Measure cell type replicability using MetaNeighbor (BICCN workshop)

This notebook illustrates how MetaNeighbor can be used to compute cell type replicability across scRNAseq datasets. Because MetaNeighbor does not require dataset integration and alleviates batch effects using a neighbor voting approaches, it provides an extremely efficient procedure to compare cell types across datasets. In BICCN efforts, MetaNeighbor was used to measure the consistency of cell type taxonomies across single-cell technologies and clustering pipelines. Intuitively, if there is shared biological variation that drive cell type differences, we expect to find highly replicable cell types, even across datasets that have been processed independently. On the other hand, if data have been overclustered (clusters driven by technical variability), we expect low values of replicability across datasets and imperfect matches across cell types.

In this notebook, we'll run MetaNeighbor on a subset of GABAergic cell types from 3 BICCN datasets to illustrate how to set up a MetaNeighbor pipeline and interpret its output. For a more comprehensive description of the process, please take a look at these protocols.

Setup MetaNeighbor

We start by installing MetaNeighbor:

```
if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
    BiocManager::install()
}
BiocManager::install("MetaNeighbor")
```

Note that you need Bioconductor version 3.12 or higher to get an up-to-date version of MetaNeighbor. If you're having trouble installing from Bioconductor, consider installing the development version of MetaNeighbor (`devtools::install_github("gillislab/MetaNeighbor")`).

Download BICCN data

We'll consider 3 datasets in this notebook, which are subsets of the BICCN mouse MOp data. The truncated datasets can be downloaded directly from FigShare:

```
download.file("https://figshare.com/ndownloader/files/35875268", "scCv3.rds")
download.file("https://figshare.com/ndownloader/files/35875271", "scSS.rds")
download.file("https://figshare.com/ndownloader/files/35875274", "snCv2.rds")
download.file("https://figshare.com/ndownloader/files/35875277", "go_mouse.rds")
```

Data preprocessing

The input for MetaNeighbor is single-cell data in the SingleCellExperiment (SCE) format, which was automatically installed from Bioconductor when we installed MetaNeighbor. All the data are provided in SCE

format, but note that Seurat objects can easily be converted to SCE using the `as.SingleCellExperiment` function. We'll use 3 datasets reprensenting 3 different technologies (single-cell Smart-Seq - scss, single-nuclei 10X - snCv2, single-cell 10X - scCv3). All datasets contain GABAergic inhibtory neurons from the mouse primary motor cortex. We start by loading these datasets:

```
library(SingleCellExperiment)
scss = readRDS("scSS.rds")
scss
```

```
## class: SingleCellExperiment
## dim: 45768 2092
## metadata(0):
## assays(1): counts
## rownames(45768): 0610005C13Rik 0610006L08Rik ... n-R5s146 n-R5s149
## rowData names(0):
## colnames(2092): LS-15395_S41_E1-50 LS-15395_S42_E1-50 ...
##   SM-GE92M_S095_E1-50 SM-GE92M_S096_E1-50
## colData names(2): label sublabel
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
```

```
sn10x = readRDS("snCv2.rds")
sn10x
```

```
## class: SingleCellExperiment
## dim: 31053 5000
## metadata(0):
## assays(1): counts
## rownames(31053): Xkr4 Gm1992 ... Vmn2r122 CAAA01147332.1
## rowData names(0):
## colnames(5000): GACTAACGTCTAGGTT-4 GGGAATGAGCTAACAA-2 ...
##   GTACTCCTCCTCCTAG-1 TTCTTAGAGTACGTAA-3
## colData names(2): label sublabel
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
```

```
sc10x = readRDS("scCv3.rds")
sc10x
```

```
## class: SingleCellExperiment
## dim: 31053 5000
## metadata(0):
## assays(1): counts
## rownames(31053): Xkr4 Gm1992 ... Vmn2r122 CAAA01147332.1
## rowData names(0):
## colnames(5000): TTTCCTCCATCCCACT-4 GACTATGTCCGGTAAT-1 ...
##   GACGCTGAGACCGCCT-10 GACCGTGGTTCTTCAT-5
## colData names(2): label sublabel
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
```

To perform cross-dataset comparisons, we need to merge all the data as a single SCE object, which can be achieved using `MetaNeighbor::mergeSCE`:

```
library(MetaNeighbor)
all_data = mergeSCE(list(scss = scss, sn10x = sn10x, sc10x = sc10x))
all_data
```

```
## class: SingleCellExperiment
## dim: 24140 12092
## metadata(0):
## assays(1): counts
## rownames(24140): 0610005C13Rik 0610006L08Rik ... Zzz3 a
## rowData names(0):
## colnames(12092): LS-15395_S41_E1-50 LS-15395_S42_E1-50 ...
##   GACGCTGAGACCGCCT-10 GACCGTGGTTCTTCAT-5
## colData names(3): label sublabel study_id
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
```

```
head(colData(all_data))
```

```
## DataFrame with 6 rows and 3 columns
##                          label      sublabel    study_id
##                    <character>   <character> <character>
## LS-15395_S41_E1-50         Sst    Sst Myh8_1        scss
## LS-15395_S42_E1-50         Sst    Sst Myh8_1        scss
## LS-15395_S47_E1-50         Vip    Vip Chat_1        scss
## LS-15395_S49_E1-50       Lamp5 Lamp5 Slc35d3        scss
## LS-15395_S51_E1-50         Sst    Sst Myh8_2        scss
## LS-15395_S53_E1-50         Sst      Sst Etv1        scss
```

We obtain a new dataset containing ~24k genes, ~20k cells and 4 metadata columns which are stored in the `colData` slot from the SCE object. Note that `MetaNeighbor::mergeSCE` will automatically restrict the datasets to a *common* set of genes and only keep *common* metadata columns. Make sure that your datasets have normalized gene names and metadata names before merging them!

