Scaling up reproducible research for single cell transcriptomics using MetaNeighbor (Protocol 1)

1 Protocol 1: assessment of cell type replicability with unsupervised MetaNeighbor

Protocol 1 demonstrates how to compute and visualize cluster replicability across 4 human pancreas datasets. We will show steps detailing how to install MetaNeighbor, how to download and reformat the datasets with the SingleCellExperiment package, how to compute and interpret MetaNeighbor AUROCs. All code blocks can be run in R command line, Rstudio, RMarkdown notebooks or a jupyter notebook with an R kernel.

1.1 Step 0: Installation of MetaNeighbor and packages used in the protocol

1. We start by installing the latest MetaNeighbor package from the Gillis lab GitHub page.

```
if (!require('devtools')) {
  install.packages('devtools', quiet=TRUE)
}
```

```
## Loading required package: devtools
```

```
## Loading required package: usethis
```

```
#devtools::install_github("gillislab/MetaNeighbor")
devtools::install_github("gillislab/MetaNeighbor", ref="utility_dev")
```

```
## Skipping install of 'MetaNeighbor' from a github remote, the SHA1 (1da7f5e3) has not changed since la
##   Use `force = TRUE` to force installation
```

2. We also install the following packages, which are not necessary to run MetaNeighbor itself, but are needed to run the protocol.

```
to_install = c("scRNAseq", "tidyverse", "org.Hs.eg.db")
installed = sapply(to_install, requireNamespace)
```

```
## Loading required namespace: scRNAseq
```

```
## Loading required namespace: tidyverse
```

```
## Loading required namespace: org.Hs.eg.db
```

```
##
```

```
if (sum(!installed) > 0) {
    if (!requireNamespace("BiocManager", quietly = TRUE)) {
        install.packages("BiocManager")
        BiocManager::install()
    }
    BiocManager::install(to_install[!installed])
}
```

1.2 Step 1: creation of a merged SingleCellExperiment dataset

3. We consider 4 pancreatic datasets along with their independent annotation (from the original publication). MetaNeighbor expects a gene x cell matrix encapsulated in a SummarizedExperiment format. We recommend the SingleCellExperiment (SCE) package, because it is able to handle sparse matrix formats. We load the pancreas datasets using the scRNAseq package, which provide annotated datasets that are already in the SingleCellExperiment format:

```
library(scRNAseq)
```

```
## Loading required package: SingleCellExperiment
## Loading required package: SummarizedExperiment
## Loading required package: GenomicRanges
## Loading required package: stats4
## Loading required package: BiocGenerics
## Loading required package: parallel
##
## Attaching package: 'BiocGenerics'
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which, which.max, which.min
## Loading required package: S4Vectors
##
## Attaching package: 'S4Vectors'
## The following object is masked from 'package:base':
##
##     expand.grid
## Loading required package: IRanges
## Loading required package: GenomeInfoDb
## Loading required package: Biobase
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
```

```
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package: DelayedArray

## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians

##
## Attaching package: 'DelayedArray'

## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following objects are masked from 'package:base':
##
##     aperm, apply, rowsum
```

```
my_data <- list(
    baron = BaronPancreasData(),
    lawlor = LawlorPancreasData(),
    seger = SegerstolpePancreasData(),
    muraro = MuraroPancreasData()
)
```

```
## snapshotDate(): 2020-04-27

## see ?scRNAseq and browseVignettes('scRNAseq') for documentation

## loading from cache

## see ?scRNAseq and browseVignettes('scRNAseq') for documentation

## loading from cache

## snapshotDate(): 2020-04-27

## see ?scRNAseq and browseVignettes('scRNAseq') for documentation

## loading from cache

## see ?scRNAseq and browseVignettes('scRNAseq') for documentation

## loading from cache

## snapshotDate(): 2020-04-27

## see ?scRNAseq and browseVignettes('scRNAseq') for documentation

## loading from cache

## see ?scRNAseq and browseVignettes('scRNAseq') for documentation

## loading from cache

## see ?scRNAseq and browseVignettes('scRNAseq') for documentation

## loading from cache

## snapshotDate(): 2020-04-27
```

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

4. MetaNeighbor's "mergeSCE" function can be used to merge multiple SingleCellExperiment objects. Importantly, the output object will be restricted to genes, metadata columns and assays that are common to every dataset. Before we use "mergeSCE", we need to make sure that gene and metadata information align across datasets.

We start by checking if gene information aligns (stored in the "rownames" slot of the SCE object).

```
lapply(my_data, function(x) head(rownames(x), 3))
```

```
## $baron
## [1] "A1BG" "A1CF" "A2M"
##
## $lawlor
## [1] "ENSG00000229483" "ENSG00000232849" "ENSG00000229558"
##
## $seger
## [1] "SGIP1" "AZIN2" "CLIC4"
##
## $muraro
## [1] "A1BG-AS1__chr19" "A1BG__chr19"     "A1CF__chr10"
```

Two datasets (Baron, Segerstolpe) use gene symbols, one dataset (Muraro) combines symbols with chromosome information (to avoid duplicate gene names) and the last dataset (Lawlor) uses Ensemble identifiers. We convert all gene names to unique gene symbols. We start by converting gene names to symbols in the Muraro dataset, which are stored in the "rowData" slot of the SCE object:

```
rownames(my_data$muraro) <- rowData(my_data$muraro)$symbol
my_data$muraro <- my_data$muraro[!duplicated(rownames(my_data$muraro)),]
```

To circumvent the initial problem of genes with duplicate names, we also remove all duplicated symbols. Next, we convert Ensemble IDs to symbols in the Lawlor dataset, removing all IDs with no match and duplicated symbols:

```
library(org.Hs.eg.db)
```

```
## Loading required package: AnnotationDbi
```

```
symbols <- mapIds(org.Hs.eg.db, keys=rownames(my_data$lawlor), keytype="ENSEMBL", column="SYMBOL")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
keep <- !is.na(symbols) & !duplicated(symbols)
my_data$lawlor <- my_data$lawlor[keep,]
rownames(my_data$lawlor) <- symbols[keep]
```

5. We now turn our attention to metadata, which is stored in the "colData" slot of the SCE objects. Here we need to make sure that the column that contains cell type information is labeled identically in all datasets.

```
lapply(my_data, function(x) colnames(colData(x)))
```

```
## $baron
## [1] "donor" "label"
```

```
##
## $lawlor
## [1] "title"         "age"           "bmi"           "cell type"
## [5] "disease"       "islet unos id" "race"          "Sex"
##
## $seger
## [1] "Source Name"              "individual"
## [3] "single cell well quality" "cell type"
## [5] "disease"                  "sex"
## [7] "age"                      "body mass index"
##
## $muraro
## [1] "label" "donor" "plate"
```

