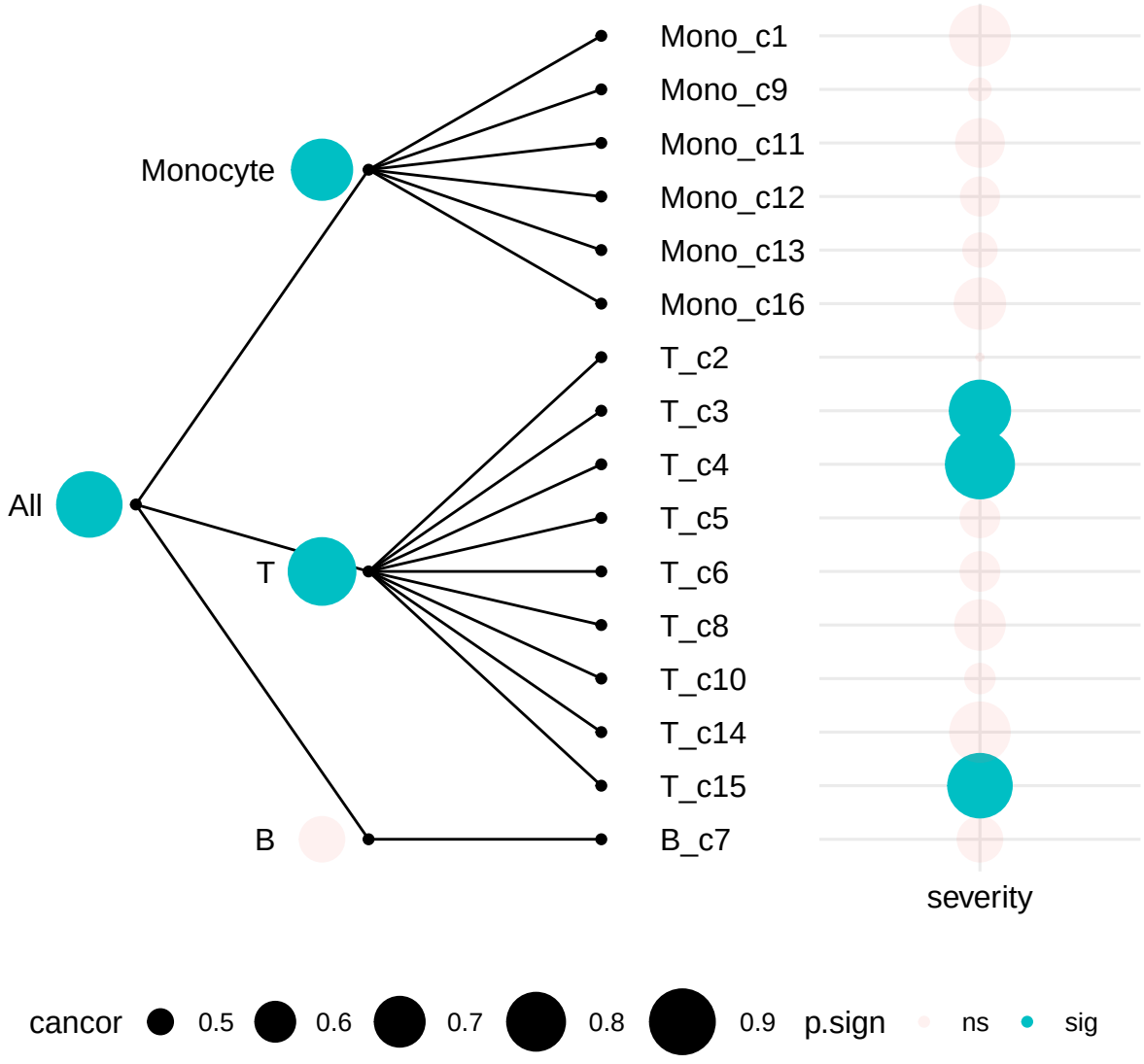
Tree skeleton representation

For displaying the tree, the nodes can be connected using straight lines, a classical angle bend representation, or a quadratic bezier curve representation. Users can choose line aesthetics such as linetype (e.g. solid or dashed) and line color according to their preference.

Edge type

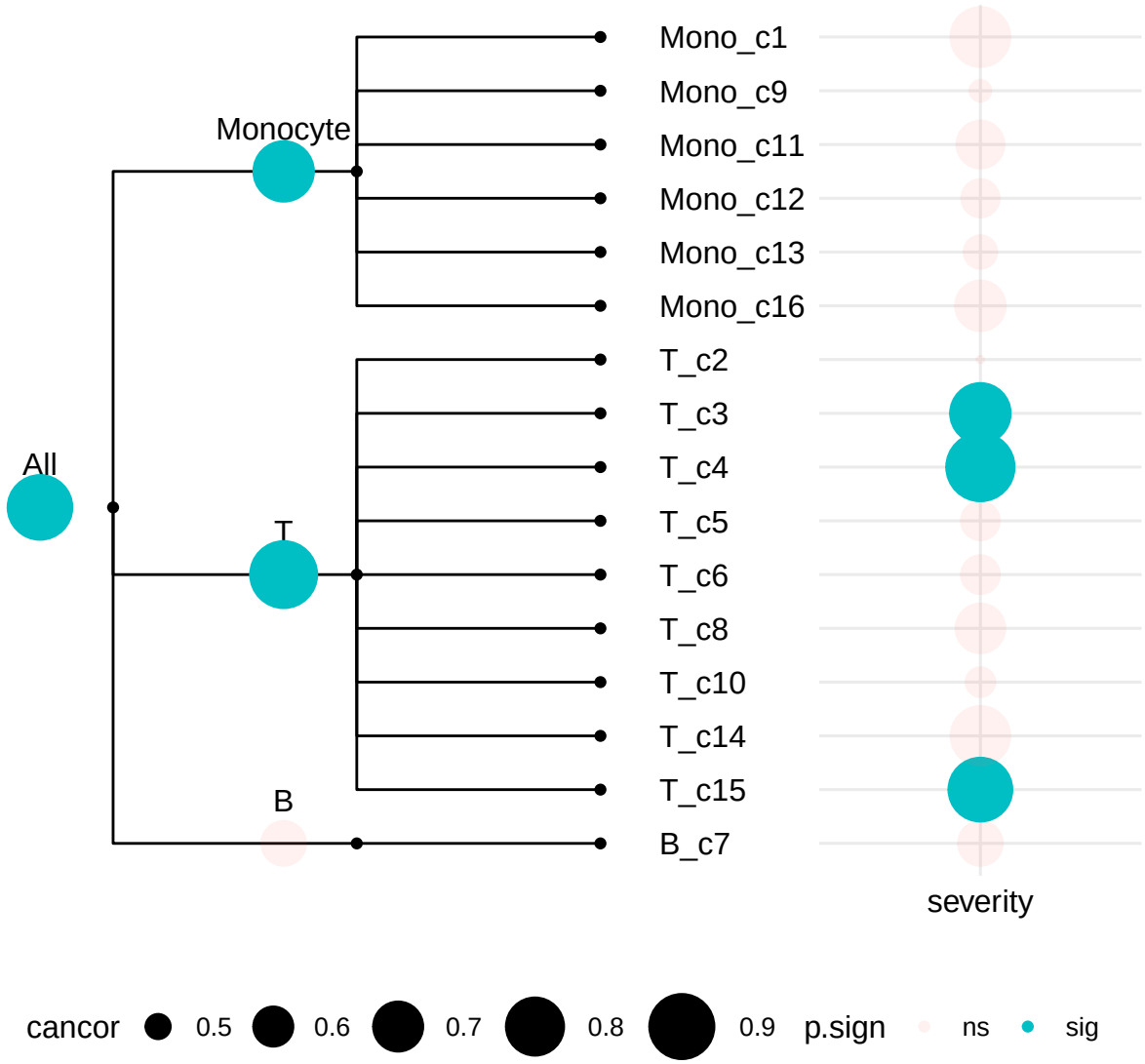
- Straight line

```
treecortreatplot(hierarchy_structure,
                 annotated_df = res_expr,
                 response_variable = 'severity',
                 color_variable = 'p.sign',
                 size_variable = 'cancor',
                 alpha_variable = 'p.sign',
                 font_size = 12,
                 nonleaf_label_pos = 0.4,
                 nonleaf_point_gap = 0.2,
                 ## parameter: edge_path_type
                 edge_path_type = "link")
```



- Classical angle bend

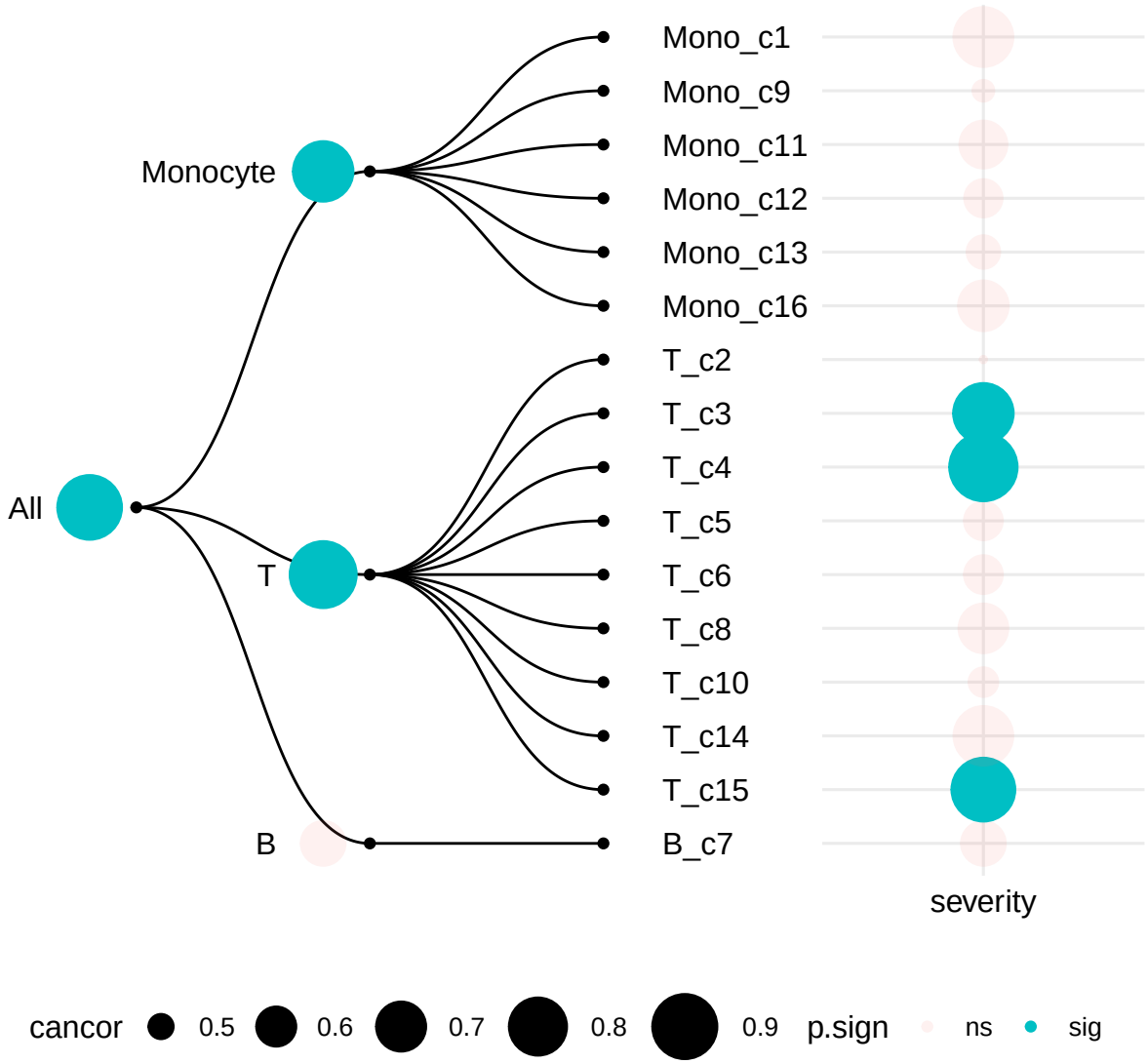
```
treecortreatplot(hierarchy_structure,
                 annotated_df = res_expr,
                 response_variable = 'severity',
                 color_variable = 'p.sign',
                 size_variable = 'cancor',
                 alpha_variable = 'p.sign',
                 font_size = 12,
                 nonleaf_label_pos = 0.6,
                 nonleaf_point_gap = 0.3,
                 ## parameter: edge_path_type
                 edge_path_type = "elbow")
```