Compute cell-type similarity across datasets

The MetaNeighbor algorithm has 2 simple steps: (a) compute highly variable genes, (b) computing cell type replicabilities. The data do not need to be normalized or integrated before running MetaNeighbor. By computing highly variable genes, we remove noisy components of the transcriptome that would lead to underestimated cell type similarities.

```
hvg = variableGenes(dat = all_data, i = "counts", exp_labels = all_data$study_id)
length(hvg)
```

```
## [1] 999
```

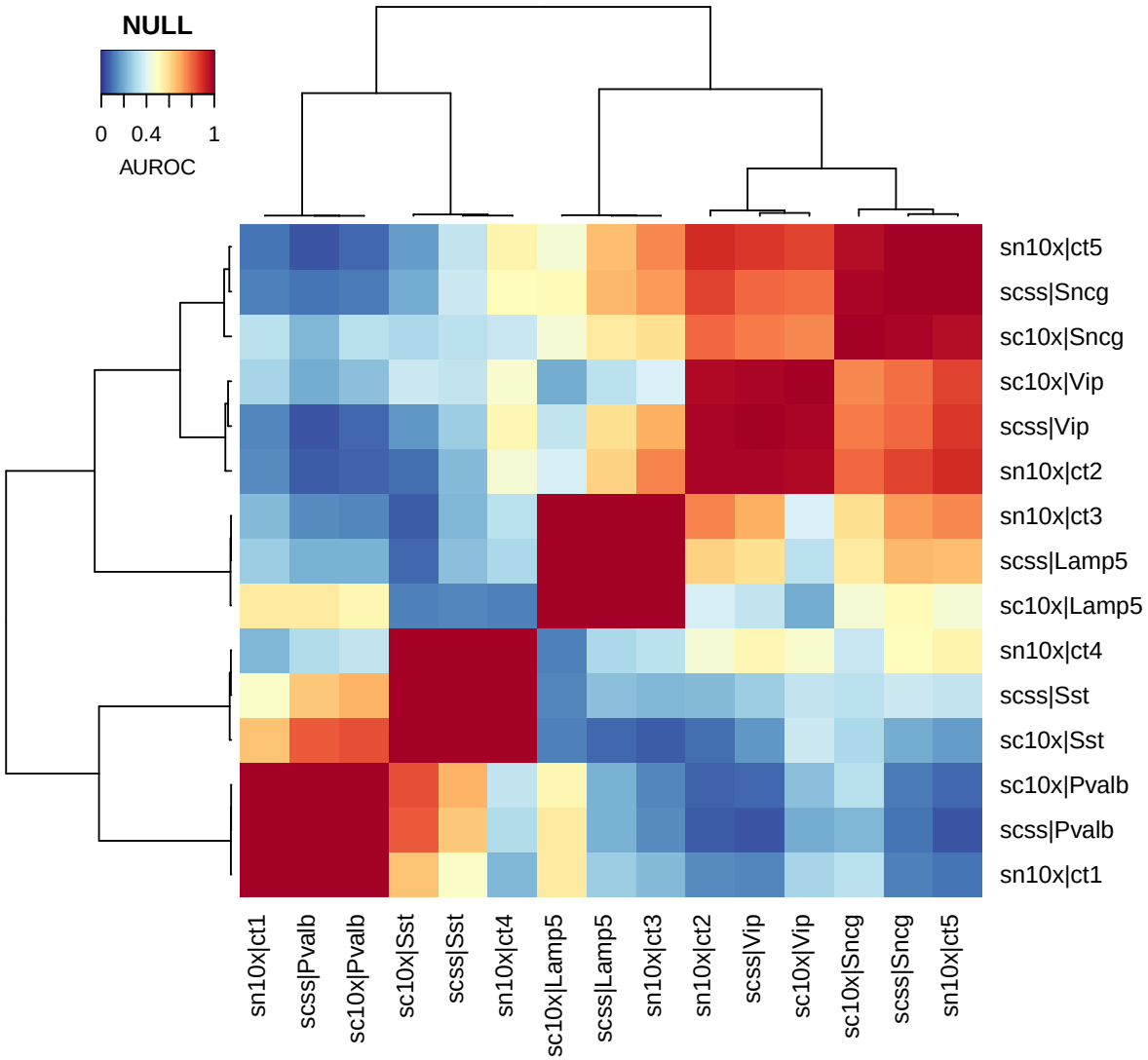
MetaNeighbor's `variableGenes` function takes 3 arguments: the merged SCE object, the slot where counts are stored in the SCE object and the metadata vector containing the dataset labels. The function computes highly variable genes, then looks for genes that are shared across datasets, resulting in a total of 999 genes. We're now ready to run MetaNeighbor itself:

```
auroc = MetaNeighborUS(var_genes = hvg, dat = all_data, i ="counts",
                       study_id=all_data$study_id, cell_type = all_data$label,
                       fast_version = TRUE)
```

`MetaNeighborUS` is the main MetaNeighbor function, it takes as input a set of highly variable genes, a merged SCE object, the name of the SCE slot containing counts, a metadata vector with dataset labels and a metadata vector with cell type labels. By default, MetaNeighbor runs a legacy version for backward compatibility, but for any decently sized dataset, you should use the `fast_version` of MetaNeighbor, which is up to 1000x faster and less memory-intensive.

`MetaNeighborUS` returns a similarity matrix of cell types, which is quantified as AUROCs. Let's plot this matrix:

```
plotHeatmap(auroc)
```



MetaNeighbor computes cell type similarities as an AUROC by using a classification framework. The question we ask is: How easily can a reference type predict a target type in another dataset? Reference cells vote for their closest neighbors according to their transcriptomic correlations: if reference and target cells match, they tend to be neighbors and target cells receive higher votes than than non-target cells, resulting in a high prediction score (AUROC close to 1). AUROCs have an interpretation that is similar to correlation, except that values range from 0 to 1, where 0.5 indicates no association. Here, we find 5 clear groups of clusters that have high replicability values (AUROC>0.9): Sncg, Vip, Lamp5, Sst, Pvalb. Note that we hid the cell types

labels from one of the datasets. With MetaNeighbor we can quickly and easily assign the missing labels, e.g., ct1 is Pvalb and ct2 is Vip.