Two datasets have the cell type information in the "cell type" column, the other two in the "label" column. For clarity, we add a "cell type" column in the latter two datasets.

```
my_data$baron$"cell type" <- my_data$baron$label
my_data$muraro$"cell type" <- my_data$muraro$label
```

6. Last, we check that count matrices, stored in the "assay" slot, have identical names.

```
lapply(my_data, function(x) names(assays(x)))
```

```
## $baron
## [1] "counts"
##
## $lawlor
## [1] "counts"
##
## $seger
## [1] "counts"
##
## $muraro
## [1] "counts"
```

7. Now that gene, cell type and count matrix information is aligned across datasets, we can create a merged dataset. "mergeSCE" takes a list of SCE objects as an input and outputs a single SCE object.

```
library(MetaNeighbor)
#devtools::load_all("~/projects/metaneighbor/MetaNeighbor")
fused_data = mergeSCE(my_data)
dim(fused_data)
```

```
## [1] 15295 15793
```

```
head(colData(fused_data))
```

```
## DataFrame with 6 rows and 2 columns
##                               cell type    study_id
##                              <character> <character>
## human1_lib1.final_cell_0001      acinar       baron
## human1_lib1.final_cell_0002      acinar       baron
## human1_lib1.final_cell_0003      acinar       baron
## human1_lib1.final_cell_0004      acinar       baron
## human1_lib1.final_cell_0005      acinar       baron
## human1_lib1.final_cell_0006      acinar       baron
```

The new dataset contains 15,295 common genes, 15,793 cells and two metadata columns: a concatenated "cell type" column, and "study_id", a column created by "mergeSCE" containing the name of the original study (corresponding to the names provided in the "my_data" list).

8. To avoid having to recreate the merged object, we recommend saving it as an RDS file.

```
saveRDS(fused_data, "merged_pancreas.rds")
```

1.3 Step 2: Hierarchical cluster replicability analysis

9. We load the MetaNeighbor (analysis) and the SingleCellExperiment (data handling) libraries, as well as the previously created pancreas dataset.

```
library(MetaNeighbor)
#devtools::load_all("~/projects/metaneighbor/MetaNeighbor")
library(SingleCellExperiment)

pancreas_data = readRDS("merged_pancreas.rds")
```

10. To perform neighbor voting, MetaNeighbor builds a cell-cell similarity network, which we defined as the Spearman correlation over a user-defined set of genes. We found that we obtained best results by picking genes that are highly variable across datasets, which can be picked using the "variableGenes" function.

```
system.time({
global_hvgs = variableGenes(dat = pancreas_data, exp_labels = pancreas_data$study_id)
})
```

```
##    user  system elapsed
##   9.848   0.948  10.822
```

```
length(global_hvgs)
```

```
## [1] 600
```

The function returns a list of 600 genes that were detected as highly variable in each of the 4 datasets.

11. The data and a set of biological meaningful genes is all we need to run MetaNeighbor and obtain cluster similarities.

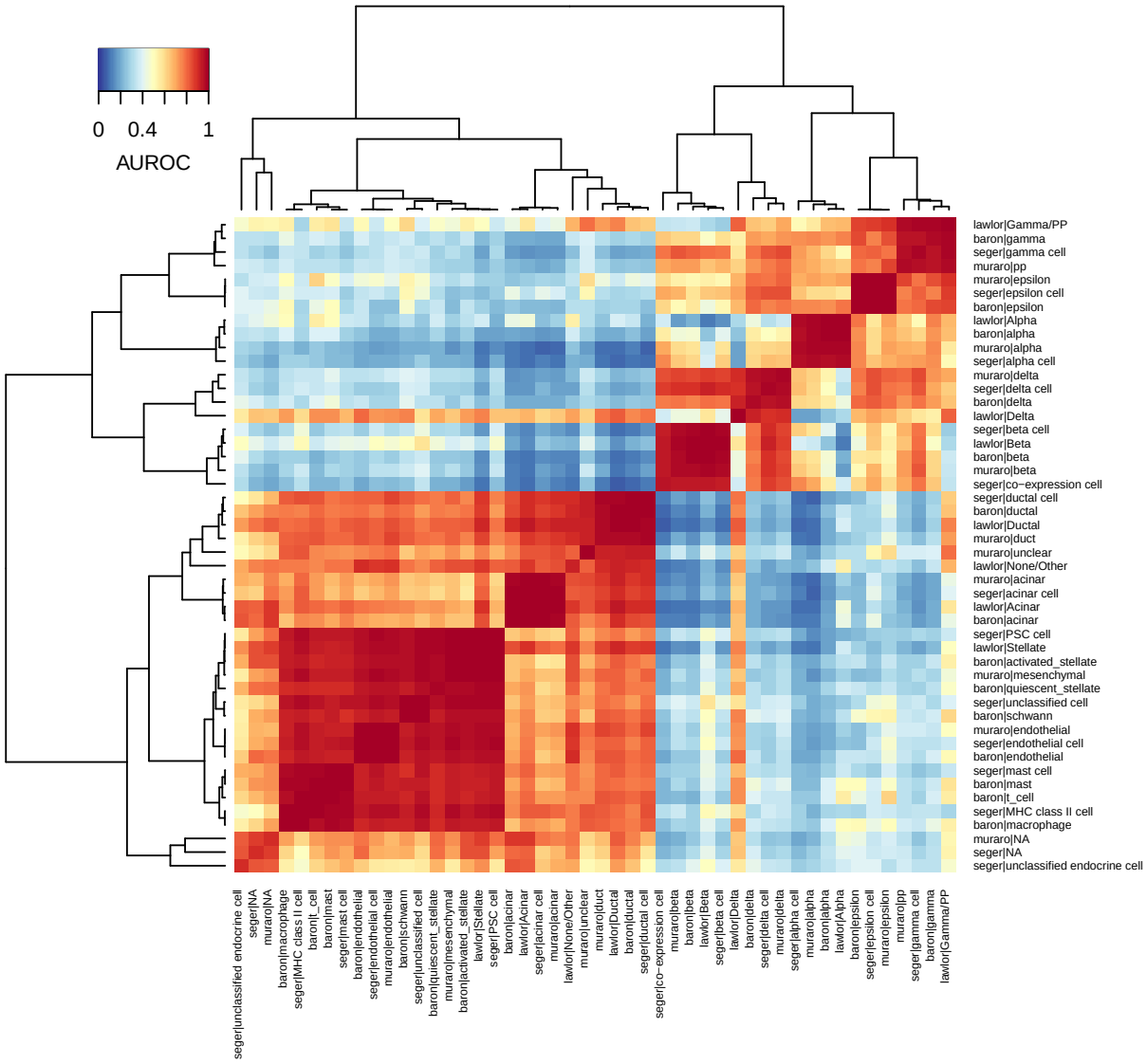
```
system.time({
aurocs = MetaNeighborUS(var_genes = global_hvgs,
                        dat = pancreas_data,
                        study_id = pancreas_data$study_id,
                        cell_type = pancreas_data$"cell type",
                        fast_version = TRUE)
})
```

```
##    user  system elapsed
##   6.889   5.840   1.168
```