Based on our experience, we recommend to use classical angle bend in a data-driven approach, avoiding intertwined lines.

(3) Quadratic Bezier curves

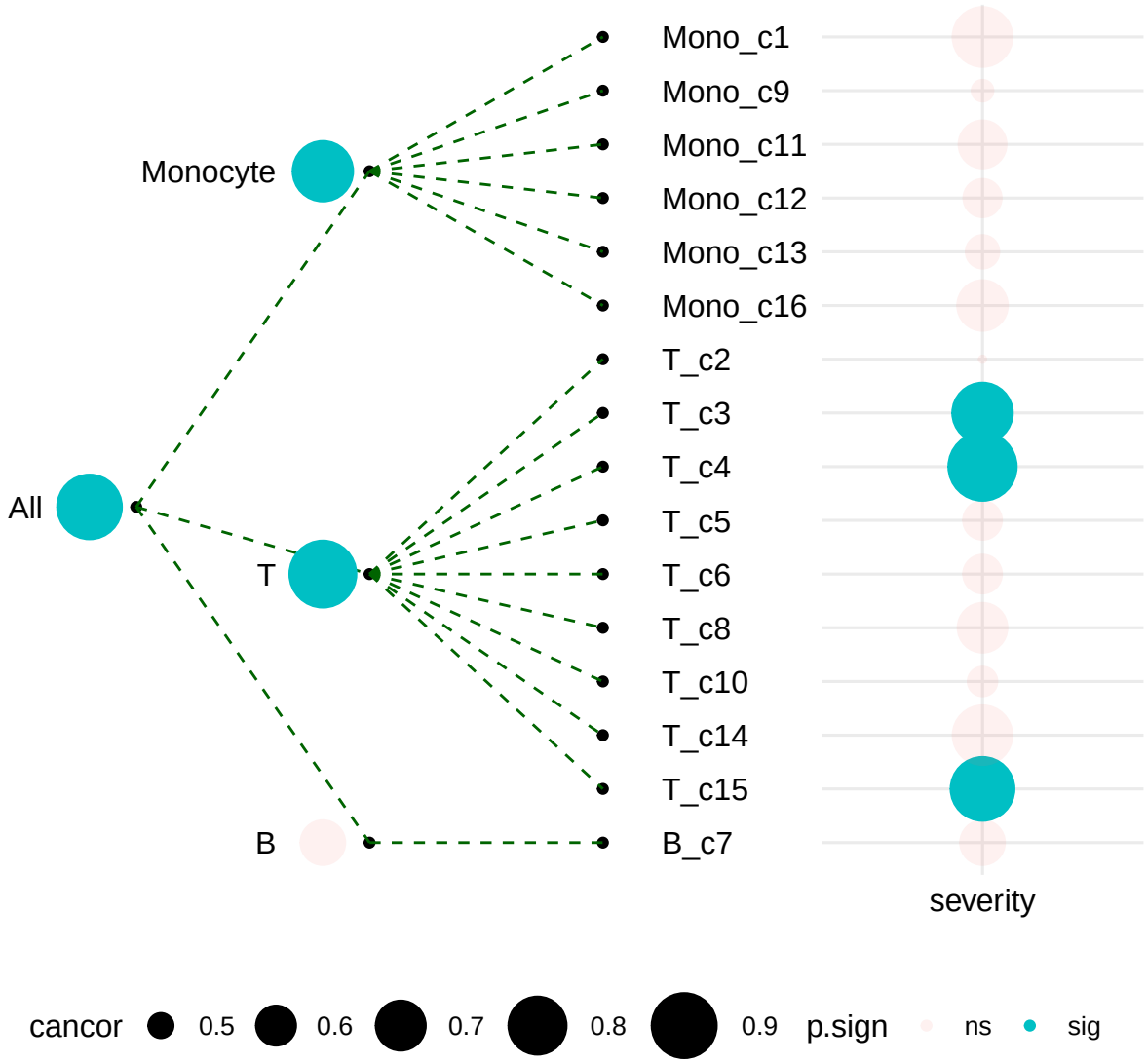
```
treecortreatplot(hierarchy_structure,
                 annotated_df = res_expr,
                 response_variable = 'severity',
                 color_variable = 'p.sign',
                 size_variable = 'cancor',
                 alpha_variable = 'p.sign',
                 font_size = 12,
                 nonleaf_label_pos = 0.4,
                 nonleaf_point_gap = 0.2,
                 ## parameter: edge_path_type
                 edge_path_type = "diagonal")
```



Line aesthetics: line type and color

```
treecortreatplot(hierarchy_structure,
                 annotated_df = res_expr,
```

```
                response_variable = 'severity',
                color_variable = 'p.sign',
                size_variable = 'cancor',
                alpha_variable = 'p.sign',
                font_size = 12,
                nonleaf_label_pos = 0.4,
                nonleaf_point_gap = 0.2,
                ## parameter: line_type
                line_type = 'dashed',
                ## parameter: line_color
                line_color = 'darkgreen')
```



Modify label or circle position

In some cases where label length (cell type name) is long or circle size is quite large, default label or circle position on the non-leaf part (left) may not work well. Therefore, we introduce `nonleaf_label_pos` and `nonleaf_point_gap` parameters which allow users to manually fix the label position or gap between

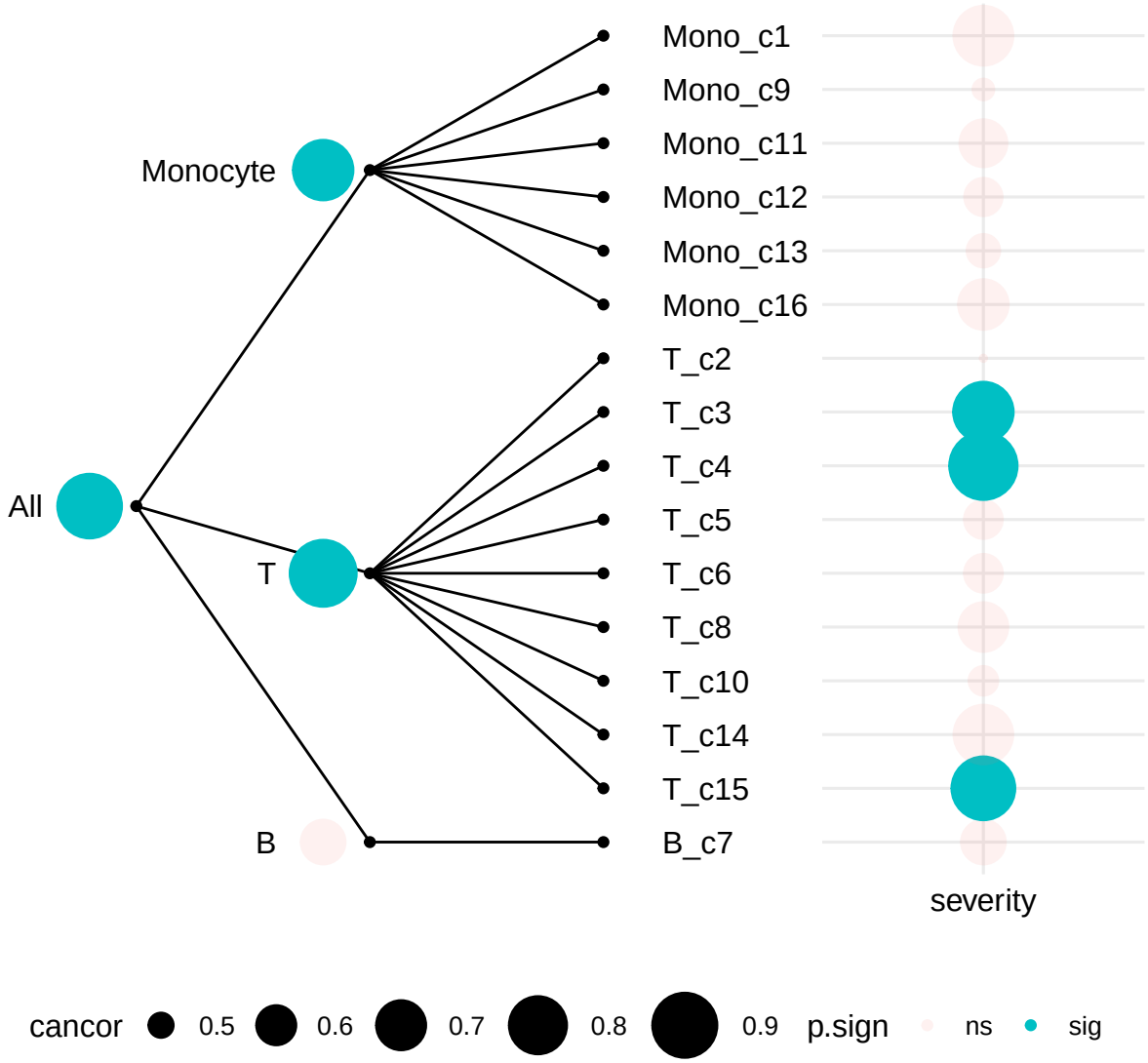
multivariate phenotype. A tip is that if one have k phenotypes to be analyzed separately, one can first choose a proper gap between points (i.e. `nonleaf_point_gap = 0.1`) and let `nonleaf_label_pos = $k*nonleaf_point_gap + small_number$` (i.e. let $k = 3$, `nonleaf_label_pos = 0.1*3+0.2 = 0.5`).In the above examples, we specify gap between points at 0.2 and label position at 0.4.