To visualize results as a table instead of a matrix, we can use the `topHitsByStudy` function:

```
topHitsByStudy(auroc = auroc, threshold=0.9)
```

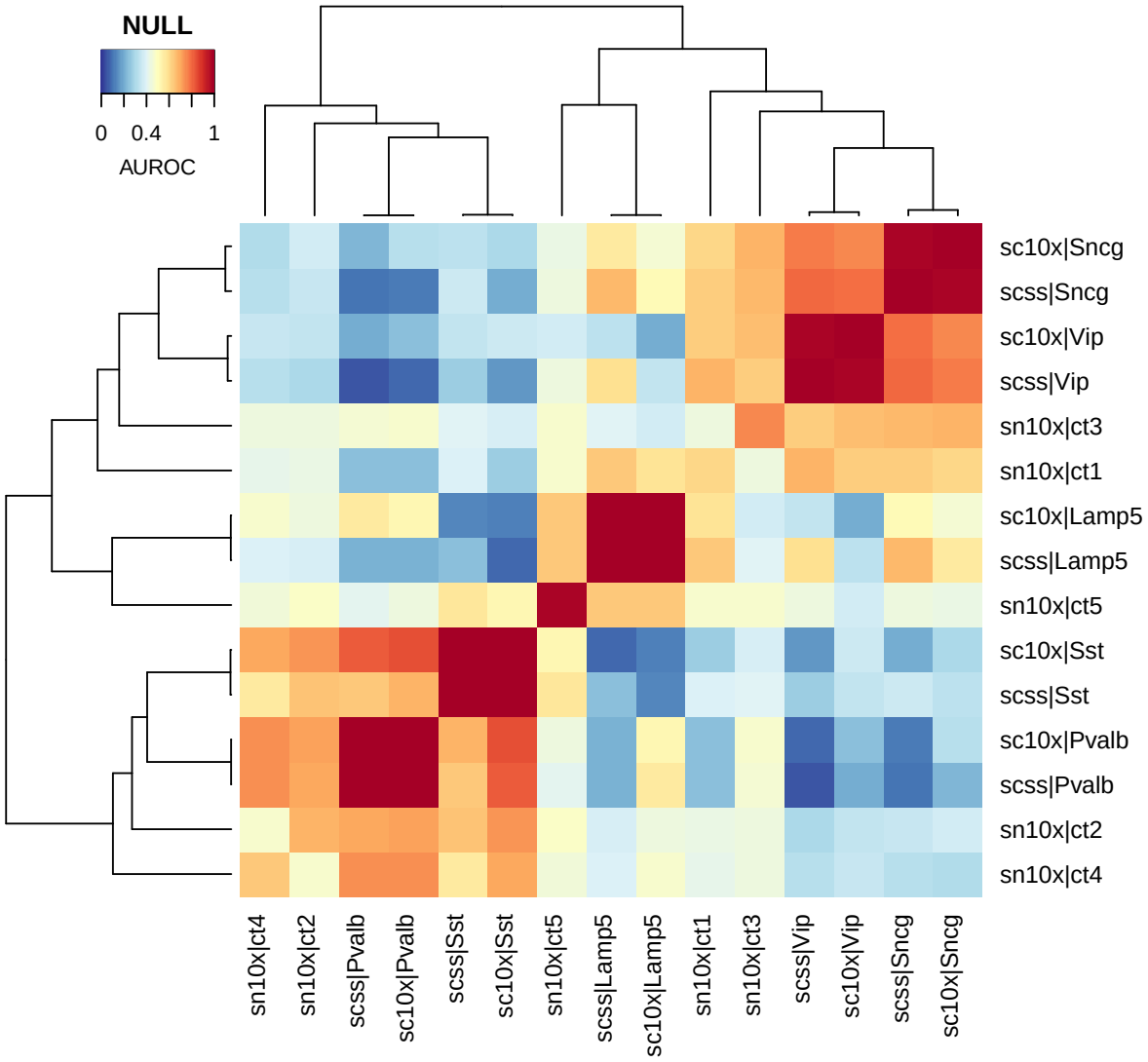
Study_ID|Celltype_1	Study_ID|Celltype_2	AUROC	Match_type
scss|Pvalb	sc10x|Pvalb	1.00	Reciprocal_top_hit
scss|Lamp5	sn10x|ct3	1.00	Reciprocal_top_hit
sn10x|ct1	sc10x|Pvalb	1.00	Reciprocal_top_hit
scss|Sst	sn10x|ct4	1.00	Reciprocal_top_hit
scss|Lamp5	sc10x|Lamp5	1.00	Reciprocal_top_hit
scss|Pvalb	sn10x|ct1	1.00	Reciprocal_top_hit
scss|Sst	sc10x|Sst	1.00	Reciprocal_top_hit
sn10x|ct3	sc10x|Lamp5	1.00	Reciprocal_top_hit
scss|Sncg	sn10x|ct5	0.99	Reciprocal_top_hit
sn10x|ct4	sc10x|Sst	0.99	Reciprocal_top_hit
scss|Vip	sc10x|Vip	0.99	Reciprocal_top_hit
scss|Sncg	sc10x|Sncg	0.98	Reciprocal_top_hit
scss|Vip	sn10x|ct2	0.98	Reciprocal_top_hit
sn10x|ct2	sc10x|Vip	0.98	Reciprocal_top_hit
sn10x|ct5	sc10x|Sncg	0.97	Reciprocal_top_hit

This function is particularly useful in the presence of a large number of cell types. This summary enables us to assign the missing cell types (as before with the matrix), but also access the confidence values associated with the calls (e.g., ct3 matches with Lamp5 with an AUROC of 1.00). A simple criterion to detect replicable cell types is to look for reciprocal matching cell types with AUROC > 0.9.

In comparison, let's run MetaNeighbor with permuted labels:

```
random_labels = all_data$label
is_sn = all_data$study_id == "sn10x"
random_labels[is_sn] = sample(random_labels[is_sn])
null_auroc = MetaNeighborUS(var_genes = hvg, dat = all_data, i ="counts",
                            study_id=all_data$study_id, cell_type = random_labels,
                            fast_version = TRUE)
```

```
plotHeatmap(null_auroc)
```




```
topHitsByStudy(auroc = null_auroc, threshold = 0.9)
```

Study_ID|Celltype_1	Study_ID|Celltype_2	AUROC	Match_type
scss|Pvalb	sc10x|Pvalb	1.00	Reciprocal_top_hit
scss|Lamp5	sc10x|Lamp5	1.00	Reciprocal_top_hit
scss|Sst	sc10x|Sst	1.00	Reciprocal_top_hit
scss|Vip	sc10x|Vip	0.99	Reciprocal_top_hit
scss|Sncg	sc10x|Sncg	0.98	Reciprocal_top_hit

None of the "hidden" cell types reach a replicability of 0.9, indicating that they don't match any of the cell types in the two other datasets, which suggests a problem in the clustering pipeline or the presence of biologically unrelated cell types.