Cluster similarities are defined as an Area Under the ROC curve (AUROC), which range between 0 and 1. The cross-dataset voting framework makes it batch-effect free (very different from average correlation)

12. For ease of interpretation the results can be visualized as a symmetric heatmap, where rows and columns are clusters from all datasets.

```
plotHeatmap(aurocs, cex = 0.5)
```



In the heatmap, the color of each square indicates the proximity of a pair of cluster, ranging from blue (low similarity) to red (high similarity). For example, "baron|gamma" (2nd row) is highly similar to "seger|gamma" (3rd column from the right) but very dissimilar from "muraro|ductal" (middle column). To group similar clusters together, "plotHeatmap" applies hierarchical clustering on the AUROC matrix. On the heatmap, we see two red blocks that indicate clear hierarchical structure in the data, with endocrine cell types clustering together (e.g., alpha, beta, gamma) and mesenchymal cells on the other side (e.g., amacrine, ductal, endothelial). Note that each red block is composed of smaller red blocks, indicating that clusters can be matched at an even higher resolution. We also see some off-diagonal patterns (e.g., lawlor|Gamma/PP, lawlor|Delta), which generally indicate the presence of doublets or contamination (presence of cells from other cell types), but what matters here is the clear presence of red blocks, which is a strong indicator of replicability.

13. To identify pairs of replicable clusters, we rely on a simple heuristics: a pair of cluster is replicable if they are reciprocal top hits (they preferentially vote for each other) and the AUROC exceeds a given threshold value (in our experience, 0.95 is a good heuristic value).

```
topHits(aurocs, dat = pancreas_data, study_id = pancreas_data$study_id, cell_type = pancreas_data$"cell
```

```
##        Study_ID|Celltype_1      Study_ID|Celltype_2 Mean_AUROC
## 1       seger|epsilon cell           muraro|epsilon       1.00
## 2       seger|epsilon cell            baron|epsilon       1.00
## 3               baron|mast          seger|mast cell       1.00
## 4   seger|endothelial cell       muraro|endothelial       1.00
## 5          lawlor|Stellate          seger|PSC cell       1.00
## 6         baron|macrophage  seger|MHC class II cell       1.00
## 7       muraro|endothelial        baron|endothelial       1.00
## 8          lawlor|Stellate baron|activated_stellate       1.00
## 9             baron|acinar            lawlor|Acinar       1.00
## 10          seger|PSC cell       muraro|mesenchymal       1.00
## 11            baron|alpha             lawlor|Alpha       1.00
## 12           lawlor|Acinar        seger|acinar cell       1.00
## 13           baron|schwann  seger|unclassified cell       1.00
## 14       seger|acinar cell            muraro|acinar       0.99
## 15             lawlor|Beta          seger|beta cell       0.99
## 16            baron|ductal        seger|ductal cell       0.99
## 17             lawlor|Beta               baron|beta       0.99
## 18            baron|ductal            lawlor|Ductal       0.99
## 19 seger|MHC class II cell             baron|t_cell       0.99
## 20             baron|gamma          lawlor|Gamma/PP       0.99
## 21             lawlor|Beta              muraro|beta       0.98
## 22       seger|ductal cell              muraro|duct       0.98
## 23            lawlor|Alpha             muraro|alpha       0.98
## 24          seger|PSC cell baron|quiescent_stellate       0.98
## 25         lawlor|Gamma/PP         seger|gamma cell       0.98
## 26        seger|delta cell             muraro|delta       0.98
## 27         lawlor|Gamma/PP                muraro|pp       0.98
## 28            muraro|alpha         seger|alpha cell       0.98
## 29            muraro|delta              baron|delta       0.96
##            Match_type
## 1  Reciprocal_top_hit
## 2          Above_0.95
## 3  Reciprocal_top_hit
## 4  Reciprocal_top_hit
## 5  Reciprocal_top_hit
## 6  Reciprocal_top_hit
## 7          Above_0.95
## 8          Above_0.95
## 9  Reciprocal_top_hit
## 10         Above_0.95
## 11 Reciprocal_top_hit
## 12         Above_0.95
## 13 Reciprocal_top_hit
## 14         Above_0.95
## 15 Reciprocal_top_hit
## 16 Reciprocal_top_hit
## 17         Above_0.95
## 18         Above_0.95
## 19         Above_0.95
## 20 Reciprocal_top_hit
## 21         Above_0.95
## 22         Above_0.95
## 23         Above_0.95
```

```
## 24         Above_0.95
## 25         Above_0.95
## 26 Reciprocal_top_hit
## 27         Above_0.95
## 28         Above_0.95
## 29         Above_0.95
```

We find a long list of replicable clusters within endocrine and mesenchymal cell types. This list provides strong evidence that these cell types are robust, as they are identified across all datasets with high AUROC.

14. In the case where there is a clear structure in the data (endocrine vs mesenchymal here), we can refine AUROCs by splitting the data. AUROCs have a simple interpretation: an AUROC of 0.6 indicates that cells from a given cell type are ranked in front of 60% of other test cells. However, this interpretation is out-group dependent: because endocrine cells represent ~65% of cells, even unrelated mesenchymal cell types will have an AUROC > 0.65, just because they will always be ranked in front of endocrine cells.

By starting with the full datasets, we uncovered the global structure in the data. However, to evaluate replicability of endocrine cell types and reduce dataset composition effects, we can make the assessment more stringent by restricting the outgroup to close cell types, i.e. by keeping only endocrine subtypes. We split cell types in two by using the "splitClusters" function and retain only endocrine cell types:

```
level1_split = splitClusters(aurocs, k = 2)
level1_split
```

```
## $`1`
##  [1] "baron|acinar"                    "baron|activated_stellate"
##  [3] "baron|ductal"                    "baron|endothelial"
##  [5] "baron|macrophage"                "baron|mast"
##  [7] "baron|quiescent_stellate"        "baron|schwann"
##  [9] "baron|t_cell"                    "lawlor|Acinar"
## [11] "lawlor|Ductal"                   "lawlor|None/Other"
## [13] "lawlor|Stellate"                 "seger|acinar cell"
## [15] "seger|ductal cell"               "seger|endothelial cell"
## [17] "seger|mast cell"                 "seger|MHC class II cell"
## [19] "seger|NA"                        "seger|PSC cell"
## [21] "seger|unclassified cell"         "seger|unclassified endocrine cell"
## [23] "muraro|acinar"                   "muraro|duct"
## [25] "muraro|endothelial"              "muraro|mesenchymal"
## [27] "muraro|NA"                       "muraro|unclear"
##
## $`2`
##  [1] "baron|alpha"              "baron|beta"
##  [3] "baron|delta"              "baron|epsilon"
##  [5] "baron|gamma"              "lawlor|Alpha"
##  [7] "lawlor|Beta"              "lawlor|Delta"
##  [9] "lawlor|Gamma/PP"          "seger|alpha cell"
## [11] "seger|beta cell"          "seger|co-expression cell"
## [13] "seger|delta cell"         "seger|epsilon cell"
## [15] "seger|gamma cell"         "muraro|alpha"
## [17] "muraro|beta"              "muraro|delta"
## [19] "muraro|epsilon"           "muraro|pp"
```

```
first_split = level1_split[[2]]
```

By outputting "level1_split" (not shown here), we found that the clusters were nicely split between mesenchymal and endocrine, and that endocrine clusters where in the second element of the list.