Leaf representation

For external node configuration, TreeCorTreat plot aligns cell types in the rows and phenotypes in the columns. There are three different ways to visualize the results: balloon plot, heatmap and barplot. Users can use color (`color_variable`), size (`size_variable`) and transparency (`alpha_variable`) to encode different information.

Balloon plot

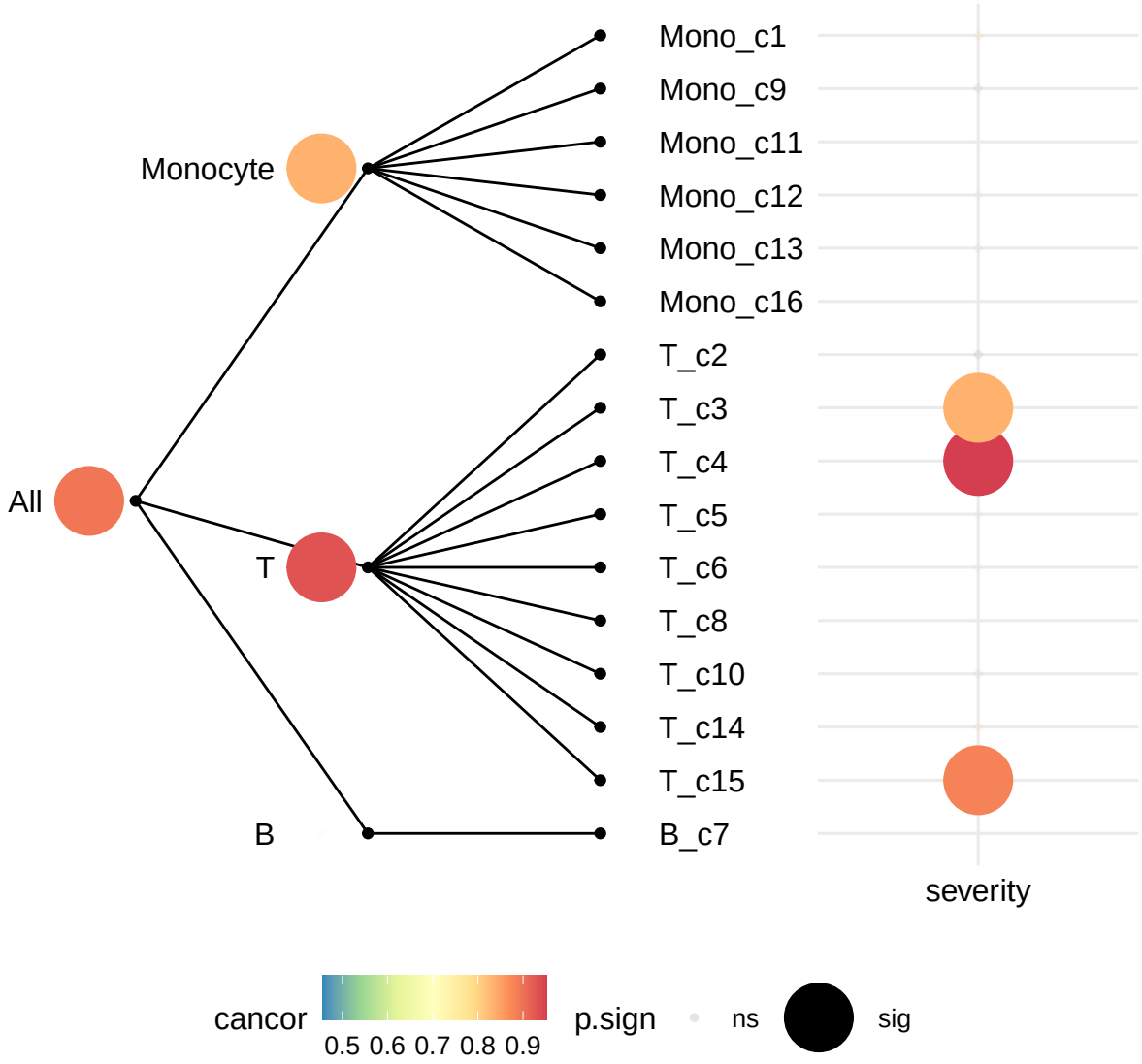
```
treecortreatplot(hierarchy_structure,
                 annotated_df = res_expr,
                 response_variable = 'severity',
                 color_variable = 'p.sign',
                 size_variable = 'cancor',
                 alpha_variable = 'p.sign',
                 font_size = 12,
                 nonleaf_label_pos = 0.4,
                 nonleaf_point_gap = 0.2,
                 ## parameter: plot_type
                 plot_type = 'balloon')
```



In the above balloon plot, the size of the balloon reflects the magnitude of canonical correlation between global gene expression profile and disease severity and color as well as transparency correspond to p-value significance.

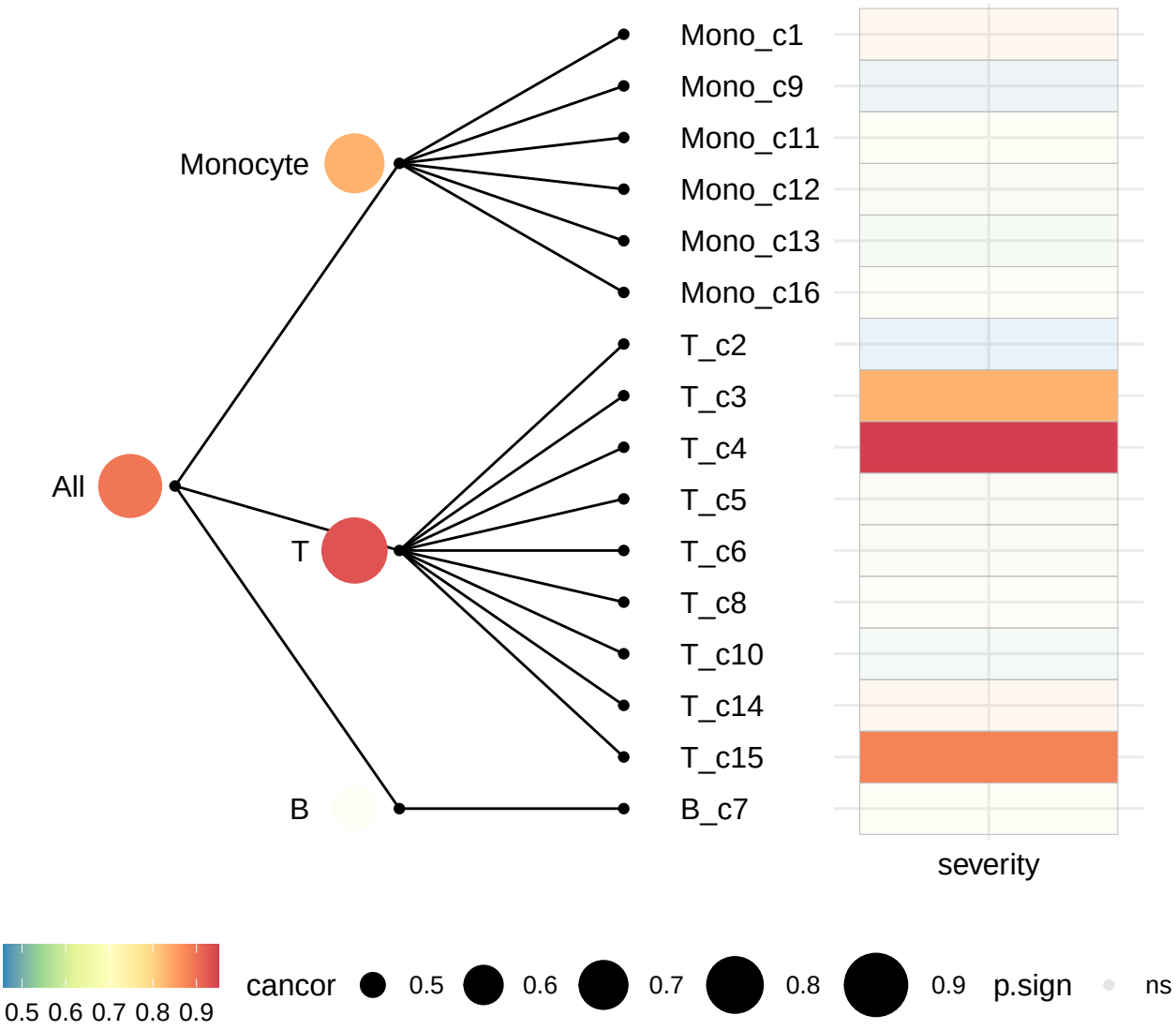
Alternatively, we can use color to represent canonical correlation and size/transparency to represent p-value significance using the following code:

```
treecortreatplot(hierarchy_structure,
                 annotated_df = res_expr,
                 response_variable = 'severity',
                 color_variable = 'cancor',
                 size_variable = 'p.sign',
                 alpha_variable = 'p.sign',
                 font_size = 12,
                 nonleaf_label_pos = 0.4,
                 nonleaf_point_gap = 0.2,
                 ## parameter: plot_type
                 plot_type = 'balloon')
```