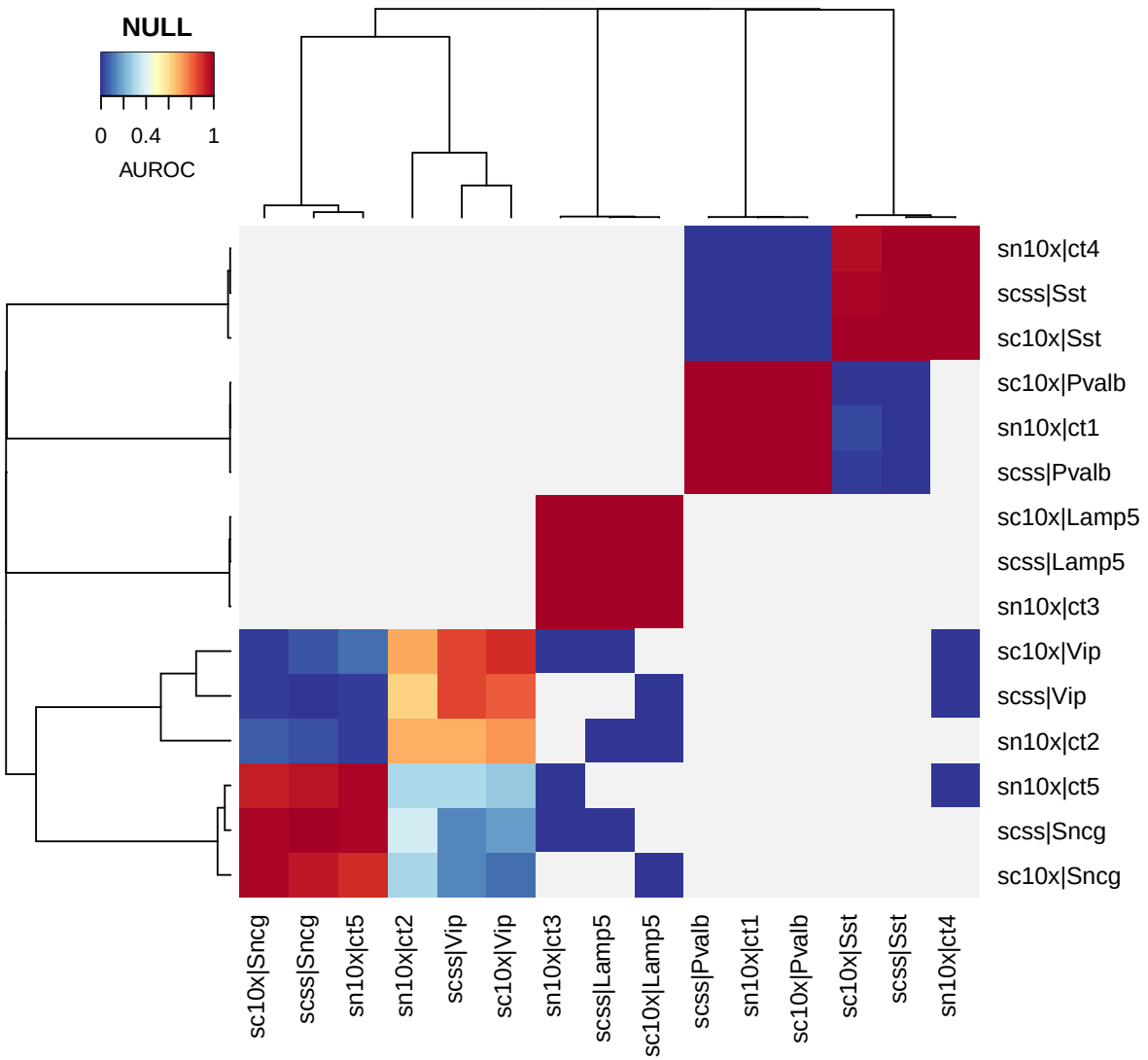
Rapidly identify and extract replicable cell types

In its original formulation, MetaNeighbor computes the similarity between all pairs of cell types across all datasets, which we refer to as "1-vs-all". However, one might be interested in the presence of 1:1 matches across datasets, which can be evaluated using "1-vs-best" AUROCs. In this mode, MetaNeighbor computes whether a cell type can be easily distinguished from its nearest neighbor (as opposed to any non-target cell), which is a stronger assessment of replicability. Let's try it out on our dataset:

```
auroc = MetaNeighborUS(var_genes = hvg, dat = all_data, i = "counts",
                       study_id = all_data$study_id, cell_type = all_data$label,
                       fast_version = TRUE, one_vs_best = TRUE, symmetric_output = FALSE)
```

All parameters are identical to the "1-vs-all" mode, except for the `one_vs_best` and `symmetric_output` parameters. Let's look at the AUROC similarities:

```
plotHeatmap(auroc)
```