15. We repeat the MetaNeighbor analysis on endocrine cells only. First, we subset the data to the endocrine cell types (stored in "first_split").

```
to_keep = makeClusterName(pancreas_data$study_id, pancreas_data$"cell type") %in% first_split
subdata = pancreas_data[, to_keep]
dim(subdata)
```

```
## [1] 15295  9341
```

The new dataset contains the 9341 putative endocrine cells.

16. To focus on variability that is specific to endocrine cells, we re-pick highly variable genes:

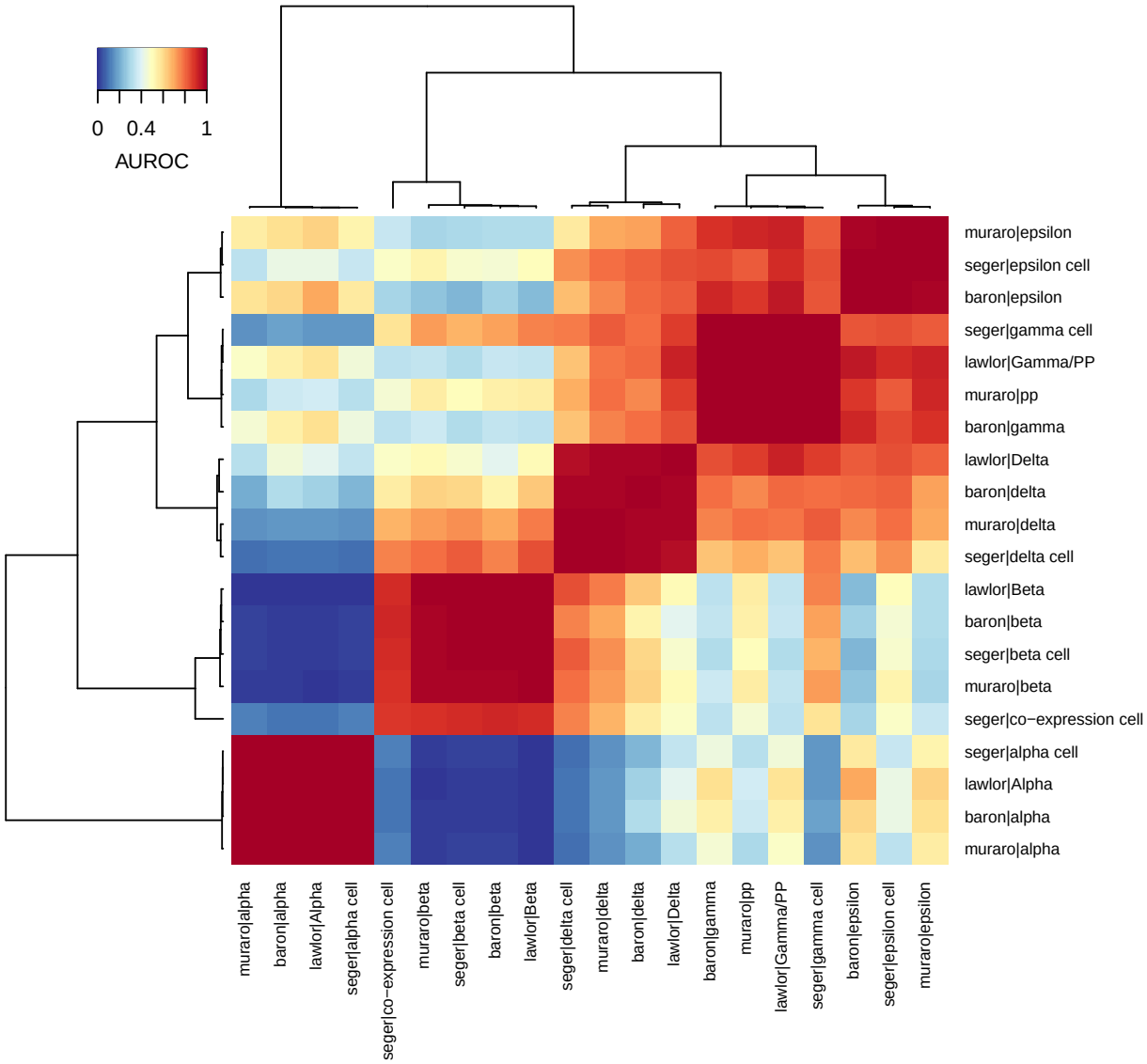
```
var_genes = variableGenes(dat = subdata, exp_labels = subdata$study_id)
```

17. Finally we recompute cluster similarities and visualize AUROCs.

```
system.time({
aurocs = MetaNeighborUS(var_genes = var_genes,
                        dat = subdata, fast_version = TRUE,
                        study_id = subdata$study_id,
                        cell_type = subdata$"cell type")
})
```

```
##    user  system elapsed
##   1.414   1.064   0.403
```

```
plotHeatmap(aurocs, cex = 0.7)
```



The resulting heatmap illustrates an example of a strong set of replicating clusters: when the assessment become more stringent (restriction to closely related cell types), the similarity of replicating clusters remains strong (AUROC~1 for alpha, beta, gamma, delta and epsilon cells) while the cross-cluster similarity has decreased (shift from red to blue, e.g. similarity of alpha and beta clusters has shifted from orange/red to dark blue) by virtue of zooming in on a subpart of the dataset.