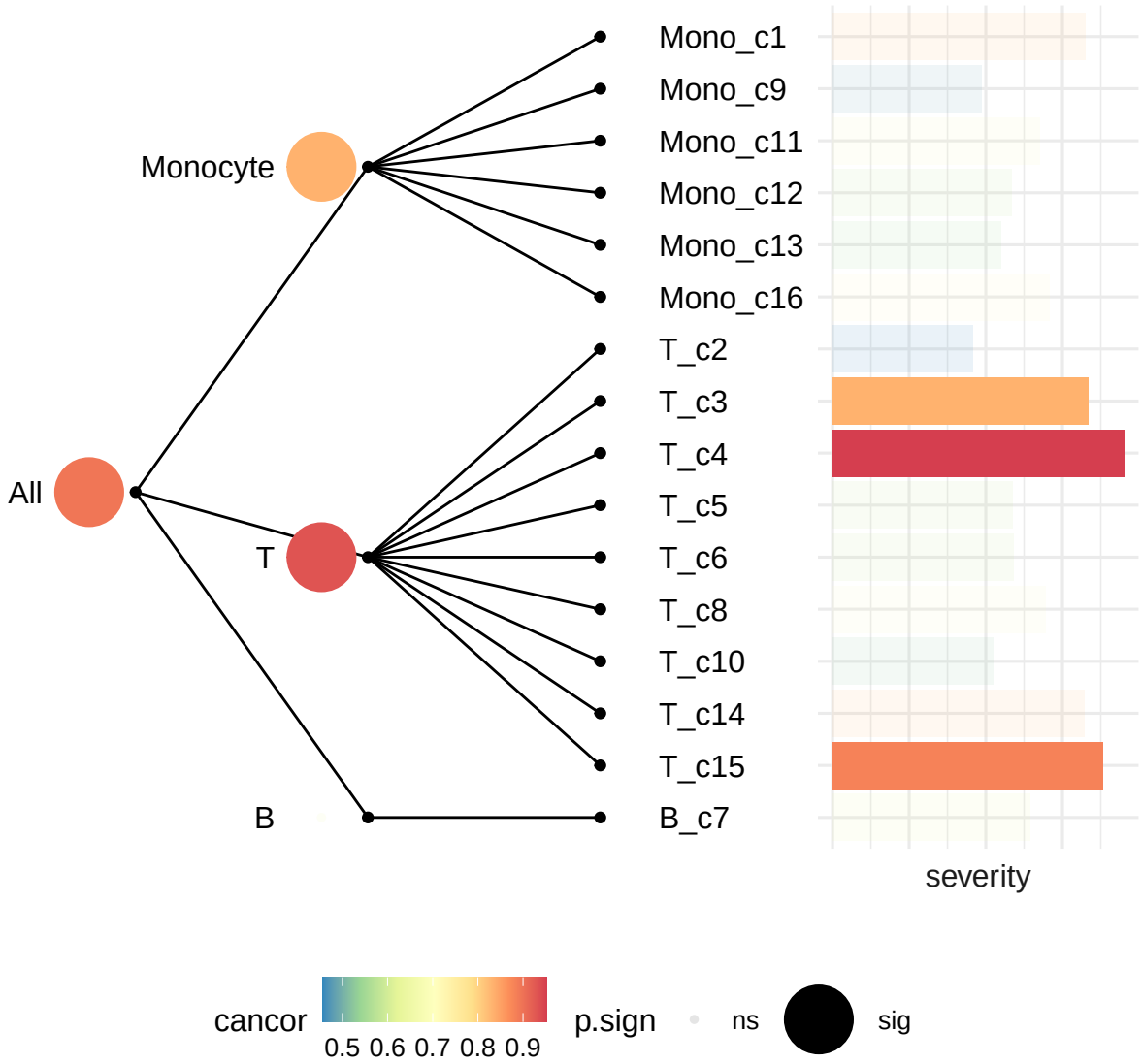
Heatmap

```
treecortreatplot(hierarchy_structure,
                 annotated_df = res_expr,
                 response_variable = 'severity',
                 color_variable = 'cancor',
                 size_variable = 'cancor',
                 alpha_variable = 'p.sign',
                 font_size = 12,
                 nonleaf_label_pos = 0.4,
                 nonleaf_point_gap = 0.2,
                 ## parameter: plot_type
                 plot_type = 'heatmap')
```



Barplot

```
treecortreatplot(hierarchy_structure,
                 annotated_df = res_expr,
                 response_variable = 'severity',
                 color_variable = 'cancor',
                 size_variable = 'p.sign',
                 alpha_variable = 'p.sign',
                 font_size = 12,
                 nonleaf_label_pos = 0.4,
                 nonleaf_point_gap = 0.2,
                 ## parameter: plot_type
                 plot_type = 'bar')
```

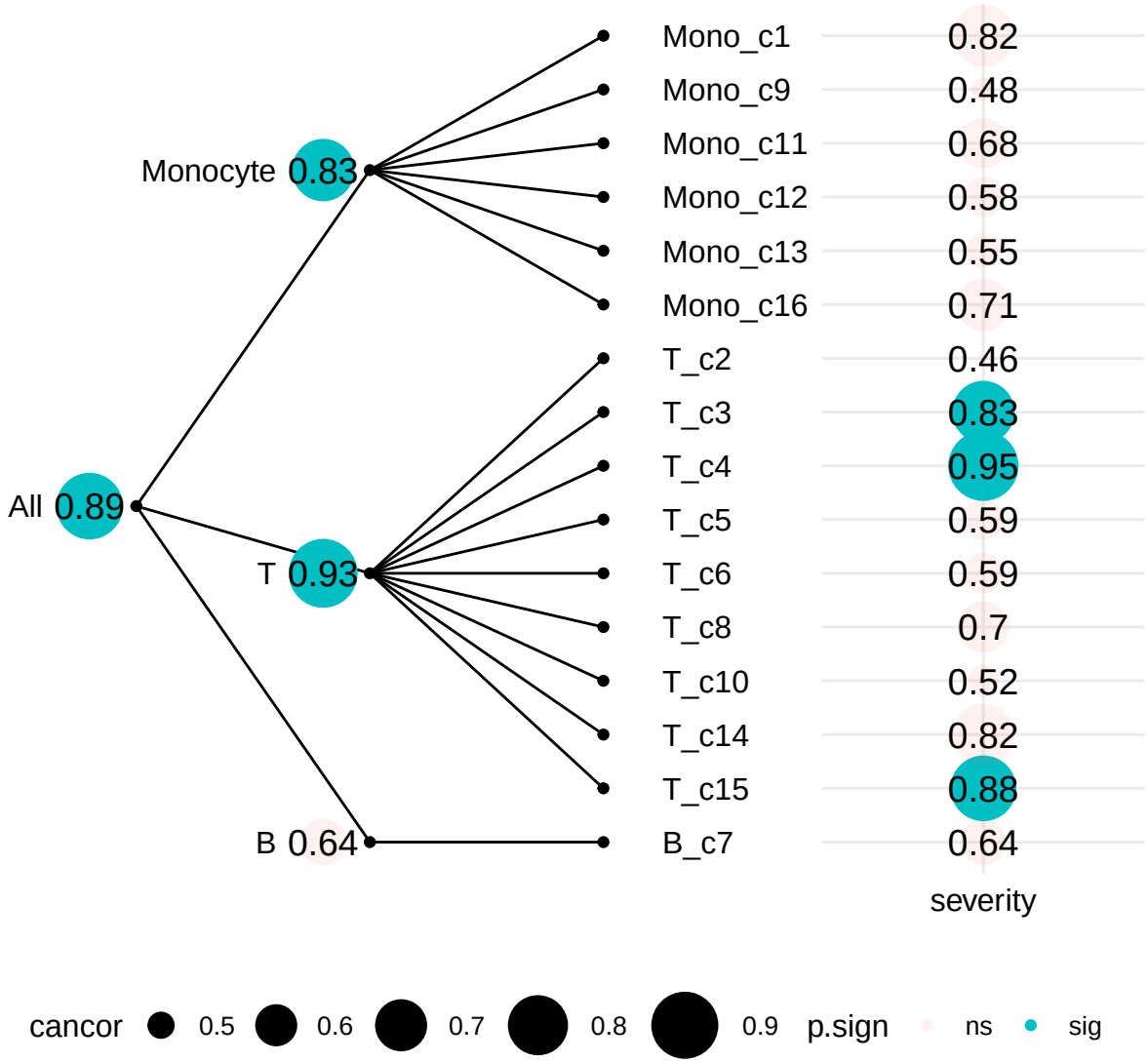


More advanced plotting figures

Annotate numbers

Users can specify `annotate_number`, `annotate_number_column` and `annotate_number_color` to specify which column/color to be used in annotating the TreeCorTreat plot.

```
treecortreatplot(hierarchy_structure, annotated_df = res_expr,
    response_variable = "severity", color_variable = "p.sign",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2, plot_type = "balloon",
    annotate_number = T, annotate_number_column = "cancor", annotate_number_color = "black")
```



Modify legend title

If we want to modify legend title, one way is to change column names of `annotated_df`. The column names for summary statistic and statistical significance are usually defined in `response_variable.colname` format (e.g. severity.cancor, severity.p, etc). Thus, it's crucial to check the column names are in the correct format before generating TreeCorTreat plot.

Suppose we hope to modify names for both response variables ('severity') and legends (i.e. 'cancor' and 'p.sign'): capitalize the first letter in 'severity' and 'cancor' and replace 'p.sign' by 'P-value significance'.

```
## modify column names
colnames(res_expr)
```

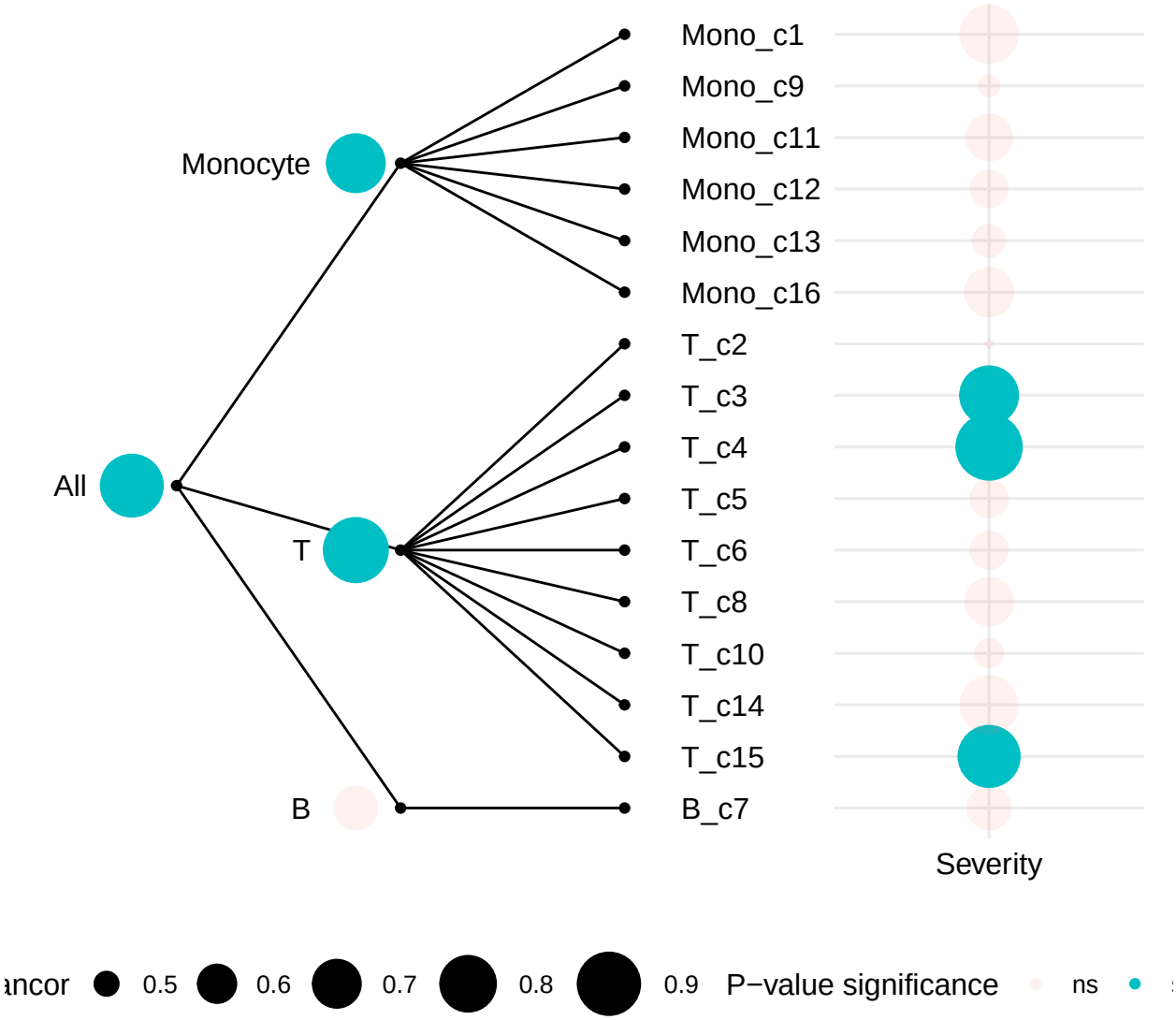
```
##  [1] "id"                "severity.cancor"    "severity.p"
##  [4] "severity.adjp"     "severity.direction" "severity.p.sign"
##  [7] "severity.adjp.sign" "x"                  "y"
## [10] "label"             "leaf"
```

```
res_expr_new <- res_expr
colnames(res_expr_new) <- gsub("severity", "Severity", colnames(res_expr_new))
colnames(res_expr_new) <- gsub("cancor", "Cancor", colnames(res_expr_new))
colnames(res_expr_new) <- gsub("\\.p.sign", "\\.P-value significance",
    colnames(res_expr_new))

## check modified column names
colnames(res_expr_new)
```

```
##  [1] "id"                            "Severity.Cancor"
##  [3] "Severity.p"                    "Severity.adjp"
##  [5] "Severity.direction"            "Severity.P-value significance"
##  [7] "Severity.adjp.sign"            "x"
##  [9] "y"                             "label"
## [11] "leaf"
```