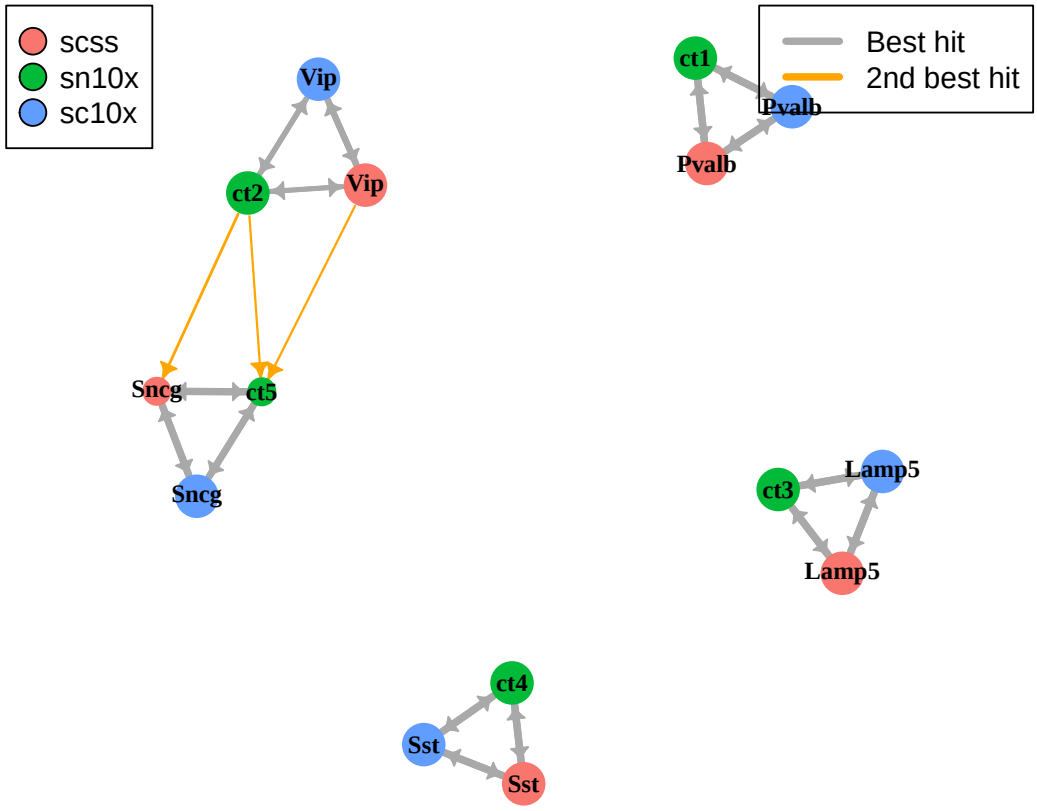


Most of similarity matrix is empty: each query cell type (column) is matched with its two best matches in each

target dataset, then MetaNeighbor quantifies how well the best match can be distinguished from the second best match. For example, the best matches of "sc10x|Sncg" (column 1) in the scss dataset are "scss|Sncg" and "scss|Vip". "sc10x|Sncg" can be very clearly matched with "scss|Sncg", as suggested by the high AUROC value (AUROC>0.9). Note that because this mode is more stringent, a threshold of AUROC>0.7 is already indicative of strong replicability.

Since the similarity matrix is sparse, we can more conveniently visualize results as a similarity graph between cell types:

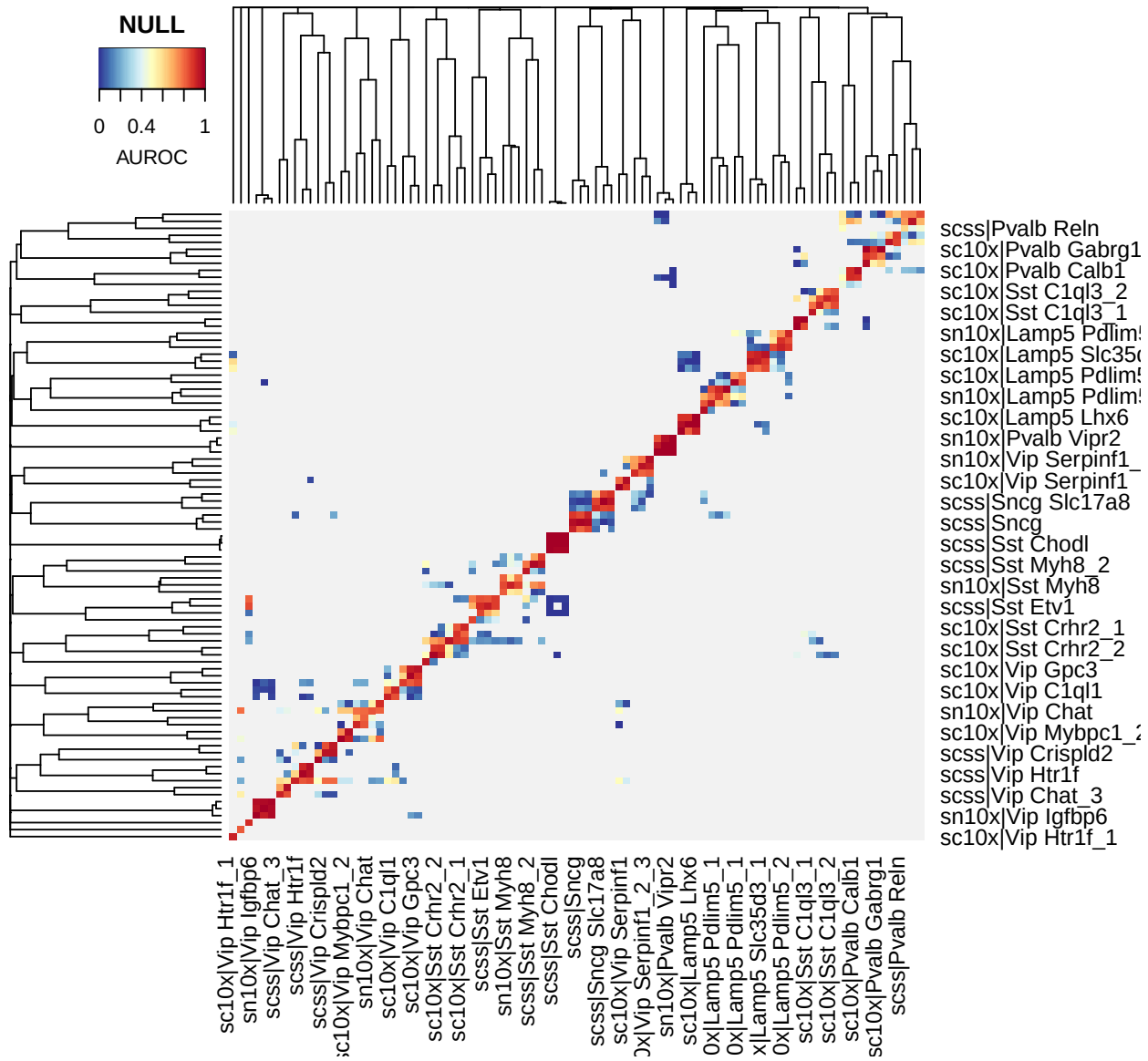
```
cluster_graph = makeClusterGraph(best_hits = auroc, low_threshold = 0.3)
plotClusterGraph(graph = cluster_graph, study_id = all_data$study_id,
                 cell_type = all_data$label, size_factor=4)
```



In this visualization, we can easily appreciate cases where a best hit cannot be perfectly separated from a second best hit. Gray edges indicate best hits, orange edges indicate second best hits. Here, we plot all second best hits corresponding to AUROCs for best hits that are lower than 0.7. We find that there are two cases with imperfect replicability, both implicating the Vip and Sncg cell types, which indicates that the cell type boundaries don't match exactly across datasets.