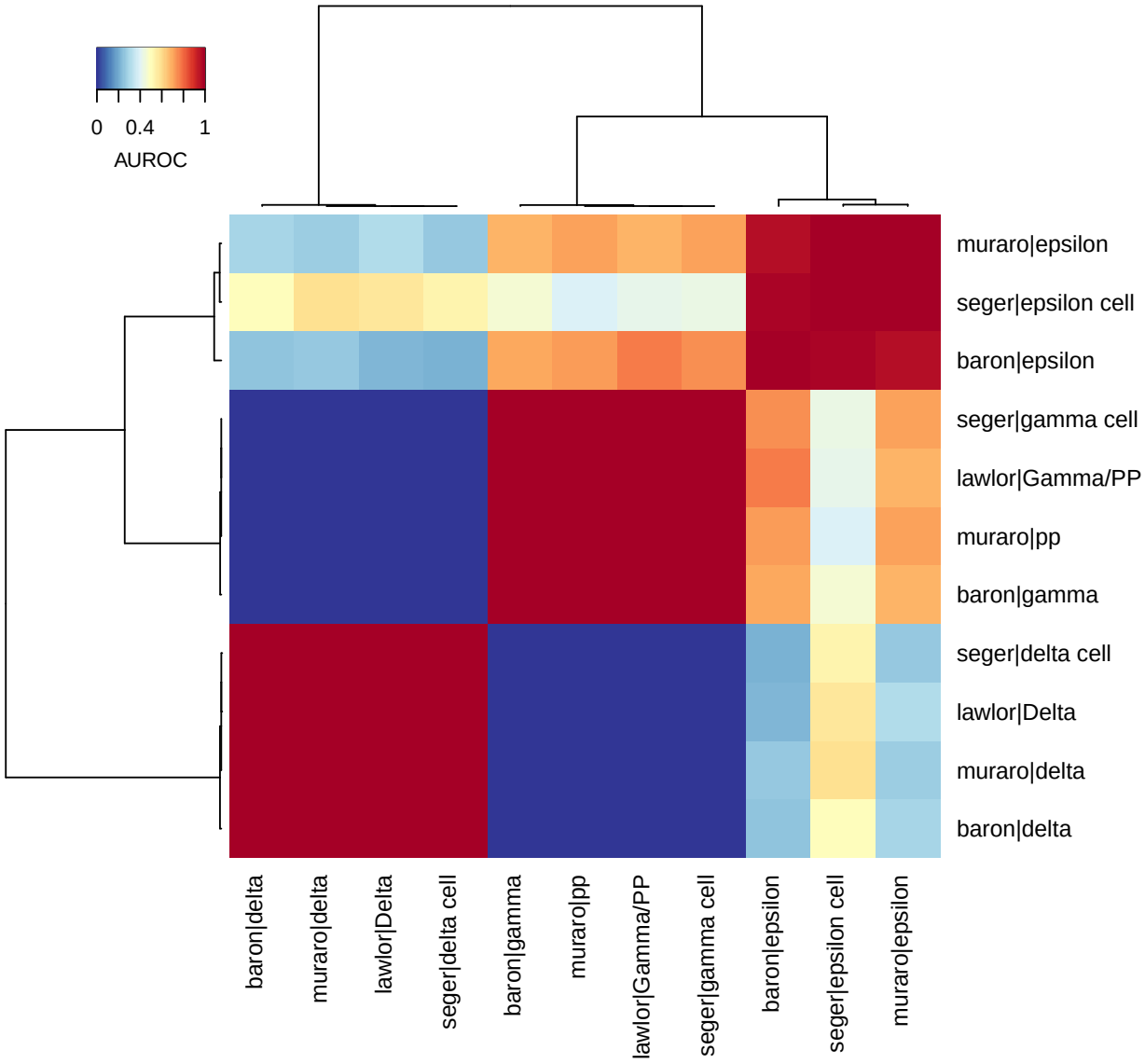
18. We can continue to zoom in as long as there are at least two cell types per dataset:

```
level2_split = splitClusters(aurocs, k = 3)
my_split = level2_split[[3]]
keep_cell = makeClusterName(pancreas_data$study_id, pancreas_data$"cell type") %in% my_split
subdata = pancreas_data[, keep_cell]
var_genes = variableGenes(dat = subdata, exp_labels = subdata$study_id)
length(var_genes)
```

```
## [1] 274
```

```
aurocs = MetaNeighborUS(var_genes = var_genes,
                        dat = subdata, fast_version = TRUE,
```

```
                        study_id = subdata$study_id,
                        cell_type = subdata$"cell type")
plotHeatmap(aurocs, cex = 1)
```



Here we removed the alpha and beta cells (representing close to 85% of endocrine cells) and validate that, even when restricting to neighboring cell types, there is still a clear distinction between delta, gamma and epsilon cells (AUROC ~ 1).

1.4 Step 3: stringent assessment of replicability with one-vs-best AUROCs

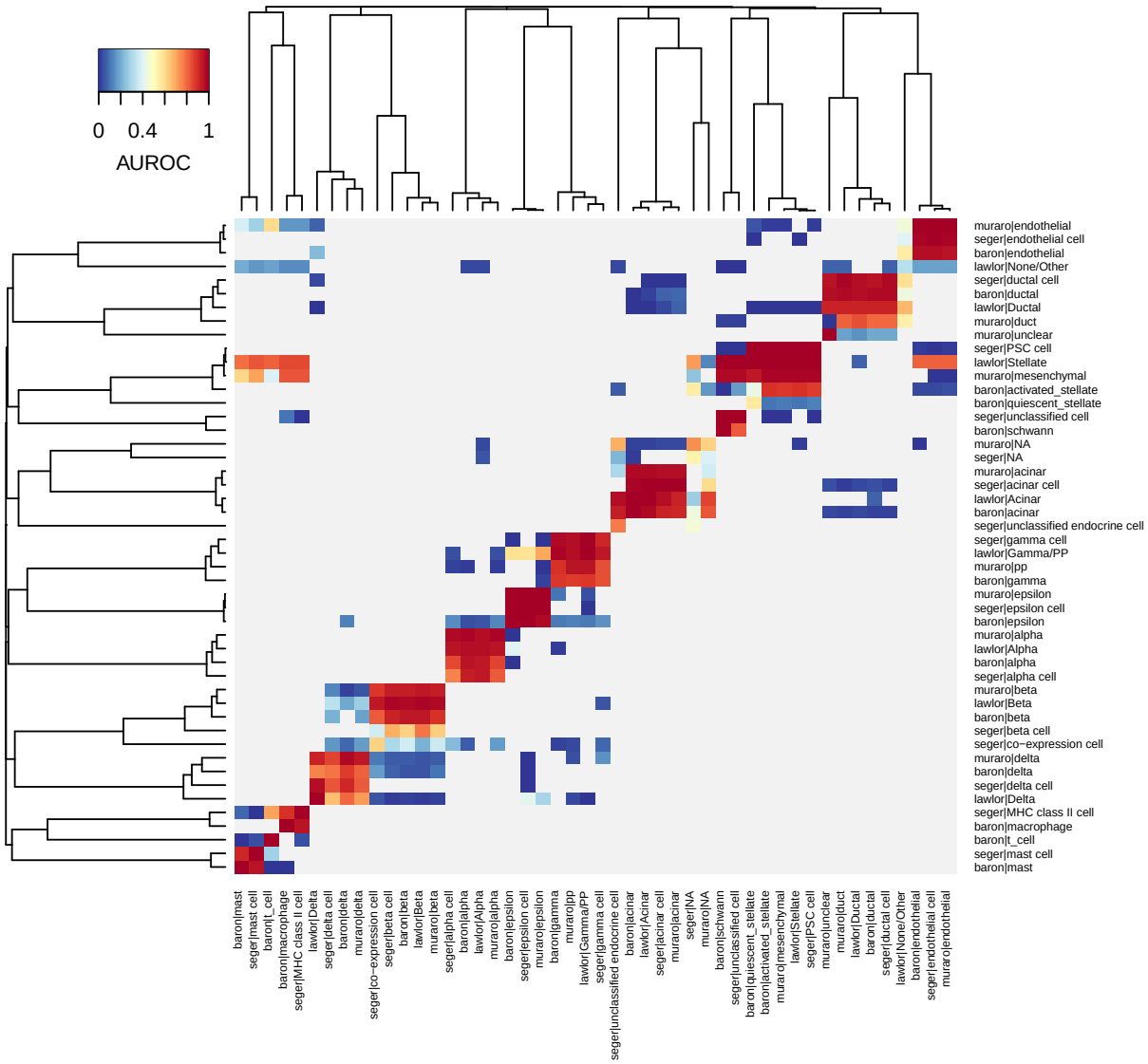
In the previous section, we created progressively more stringent replicability assessments of replicability by selecting more and more specific subsets of related cell types. As an alternative, we provide the "one-vs-best" parameter, which offers similar results without having to restrict the dataset by hand. In this scoring mode, MetaNeighbor will automatically identify the two closest matching clusters in each dataset and compute an AUROC based on the voting result for cells from the closest match against cells from the second closest match. Essentially, we are asking how easily a cluster can be distinguished from its closest neighbor.