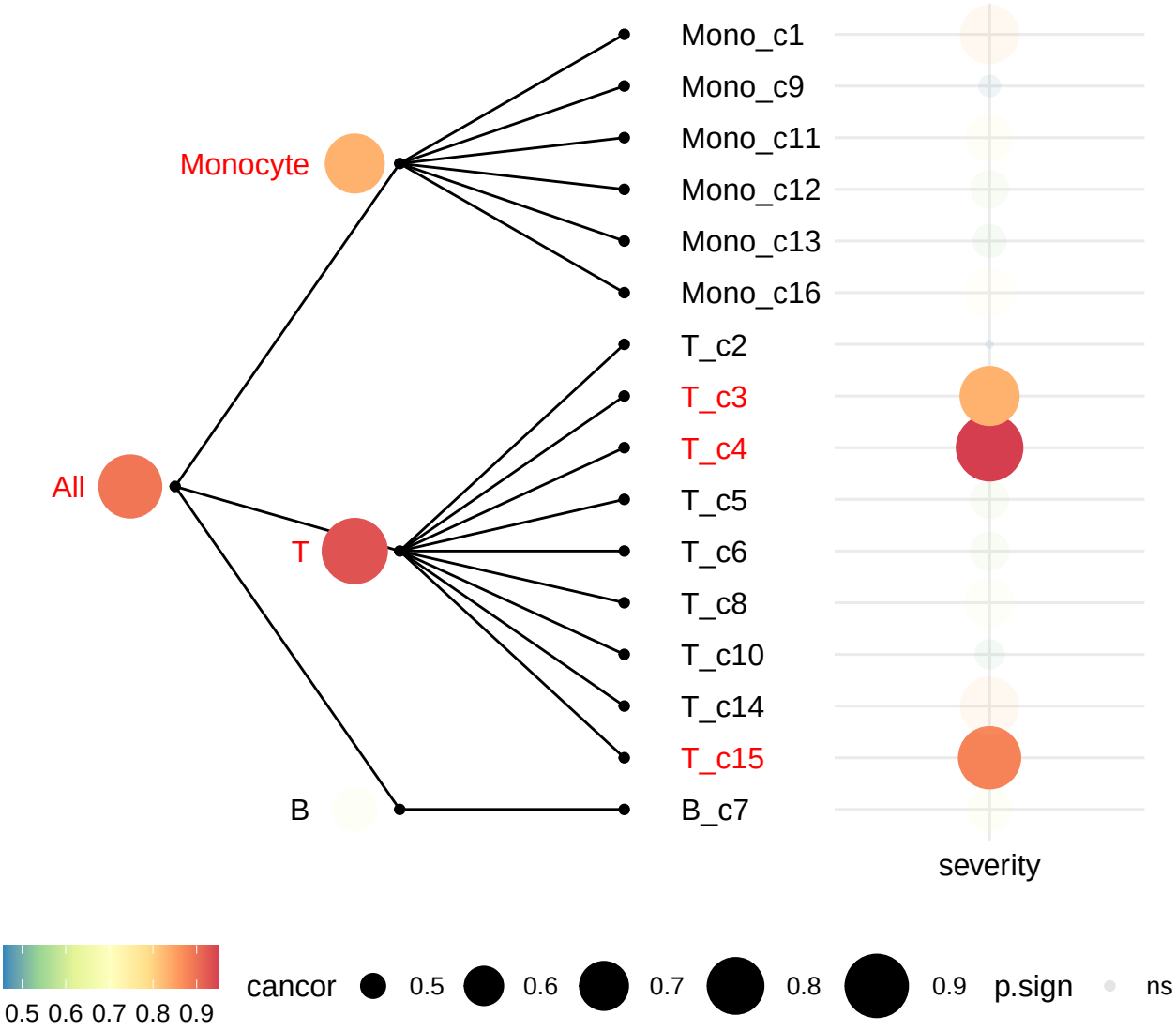
```
treecortreatplot(hierarchy_structure, annotated_df = res_expr_new,
    response_variable = "Severity", color_variable = "P-value significance",
    size_variable = "Cancor", alpha_variable = "P-value significance",
    font_size = 12, nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2,
    plot_type = "balloon")
```



Modify label colors

Users can modify different label colors to highlight cell types of interest. For example, we can highlight cell types that have significant global gene expression-disease severity correlation in red:

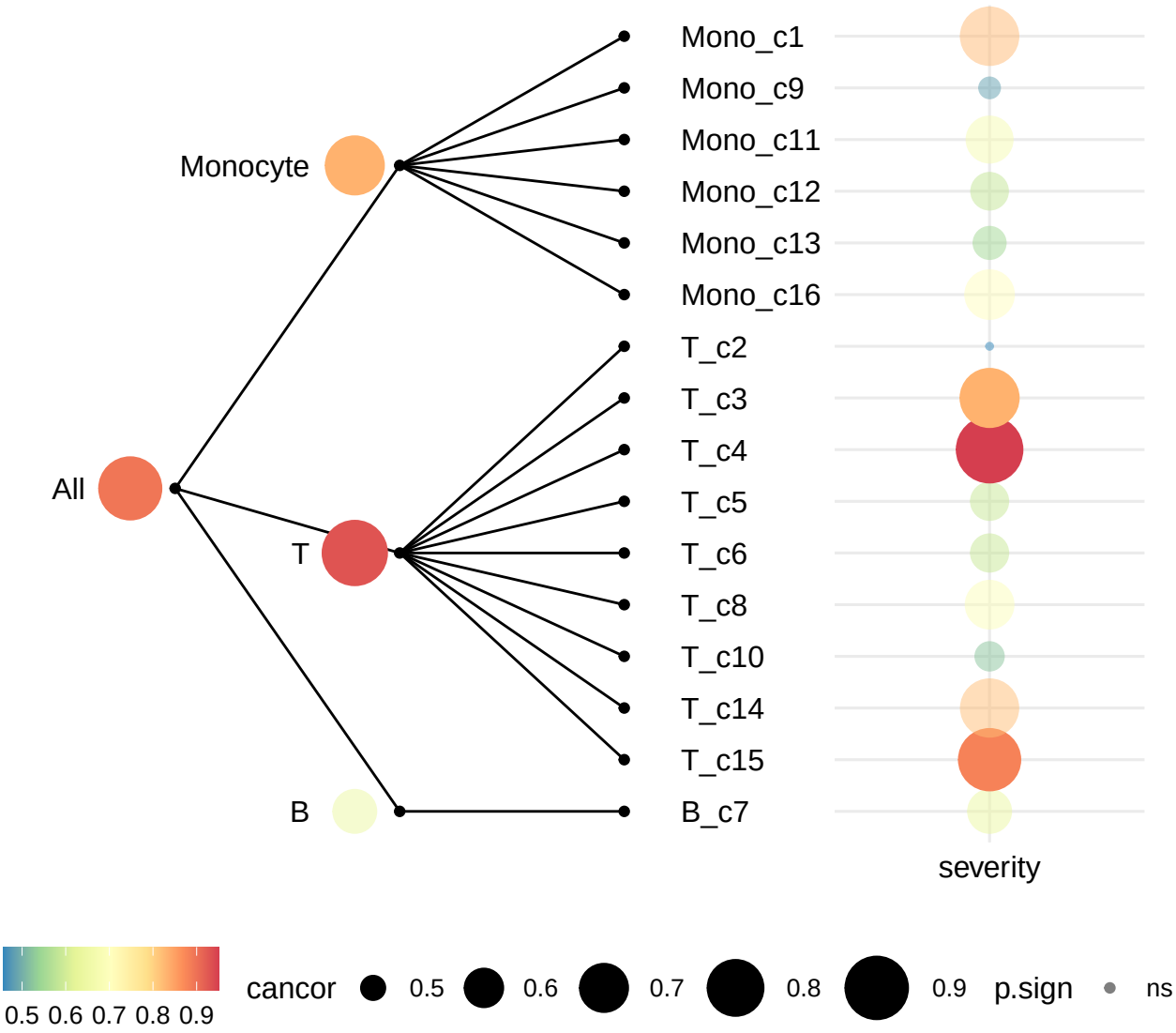
```
label_info <- data.frame(label = res_expr$label, label.color = ifelse(res_expr$severity.p.sign ==
    "ns", "black", "red"))
treecortreatplot(hierarchy_structure, annotated_df = res_expr,
    response_variable = "severity", color_variable = "cancor",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2, plot_type = "balloon",
    advanced_list = list(label_info = label_info))
```



Create your own mapping from levels to aesthetic values

Users can customize mapping from levels in the data (e.g. categorical variables) to certain aesthetic values (e.g. color/size/transparency) via `breaks` and `values` (alike `scale_xx_manual()` in `ggplot2`). The following code illustrates an example of changing color of statistical significance as blue (non-significant) and red (significant) and modifying transparency:

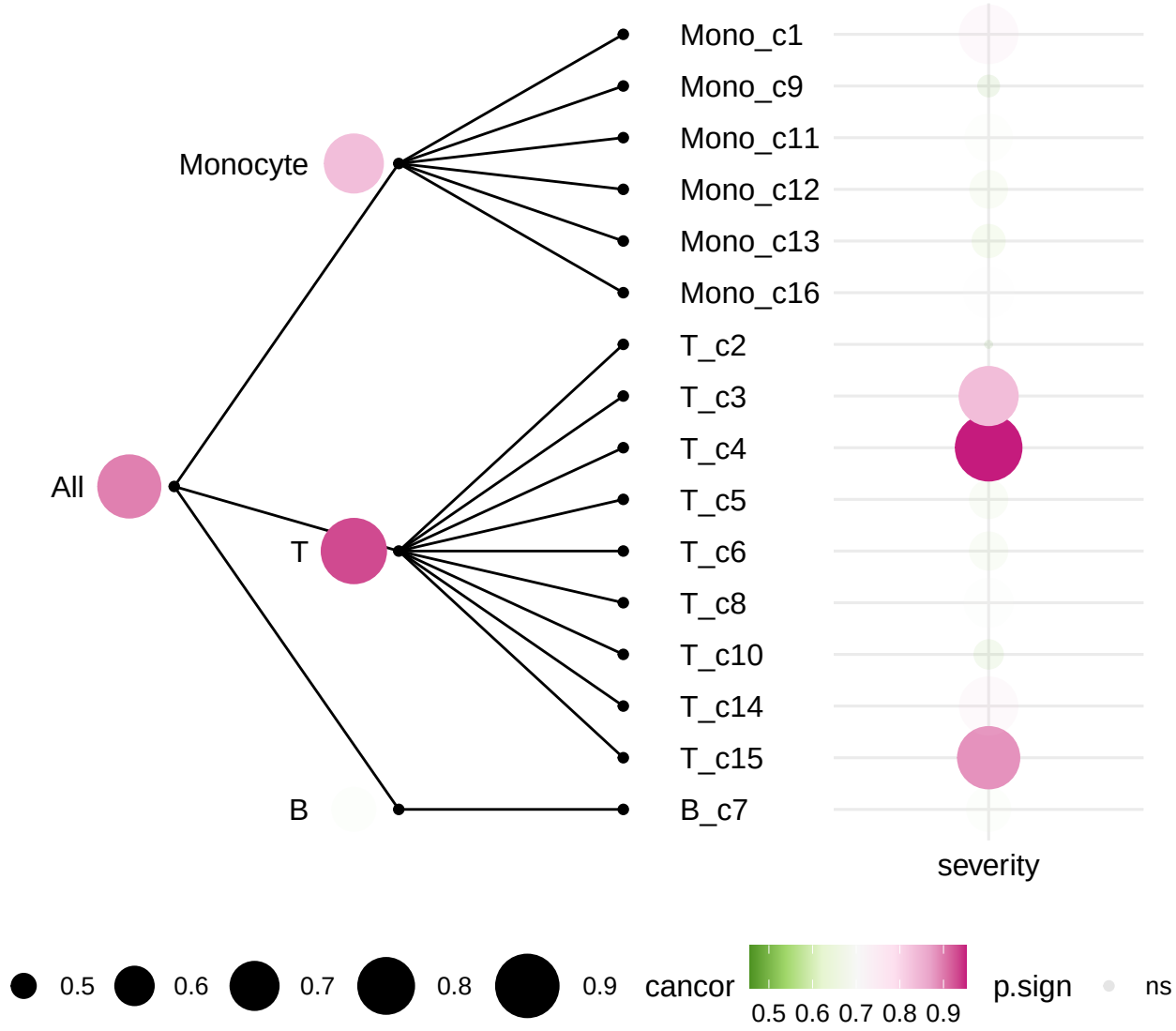
```
advanced_list <- list(color_info = data.frame(breaks = c("ns",
    "sig"), values = c("blue", "red")), alpha_info = data.frame(breaks = c("ns",
    "sig"), values = c(0.5, 1)))
treecortreatplot(hierarchy_structure, annotated_df = res_expr,
    response_variable = "severity", color_variable = "cancor",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2, plot_type = "balloon",
    advanced_list = advanced_list)
```



Choose a different color palette

For categorical variable, color palette can be specified by `color_info` in advanced options; For continuous variable, we provide different palettes from `RColorBrewer` package (default is 'Spectral') and can be modified via `palette` as a element in the `advanced_list`.

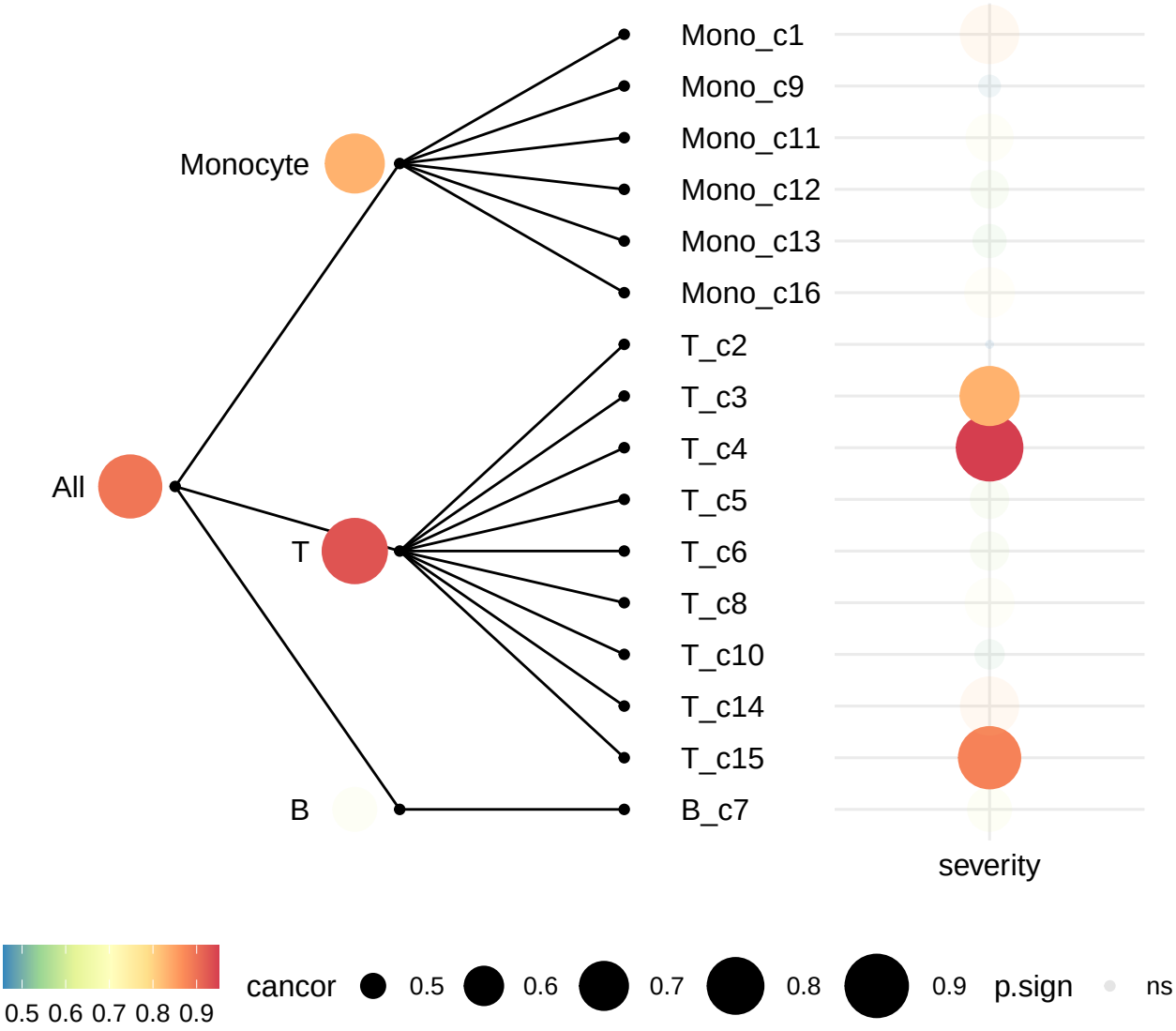
```
treecortreatplot(hierarchy_structure, annotated_df = res_expr,
    response_variable = "severity", color_variable = "cancor",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2, plot_type = "balloon",
    advanced_list = list(palette = "PiYG"))
```



Configure layout

The layout of TreeCorTreat plot can be modified via `layout_widths` in the `advanced_list` parameter, which specifies the relative left-to-right ratio (or non-leaf: leaf ratio).

```
treecortreatplot(hierarchy_structure, annotated_df = res_expr,
    response_variable = "severity", color_variable = "cancor",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2, plot_type = "balloon",
    advanced_list = list(layout_widths = c(2, 1)))
```



Save TreeCorTreat plots

TreecorTreat plots can be saved into pdf, png or other graphical formats.

```
png("TreeCorTreat_plot.png", height = 8, width = 15, res = 300,
    units = "in")
treecortreatplot(hierarchy_structure, annotated_df = res_expr,
    response_variable = "severity", color_variable = "cancor",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2, plot_type = "balloon")
dev.off()
```

TreeCorTreat supports versatile multi-resolution analysis

To analyze cell type-phenotype association at multiple resolutions, fine-grained cell clusters (children) are progressively merged into bigger clusters (parents). When a parent node is created by merging two or more child nodes, the information from the children can be combined in multiple different ways with different

implications. The two basic methods to combine child nodes are 'aggregation' and 'concatenation'. The 'aggregation' method pools all cells from the child nodes into a single pseudobulk sample which is used to represent the parent node. In the 'concatenation' method, one first pools cells within each child node to form a pseudobulk sample for that node. One represents each child node using a feature or a feature vector extracted from its pseudobulk sample (e.g., highly variable genes). The parent node is then represented by concatenating the feature vectors from all child nodes into a longer vector.

To support different needs of users, TreeCorTreat provides three different ways to combine child nodes: 'aggregate' (default), 'concatenate leaf nodes', and 'concatenate immediate children'.

Aggregate

The default setting for both `treecor_expr` and `treecor_ctprop` is aggregation (see previous examples), where we aggregate raw read counts for gene expression or number of cells for cell composition for non-leaf nodes.

Concatenate leaf nodes

In the 'concatenate leaf nodes' approach, feature vectors of all terminal leaf nodes derived from a node are concatenated into a long vector to serve as the feature vector of the node. For global gene expression analysis, this will result in a concatenated vector consisting of pseudobulk expression of highly variable genes obtained from each leaf node. For cell type proportion analysis, this will result in a vector of cell type proportions of the leaf nodes.

```
# gene expression pipeline; concat_leaf
res_expr_concatLeaf <- treecor_expr(count, hierarchy_structure,
    cell_meta, sample_meta, response_variable = "severity", method = "concat_leaf",
    num_permutations = 100)$canonical_corr
head(res_expr_concatLeaf)
```

```
##   id severity.cancor severity.p severity.adjp severity.direction
## 1  1       0.7803956 0.03030303     0.4360897                 +
## 2  2       0.6428883 0.15151515     0.9911129                 +
## 3  3       0.8556124 0.03030303     0.4360897                 +
## 4  4       0.6743079 0.03030303     0.4360897                 +
## 5  5       0.6428883 0.15151515     0.9911129                 +
## 6  6       0.8807827 0.04040404     0.4845441                 +
##   severity.p.sign severity.adjp.sign     x y    label  leaf
## 1             sig                 ns  6.25 2      All FALSE
## 2              ns                 ns  0.00 1        B FALSE
## 3             sig                 ns  5.00 1        T FALSE
## 4             sig                 ns 12.50 1 Monocyte FALSE
## 5              ns                 ns  0.00 0     B_c7  TRUE
## 6             sig                 ns  1.00 0    T_c15  TRUE
```