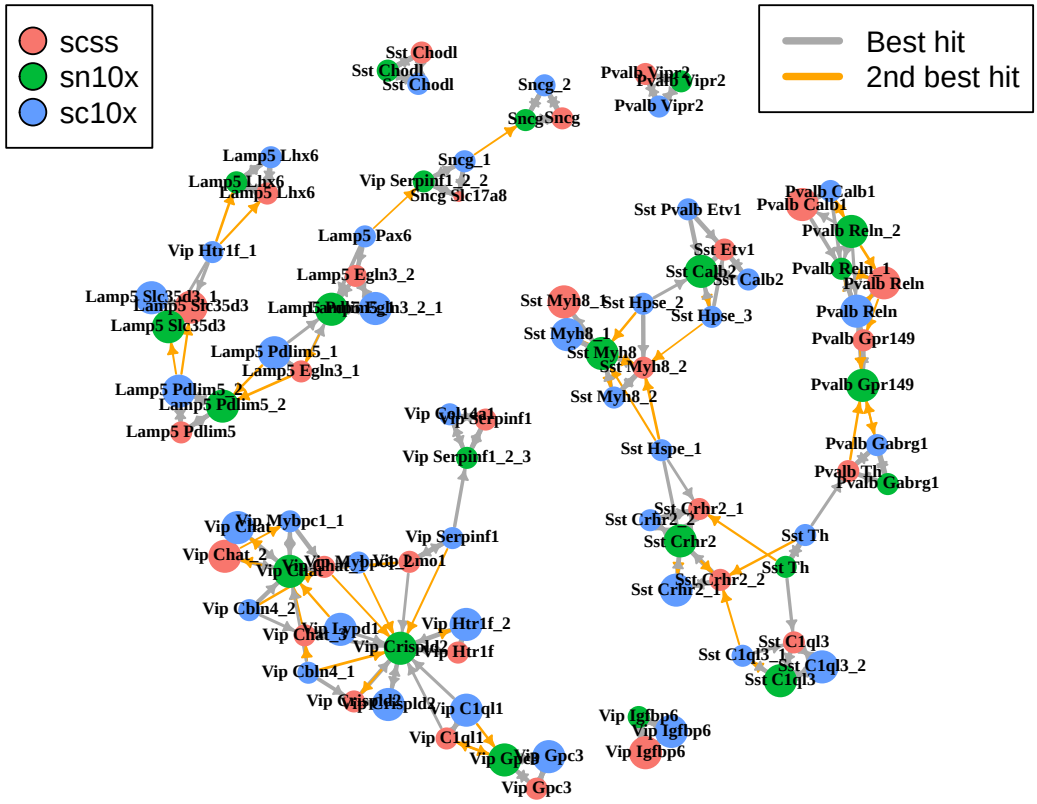
1-vs-best AUROCs are most useful in the presence of a large number of cell types, as it enables us to quickly identify 1:1 matches. In the previous example, there were only 5 cell types, so switching from 1-vs-all to 1-vs-best did not make a lot of difference. Let's try to run MetaNeighbor again, but this time we'll use the finer-grained clustering of the BICCN:

```
auroc = MetaNeighborUS(var_genes = hvg, dat = all_data, i = "counts",
                       study_id = all_data$study_id, cell_type = all_data$sublabel,
                       fast_version = TRUE, one_vs_best = TRUE, symmetric_output = FALSE)
plotHeatmap(auroc)
```



Because of the number of cell types, the similarity matrix is difficult to interpret. Let's switch to the graph representation:

```
cluster_graph = makeClusterGraph(best_hits = auroc, low_threshold = 0.3)
plotClusterGraph(graph = cluster_graph, study_id = all_data$study_id,
                 cell_type = all_data$sublabel, size_factor=3)
```



The graph suggests a mix of perfectly resolved cell types (cliues of 1:1:1 matches, like Vip IGfbp6 or Sst Chodl) and cell types with inconsistent resolution (stretches that contain multiple cell types, n:m matches). 1:1:1 matches are particularly interesting cell types, because they were found independently in all datasets, which increases the likelihood for a common biological basis rather than clustering of noisy components of the transcriptome. We can extract those as meta-clusters and rank them according to AUROC:

```
mclusters = extractMetaClusters(auroc, threshold = 0.7)
mcsummary = scoreMetaClusters(mclusters, auroc)
mcsummary
```

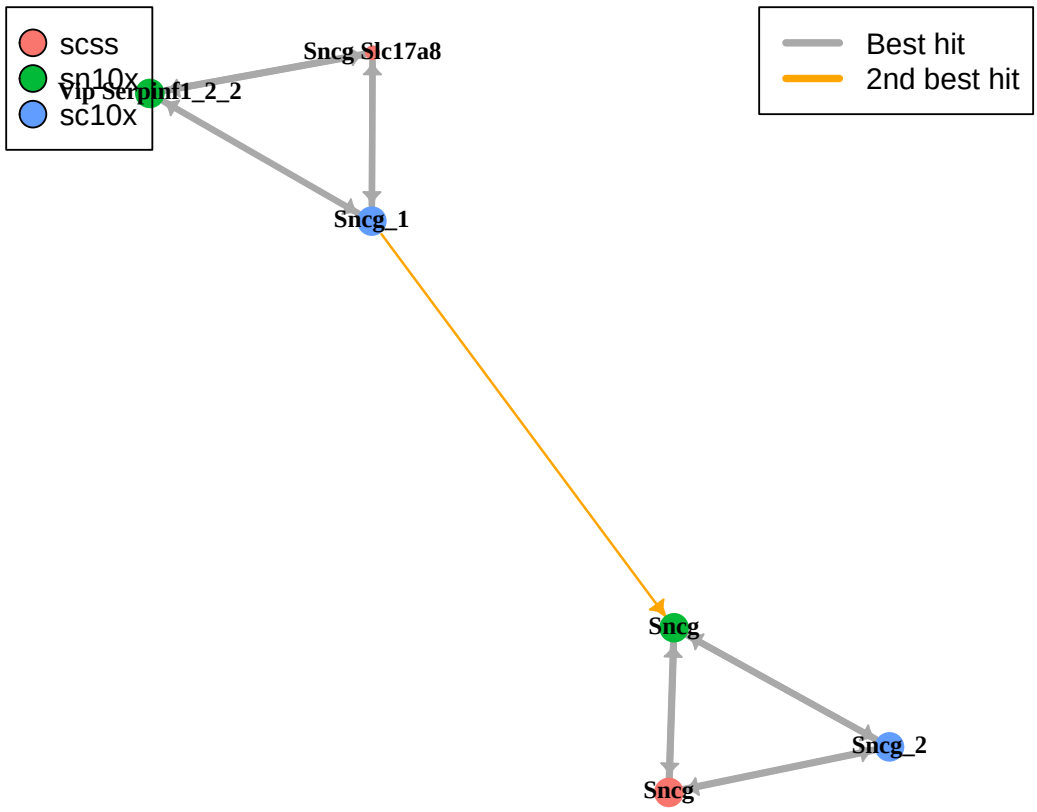
metacluster	clusters	n_studies	score
metacl_1	scss|Sst Chodl; sn10x|Sst Chodl; sc10x|Sst Chodl	3	1.00
metacl_2	scss|Vip Igfbp6; sn10x|Vip Igfbp6; sc10x|Vip Igfbp6	3	0.97
metacl_3	scss|Pvalb Vipr2; sn10x|Pvalb Vipr2; sc10x|Pvalb Vipr2	3	0.97
metacl_4	scss|Lamp5 Lhx6; sn10x|Lamp5 Lhx6; sc10x|Lamp5 Lhx6	3	0.94
metacl_5	scss|Sncg; sn10x|Sncg; sc10x|Sncg__2	3	0.91
metacl_6	scss|Lamp5 Slc35d3; sn10x|Lamp5 Slc35d3; sc10x|Lamp5 Slc35d3__1	3	0.90