19. To obtain one-vs-best AUROCs, we run the same command as before with two additional parameters: "one_vs_best = TRUE" and "symmetric_output = FALSE".

```
system.time({
best_hits = MetaNeighborUS(var_genes = global_hvgs,
                           dat = pancreas_data,
                           study_id = pancreas_data$study_id,
                           cell_type = pancreas_data$"cell type",
                           fast_version = TRUE,
                           one_vs_best = TRUE, symmetric_output = FALSE)
})
```

```
##    user  system elapsed
##  10.480   9.468   1.401
```

```
plotHeatmap(best_hits, cex = 0.5)
```



The interpretation of the heatmap is slightly different compared to one-vs-all AUROCs. First, since we only compare the two closest clusters, most cluster combinations are not tested (NAs, shown in gray on the

heatmap). Second, by setting "symmetric_output=FALSE", we broke the symmetric of the heatmap: train clusters are shown as columns and test clusters are shown as rows. Since each cluster is only tested against two clusters in each test dataset (closest and second closest match), we have 8 values per column (2 per dataset).

This representation helps to rapidly identify a cluster's closest hits as well as their closest outgroup. For example, ductal cells (2nd red square from the top right) strongly match with each other (one-vs-best AUROC>0.8) and acinar cells are their closest outgroup (blue segments in the same column). The non-symmetric view also makes it clear when best hits are not reciprocal. For example, mast cells (first two columns) heavily vote for "lawlor|Stellate" and "muraro|mesenchymal", but this vote is not reciprocal. This pattern indicates that the mast cell type is missing in the Lawlor and Muraro datasets (or that there are only a few mast cells that have been wrongly assigned to another cell type).

20. When using one-vs-best AUROCs, we recommend extracting replicating clusters as meta-clusters. Clusters are part of the same meta-cluster if they are reciprocal best hits. Note that if cluster 1 is the reciprocal best hit of 2 and 3, all three clusters are part of the same meta-cluster, even if 2 and 3 are not reciprocal best hits. To further filter for strongly replicating clusters, we specify an AUROC threshold (in our experience, 0.7 is a strong one-vs-best AUROC threshold).