Concatenate immediate children

In the 'concatenate immediate children' approach, a node's immediate children are first identified. The feature vector of each immediate child is obtained using the 'aggregate' approach. Then feature vector of target node is obtained by concatenating the feature vectors of its immediate children.

```
# gene expression pipeline; concat_immediate_children
res_expr_concatImmChild <- treecor_expr(count, hierarchy_structure,
    cell_meta, sample_meta, response_variable = "severity", method = "concat_immediate_children",
    num_permutations = 100)$canonical_corr
head(res_expr_concatImmChild)
```

```
##   id severity.cancor severity.p severity.adjp severity.direction
## 1  1       0.8504976 0.03030303     0.4360897                 +
## 2  2       0.6428883 0.15151515     0.9911129                 +
## 3  3       0.8556124 0.03030303     0.4360897                 +
## 4  4       0.6743079 0.03030303     0.4360897                 +
## 5  5       0.6428883 0.15151515     0.9911129                 +
## 6  6       0.8807827 0.04040404     0.4845441                 +
##   severity.p.sign severity.adjp.sign     x y    label  leaf
## 1             sig                 ns  6.25 2      All FALSE
## 2              ns                 ns  0.00 1        B FALSE
## 3             sig                 ns  5.00 1        T FALSE
## 4             sig                 ns 12.50 1 Monocyte FALSE
## 5              ns                 ns  0.00 0     B_c7  TRUE
## 6             sig                 ns  1.00 0    T_c15  TRUE
```

```
# compare 3 methods (non-leaf nodes)
summary_ls <- list(res_expr %>%
    filter(!leaf) %>%
    select(id, label, severity.cancor) %>%
    rename(aggregate = severity.cancor), res_expr_concatLeaf %>%
    filter(!leaf) %>%
    select(id, label, severity.cancor) %>%
    rename(concatLeaf = severity.cancor), res_expr_concatImmChild %>%
    filter(!leaf) %>%
    select(id, label, severity.cancor) %>%
    rename(concatImmChild = severity.cancor))
summary <- Reduce(inner_join, summary_ls)
summary
```

```
##   id    label aggregate concatLeaf concatImmChild
## 1  1      All 0.8942939  0.7803956      0.8504976
## 2  2        B 0.6428883  0.6428883      0.6428883
## 3  3        T 0.9297493  0.8556124      0.8556124
## 4  4 Monocyte 0.8324347  0.6743079      0.6743079
```

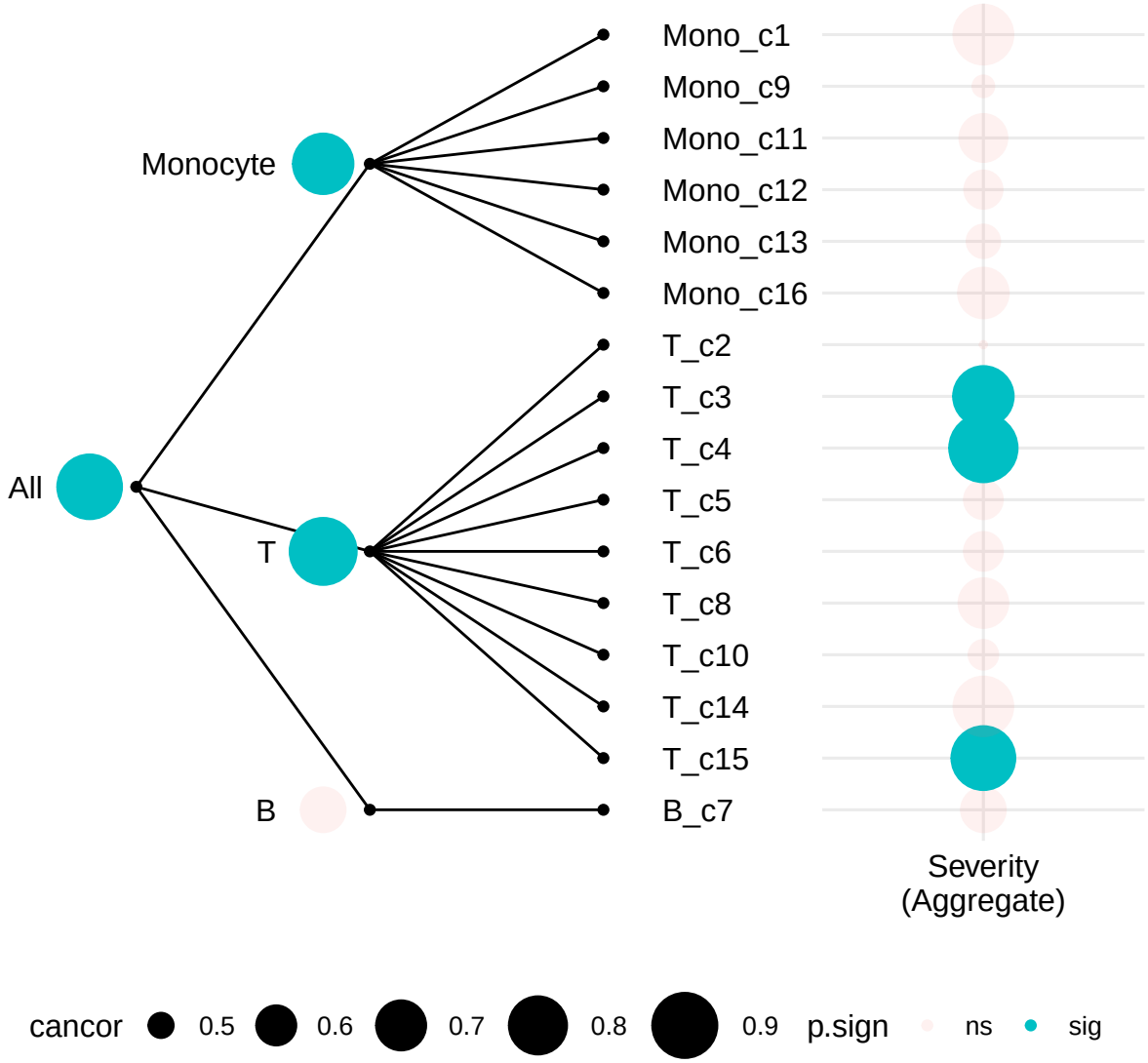
For some broad cell categories like B, T and Monocytes, the canonical correlation of leaf concatenation and immediate children concatenation are the same because the immediate children set is same as its corresponding leaf nodes. Depending on different tree hierarchies, three approaches may have different results and implications. This can also be visualized using TreeCorTreat plot:

```
# aggregate
colnames(res_expr) <- gsub("severity", "Severity\n(Aggregate)",
    colnames(res_expr))
# concat_leaf
colnames(res_expr_concatLeaf) <- gsub("severity", "Severity\n(ConcatLeaf)",
    colnames(res_expr_concatLeaf))
# concat_immediate_children
colnames(res_expr_concatImmChild) <- gsub("severity", "Severity\n(ConcatImmChild)",
    colnames(res_expr_concatImmChild))

# combine three approaches
res_combined <- Reduce(inner_join, list(res_expr, res_expr_concatLeaf,
    res_expr_concatImmChild))

# plot
treecortreatplot(hierarchy_structure, annotated_df = res_expr,
```

```
    response_variable = c("Severity\n(Aggregate)", "Severity\n(ConcatImmChild)",
        "Severity\n(ConcatLeaf)"), color_variable = "p.sign",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.4, nonleaf_point_gap = 0.2, plot_type = "balloon")
```



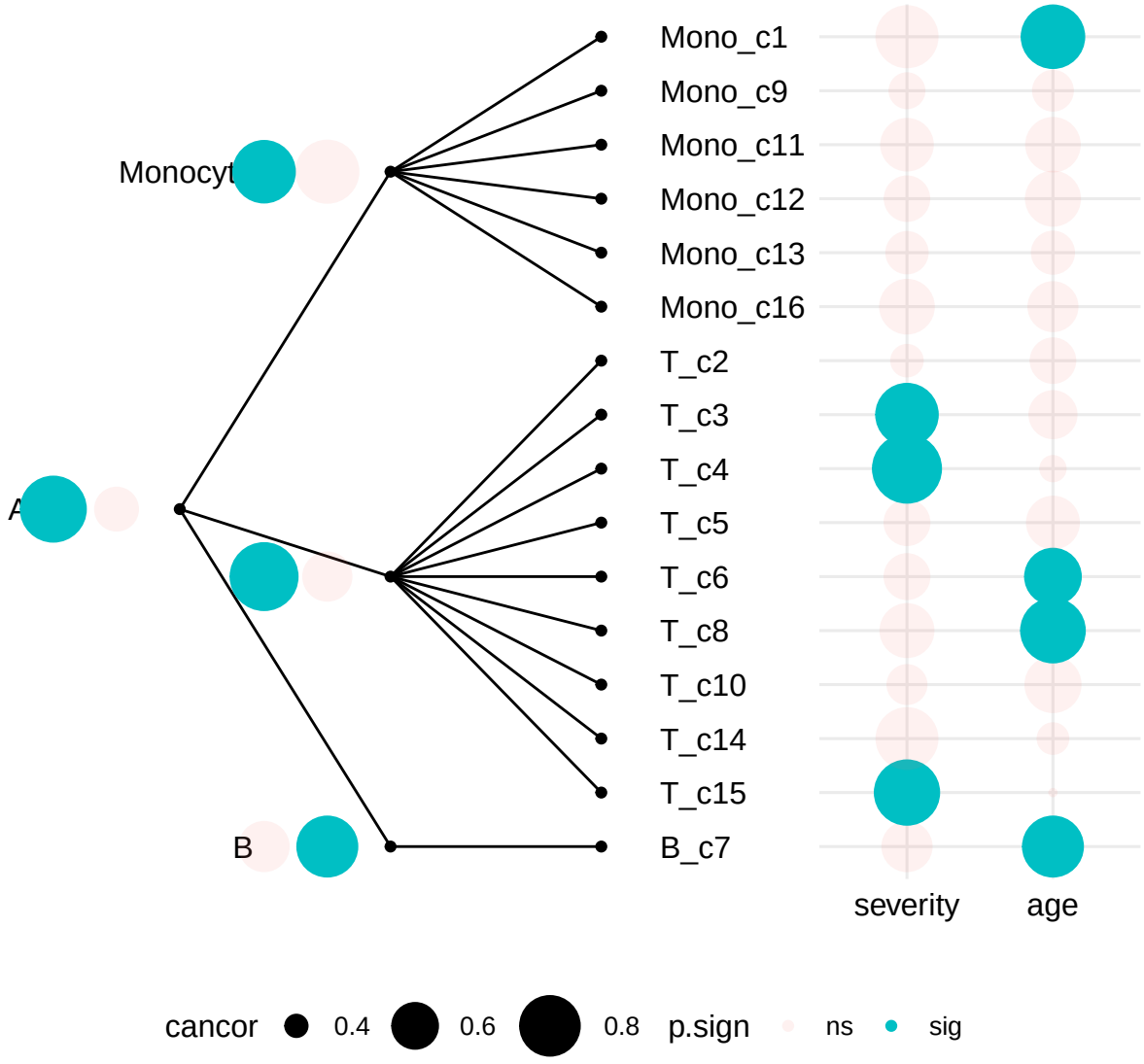
Analysis of multivariate outcomes

When there are multiple phenotypic traits, one can either analyze each trait separately as a univariate phenotype or analyze them jointly as a multivariate phenotype. For analyzing association between a multivariate phenotype and cell type proportion or global gene expression, CCA is used to compute the canonical correlation between the phenotype and cell type features (i.e. cell type proportion or gene expression).