meta_cluster	cluster	n_studies	score
meta_sncluster6	sn10x|Sncg Slc17a8; sn10x|Vip Serpinf1_2_2; sc10x|Sncg_1	3	0.88
meta_sncluster21	sn10x|Vip Gpc3; sn10x|Vip Gpc3; sc10x|Vip Gpc3	3	0.85
meta_sncluster25	sn10x|Vip Serpinf1; sn10x|Vip Serpinf1_2_3; sc10x|Vip Col14a1	3	0.85
meta_sncluster2	sn10x|Lamp5 Egln3_2; sn10x|Lamp5 Pdlim5_1; sc10x|Lamp5 Egln3_2_1	3	0.84
meta_sncluster15	sn10x|Sst Etv1; sn10x|Sst Calb2; sc10x|Sst Calb2	3	0.83
meta_sncluster11	sn10x|Sst C1ql3; sn10x|Sst C1ql3; sc10x|Sst C1ql3_2	3	0.82
meta_sncluster26	sn10x|Sst Th; sc10x|Sst Th	2	0.96
meta_sncluster22	sn10x|Vip Htr1f; sc10x|Vip Htr1f_2	2	0.95
meta_sncluster13	sn10x|Sst Crhr2_1; sc10x|Sst Crhr2_2	2	0.92
meta_sncluster20	sn10x|Vip Crispld2; sc10x|Vip Crispld2	2	0.92
meta_sncluster24	sn10x|Vip Serpinf1; sc10x|Vip Serpinf1	2	0.92
meta_sncluster3	sn10x|Pvalb Calb1; sc10x|Pvalb Calb1	2	0.91
meta_sncluster18	sn10x|Vip Mybpc1_2; sc10x|Vip Mybpc1_2	2	0.91
meta_sncluster16	sn10x|Sst Myh8_2; sc10x|Sst Myh8_2	2	0.88
meta_sncluster7	sn10x|Pvalb Th; sc10x|Pvalb Gabrg1	2	0.88
meta_sncluster17	sn10x|Vip C1ql1; sc10x|Vip C1ql1	2	0.87
meta_sncluster19	sn10x|Vip Lypd1; sc10x|Vip Lypd1	2	0.86
meta_sncluster14	sn10x|Sst Crhr2_2; sc10x|Sst Crhr2_1	2	0.86
meta_sncluster4	sn10x|Lamp5 Pdlim5; sc10x|Lamp5 Pdlim5_2	2	0.85
meta_sncluster1	sn10x|Lamp5 Egln3_1; sc10x|Lamp5 Pdlim5_1	2	0.82
outliers	scss|Pvalb Gpr149; scss|Pvalb Reln; scss|Sst Myh8_1; scss|Vip Chat_2; sn10x|Pvalb Gabrg1; sn10x|Pvalb Gpr149; sn10x|Pvalb Reln_1; sn10x|Pvalb Reln_2; sn10x|Sst Crhr2; sn10x|Sst Myh8; sn10x|Vip Chat; sn10x|Vip Crispld2; sc10x|Lamp5 Pax6; sc10x|Lamp5 Pdlim5_2; sc10x|Pvalb Reln; sc10x|Sst C1ql3_1; sc10x|Sst Hpse_2; sc10x|Sst Hpse_3; sc10x|Sst Hspe_1; sc10x|Sst Myh8_1; sc10x|Sst Pvalb Etv1; sc10x|Vip Cbln4_1; sc10x|Vip Cbln4_2; sc10x|Vip Chat; sc10x|Vip Htr1f_1; sc10x|Vip Mybpc1_1	1	NA

MetaNeighbor finds support for 26 meta-clusters, 12 of which are perfect matches across all 3 datasets. Noticeably, cell types with previous characterization such as Sst Chodl (long-projecting interneurons), Pvalb Vipr2 (Chandelier cells), Lamp5 Lhx6 (neurogliaform cells) all score extremely high in terms of replicability (AUROC>0.9).

It is possible to focus on a subpart of the graph. For example, one of the top meta-clusters contains two Sncg cell types and a Vip cell types. We can pick one of these cell types then plot its immediate neighborhood:

```
coi = "sn10x|Vip Serpinf1_2_2"
coi = extendClusterSet(cluster_graph, initial_set=coi, max_neighbor_distance=3)
subgraph = subsetClusterGraph(cluster_graph, coi)
plotClusterGraph(subgraph, all_data$study_id, all_data$sublabel, size_factor=4)
```



The graph neighborhood only contains Sncg types, which suggests that the Vip-labeled type is in fact an Sncg type.