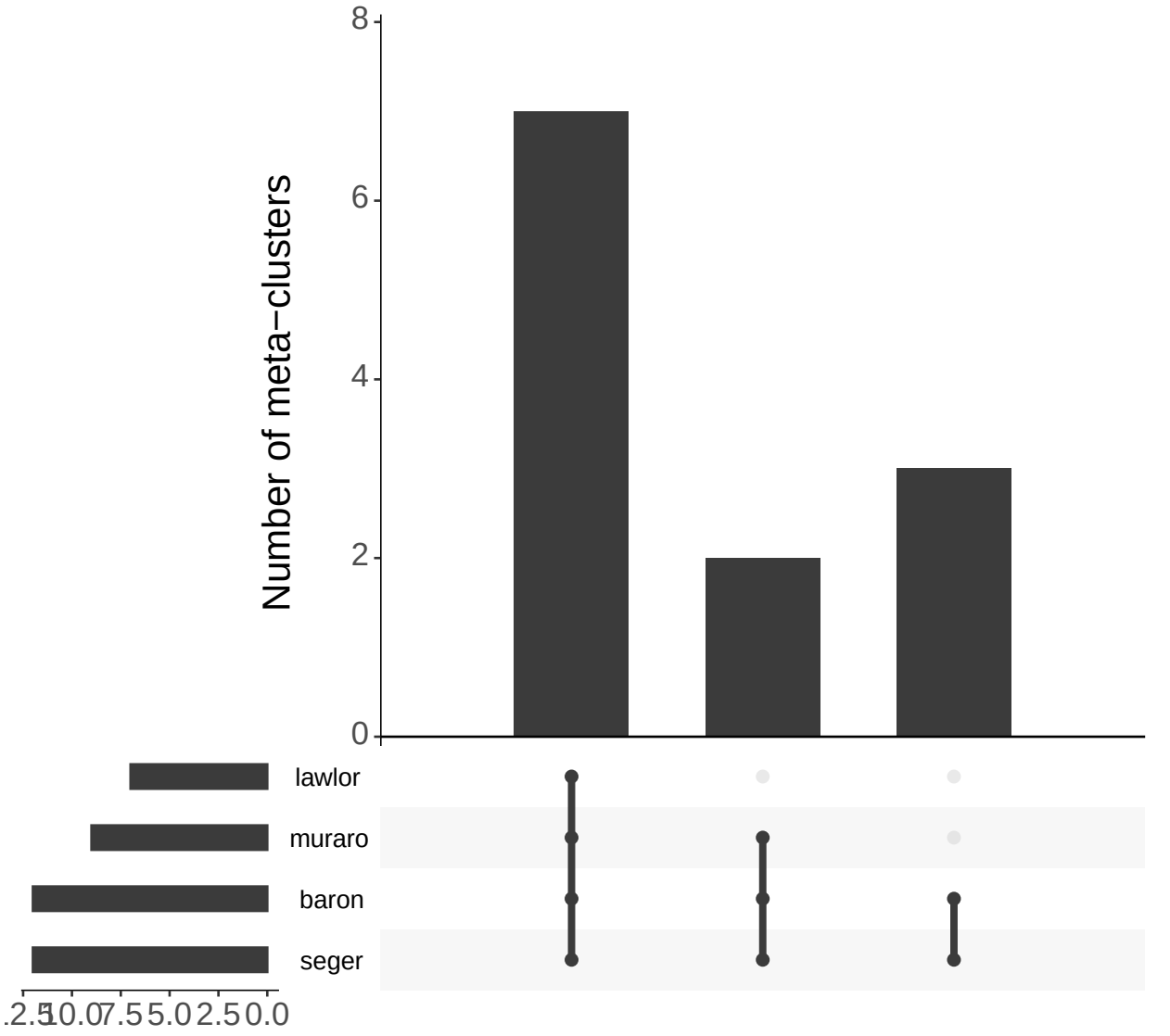
```r
mclusters = extractMetaClusters(best_hits, threshold = 0.7)
scoreMetaClusters(mclusters, best_hits)
```

```
##                    meta_cluster
## meta_cluster1   meta_cluster1
## meta_cluster2   meta_cluster2
## meta_cluster9   meta_cluster9
## meta_cluster3   meta_cluster3
## meta_cluster6   meta_cluster6
## meta_cluster4   meta_cluster4
## meta_cluster5   meta_cluster5
## meta_cluster8   meta_cluster8
## meta_cluster7   meta_cluster7
## meta_cluster11 meta_cluster11
## meta_cluster10 meta_cluster10
## meta_cluster12 meta_cluster12
## outliers             outliers
##
## meta_cluster1
## meta_cluster2                                                                                      baron|ac
## meta_cluster9
## meta_cluster3
## meta_cluster6
## meta_cluster4
## meta_cluster5
## meta_cluster8
## meta_cluster7
## meta_cluster11
## meta_cluster10
## meta_cluster12
## outliers       baron|quiescent_stellate; baron|t_cell; lawlor|None/Other; seger|co-expression cell; s
##                n_studies     score
## meta_cluster1          4 0.9717916
## meta_cluster2          4 0.9689141
## meta_cluster9          4 0.9303246
## meta_cluster3          4 0.9269162
## meta_cluster6          4 0.9207863
```

```
## meta_cluster4          4 0.8824853
## meta_cluster5          4 0.8553828
## meta_cluster8          3 0.9962277
## meta_cluster7          3 0.9832680
## meta_cluster11         2 0.9671429
## meta_cluster10         2 0.9637792
## meta_cluster12         2 0.9534128
## outliers               1        NA
```

The "scoreMetaClusters" provides a good summary of meta-clusters, ordering cell types by the number of datasets in which they replicate, then by average AUROC. We find 12 cell types that have strong support across at least 2 datasets, with 7 cell types replicating across all 4 datasets. 8 cell types are tagged as "outlier", as they had no strong match. These cell types usually contain doublets, low quality cells or contaminated cell types. The replicability structure described here can be summarized as an Upset plot.

```
plotUpset(mclusters)
```



Meta-clusters can also be visualized as heatmaps (called "cell-type badges") with the "plotMetaClusters" function (full output not shown here). Each badge shows an AUROC heatmap restricted to each specific

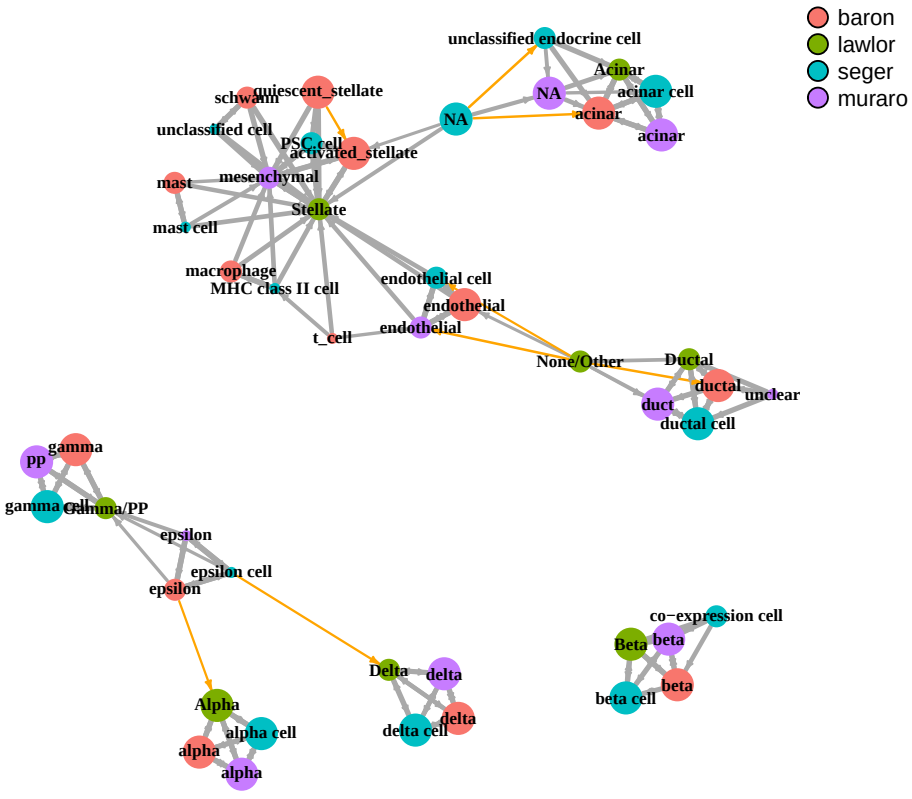
meta-cluster. These badges help diagnose cases where AUROCs are lower in a specific train or test dataset. For example, the "muraro|duct" cell type has systematically lower AUROCs, likely indicating the presence of contaminating cells in another cell type (probably "muraro|unclear", referring to the original heatmap).

```
pdf("meta_clusters.pdf")
plotMetaClusters(mclusters, best_hits)
dev.off()
```

```
## pdf
##   2
```