Separate evaluation

```
# individually
multi_expr_sep <- treecor_expr(count, hierarchy_structure, cell_meta,
```

```
    sample_meta, response_variable = c("severity", "age"), num_permutations = 100)$canonical_corr
# visualize
treecortreatplot(hierarchy_structure, annotated_df = multi_expr_sep,
    response_variable = c("severity", "age"), separate = T, color_variable = "p.sign",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.65, nonleaf_point_gap = 0.3)
```

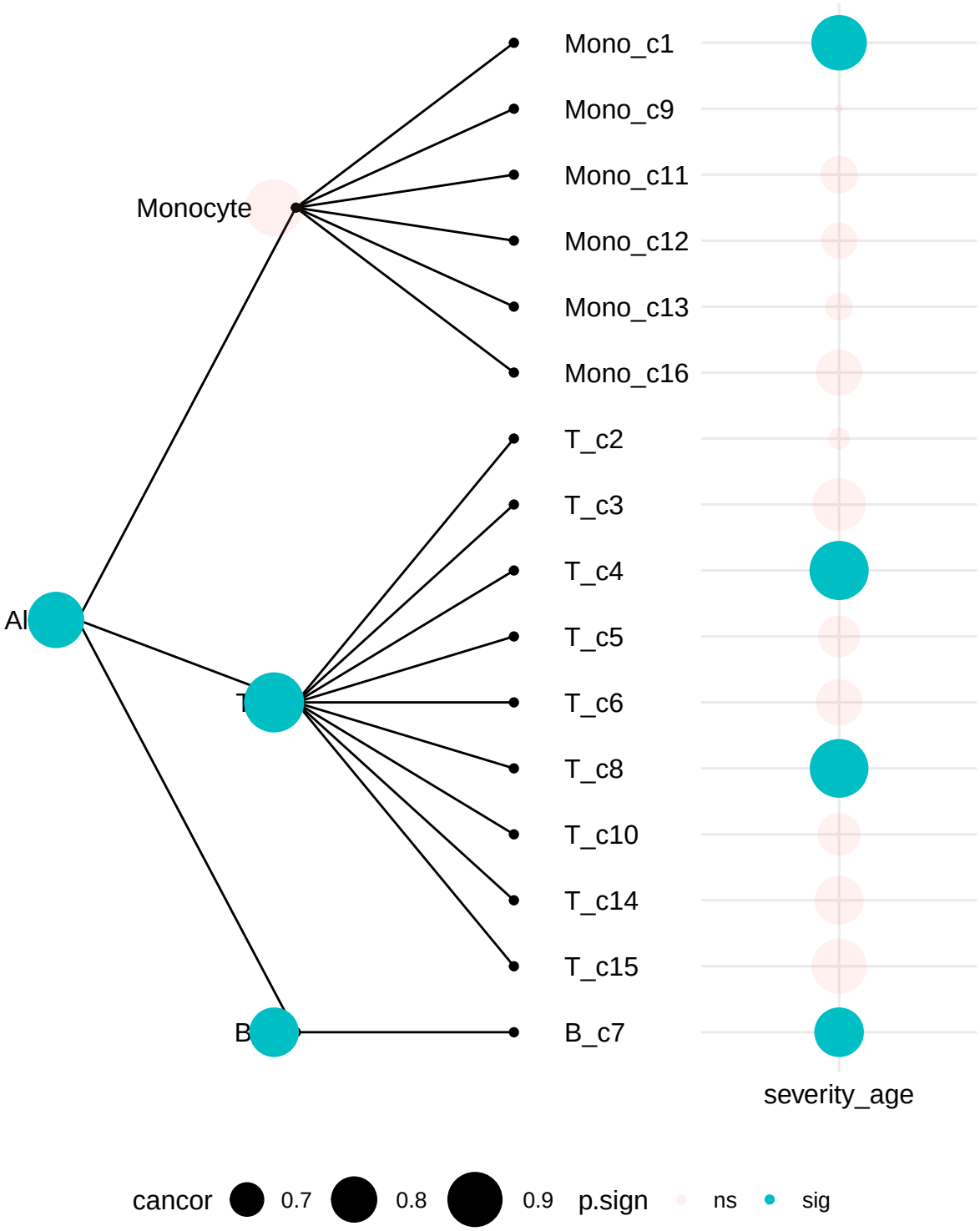


Joint evaluation

Alternatively, we can analyze both severity and age jointly.

```
# jointly
multi_expr_joint <- treecor_expr(count, hierarchy_structure,
    cell_meta, sample_meta, response_variable = c("severity",
        "age"), separate = F, num_permutations = 100)$canonical_corr
# visualize
treecortreatplot(hierarchy_structure, annotated_df = multi_expr_joint,
```

```
    response_variable = c("severity", "age"), separate = F, color_variable = "p.sign",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.2, nonleaf_point_gap = 0.1)
```



For analyzing differential gene expression, a multivariate phenotype is first transformed into a univariate

phenotype either using the principal component analysis or using a linear combination of traits with weights specified by users. The transformed univariate phenotype, which combines information from multiple traits, is then used to run differential expression analysis.

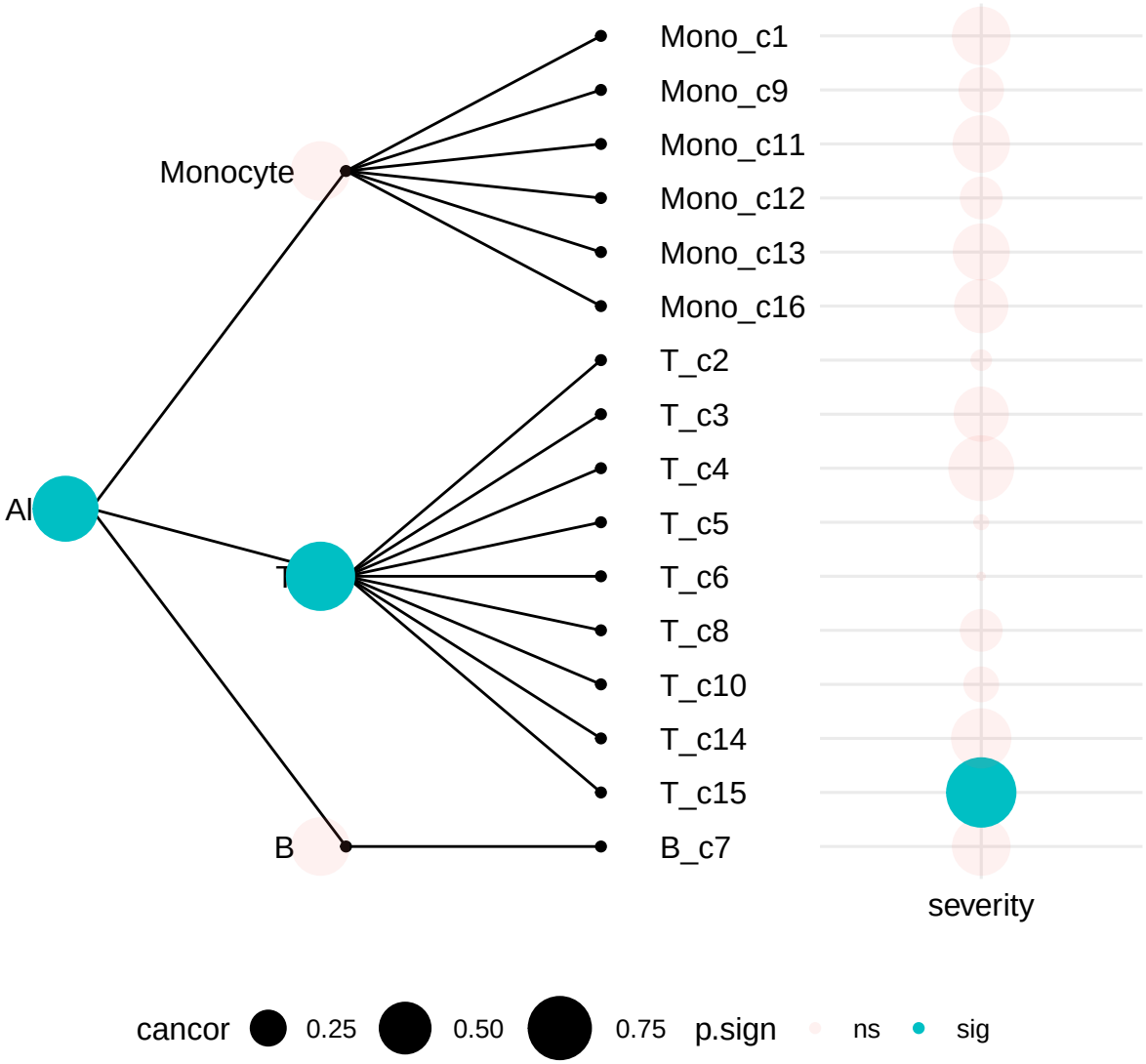
```
# jointly
multi_deg_joint <- treecor_deg(count, hierarchy_structure, cell_meta,
    sample_meta %>%
        mutate(severity = ifelse(severity == "HD", 0, 1)), response_variable = c("severity",
        "age"), separate = F, save_as_csv = F)$dge.summary
head(multi_deg_joint)
```

```
##      label combined_phenotype.num_deg     x y id  leaf
## 1      All                          1  6.25 2  1 FALSE
## 2        B                          0  0.00 1  2 FALSE
## 3        T                        700  5.00 1  3 FALSE
## 4 Monocyte                          0 12.50 1  4 FALSE
## 5     B_c7                          0  0.00 0  5  TRUE
## 6    T_c15                          0  1.00 0  6  TRUE
```

Adjusting for covariates in TreeCorTreat analysis

TreeCorTreat is capable of handling covariates in the analysis, allowing users to adjust for potential confounders or unwanted technical variation. For example, instead of viewing age as a phenotype, one can also view it as a covariate and ask which cell type features are associated with disease severity after accounting for age.

```
# adjust for age
expr_adjusted <- treecor_expr(count, hierarchy_structure, cell_meta,
    sample_meta, response_variable = "severity", formula = "~age",
    num_permutations = 100)$canonical_corr
# visualize
treecortreatplot(hierarchy_structure, annotated_df = expr_adjusted,
    response_variable = "severity", color_variable = "p.sign",
    size_variable = "cancor", alpha_variable = "p.sign", font_size = 12,
    nonleaf_label_pos = 0.2, nonleaf_point_gap = 0.1)
```


Mono_c1
Mono_c9
Mono_c11
Mono_c12
Mono_c13
Mono_c16
T_c2
T_c3
T_c4
T_c5
T_c6
T_c8
T_c10
T_c14
T_c15
B_c7
Monocyte
Al
T
B
severity
cancor
0.25
0.50
0.75
p.sign
ns
sig