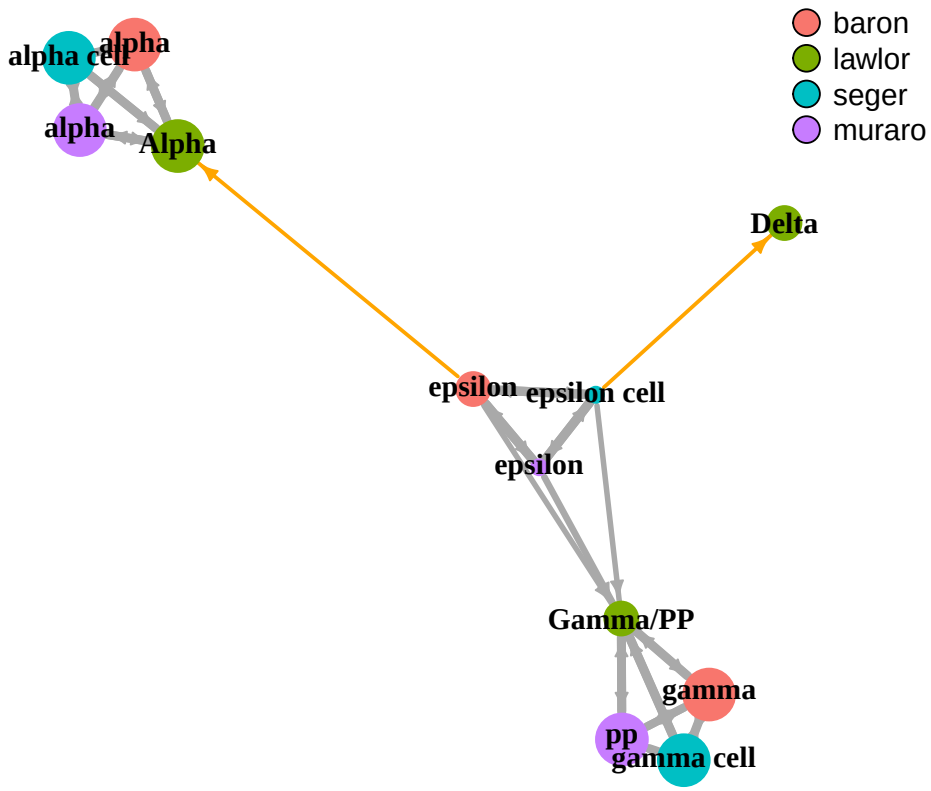
21. The last visualization is an alternative representation of the AUROC heatmap as a graph, which is particularly useful for large datasets. In this graph, top votes (AUROC > 0.5) are shown in black, while outgroup votes (AUROC < 0.5) are shown in orange. To highlight close calls, we recommend keeping only strong outgroup votes, here with AUROC $>= 0.4$.

```
cluster_graph = makeClusterGraph(best_hits, low_threshold = 0.4)
plotClusterGraph(cluster_graph, pancreas_data$study_id, pancreas_data$"cell type", size_factor=3, legend
```



We note that there are several orange edges, indicating that some cell types had two close matches. To investigate the origin of these close calls, we take "baron|epsilon" as our cluster of interest (coi), query its closest neighbors with "extendClusterSet", then zoom in on its subgraph with "subsetClusterGraph".

```
coi = "baron|epsilon"
coi = extendClusterSet(cluster_graph, initial_set = coi, max_neighbor_distance = 2)
subgraph = subsetClusterGraph(cluster_graph, coi)
plotClusterGraph(subgraph, pancreas_data$study_id, pancreas_data$"cell type", size_factor=5, legend_cex=
```




```
best_hits[coi, coi]
```

```
##                  baron|alpha baron|epsilon baron|gamma lawlor|Alpha
## baron|alpha       0.95411770  9.553836e-05          NA   0.94260533
## baron|epsilon     0.04588230  9.999045e-01   0.1176471   0.05739467
## baron|gamma              NA            NA   0.8823529           NA
## lawlor|Alpha      0.96074915  4.049279e-01   0.0172013   0.96832038
## lawlor|Delta             NA            NA          NA           NA
## lawlor|Gamma/PP          NA  5.950721e-01   0.9827987           NA
```

```
## seger|alpha cell    0.93471089           NA          NA   0.94793758
## seger|epsilon cell          NA  9.992748e-01          NA           NA
## seger|gamma cell            NA  7.251632e-04   0.9742288           NA
## muraro|alpha        0.98558748  0.000000e+00          NA   0.96437772
## muraro|epsilon              NA  1.000000e+00   0.1023102           NA
## muraro|pp           0.01441252           NA   0.8976898           NA
##                   lawlor|Delta lawlor|Gamma/PP seger|alpha cell
## baron|alpha                 NA             NA       0.85256043
## baron|epsilon               NA     0.116122004       0.14743957
## baron|gamma                 NA     0.883877996               NA
## lawlor|Alpha                NA             NA       0.95025570
## lawlor|Delta                 1     0.000000000               NA
## lawlor|Gamma/PP             NA     1.000000000       0.04974430
## seger|alpha cell            NA             NA       0.76020142
## seger|epsilon cell          NA     0.004350979               NA
## seger|gamma cell            NA     0.995649021               NA
## muraro|alpha                NA             NA       0.97423548
## muraro|epsilon              NA     0.042904290               NA
## muraro|pp                   NA     0.957095710       0.02576452
##                   seger|epsilon cell seger|gamma cell muraro|alpha
## baron|alpha                       NA               NA   0.86892137
## baron|epsilon              0.9927898        0.1549020   0.13107863
## baron|gamma                       NA        0.8450980           NA
## lawlor|Alpha                      NA               NA   0.95722920
## lawlor|Delta               0.4111111               NA           NA
## lawlor|Gamma/PP            0.5888889        0.9417088   0.04277080
## seger|alpha cell                  NA               NA   0.82760317
## seger|epsilon cell         1.0000000               NA           NA
## seger|gamma cell                  NA        0.9157881           NA
## muraro|alpha                      NA               NA   0.98124665
## muraro|epsilon             1.0000000               NA           NA
## muraro|pp                         NA        0.8423024   0.01875335
##                   muraro|epsilon  muraro|pp
## baron|alpha                   NA         NA
## baron|epsilon         0.978431373 0.12309368
## baron|gamma           0.021568627 0.87690632
## lawlor|Alpha                  NA         NA
## lawlor|Delta          0.291111111 0.03111111
## lawlor|Gamma/PP       0.708888889 0.96888889
## seger|alpha cell              NA         NA
## seger|epsilon cell    0.995649021         NA
## seger|gamma cell      0.004350979 0.96954315
## muraro|alpha                  NA         NA
## muraro|epsilon        1.000000000         NA
## muraro|pp             0.000000000 0.95126456
```

Here the explanation of the presence of the orange edges is relatively straightforward: the epsilon cell type seems to be missing in the Lawlor dataset, so votes from "baron|epsilon" were equally split between "Lawlor|Gamma/PP" and "Lawlor|Alpha".

In general, the cluster graph can be used to understand how meta-clusters are extracted, why some clusters are tagged and outliers and diagnose problems where resolution of cell types differs across datasets.