Functional characterization of replicability with supervised MetaNeighbor

We now switch to the supervised version of MetaNeighbor, which can be used screen functional gene sets that contribute cell type replicability. The supervised version also runs in two steps (like the unsupervised version), but there are two key differences: (a) highly variable genes are replaces by functionally characterized sets of genes (like Gene Ontology gene sets), (b) cell type labels *must* match across datasets. Typically, unsupervised MetaNeighbor is run first to identify replicable cell types, then supervised MetaNeighbor is run to identify gene sets contributing to replicability.

The functional gene sets we'll consider below are Gene Ontology gene sets containing between 20 and 50 genes. This size of gene sets ensures that there are enough genes to obtain meaningfull correlations across

cells and that the functions represented by the gene sets remain relatively specific:

```
go_sets = readRDS("go_mouse.rds")

known_genes = rownames(all_data)
go_sets = lapply(go_sets, function(gene_set) { gene_set[gene_set %in% known_genes] })
min_size = 20
max_size = 50
go_set_size = sapply(go_sets, length)
go_sets = go_sets[go_set_size >= min_size & go_set_size <= max_size]
length(go_sets)
```

```
## [1] 2507
```

Before running supervised MetaNeighbor, we need to make sure that all cell type labels match across studies. The names for "scss" and "sc10x" already match, but we need to define new names for the "sn10x" dataset. We create a new metadata column and use the unsupervised analysis above to match cell types:

```
all_data$new_label = all_data$label
all_data$new_label[all_data$new_label == "ct1"] = "Pvalb"
all_data$new_label[all_data$new_label == "ct2"] = "Vip"
all_data$new_label[all_data$new_label == "ct3"] = "Lamp5"
all_data$new_label[all_data$new_label == "ct4"] = "Sst"
all_data$new_label[all_data$new_label == "ct5"] = "Sncg"
table(all_data$new_label)
```

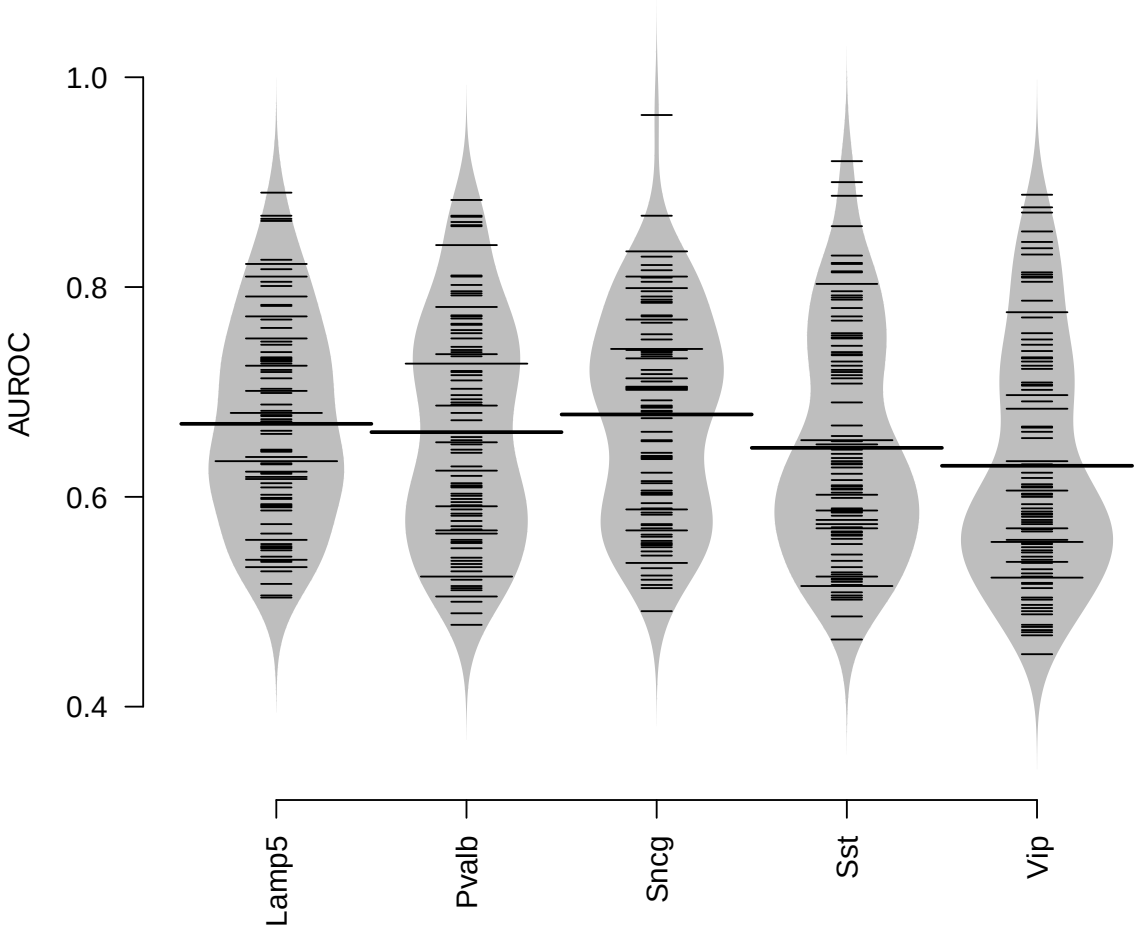
```
##
## Lamp5 Pvalb  Sncg   Sst   Vip
##  2686  2585   390  3089  3342
```

We now run supervised MetaNeighbor, with essentially the same parameter as `MetaNeighborUS`:

```
system.time({
aurocs = MetaNeighbor(dat = all_data, i = "counts",
                      experiment_labels = all_data$study_id,
                      celltype_labels = all_data$new_label,
                      genesets = go_sets,
                      fast_version = TRUE, bplot = FALSE, batch_size = 50)
})
```

Because `MetaNeighbor` is roughly equivalent to running `MetaNeighborUS` on each gene set, this step can last relatively long for longer lists of gene sets and larger datasets.

```
plotBPlot(head(aurocs, 100))
```



Most gene sets contribute lowly to replicability (AUROC~0.65), so we'll focus on gene sets with high contributions (AUROC>0.8). We're mostly interested in gene sets that have high contributions to all cell types, so let's summarize and rank the results in a new data frame:

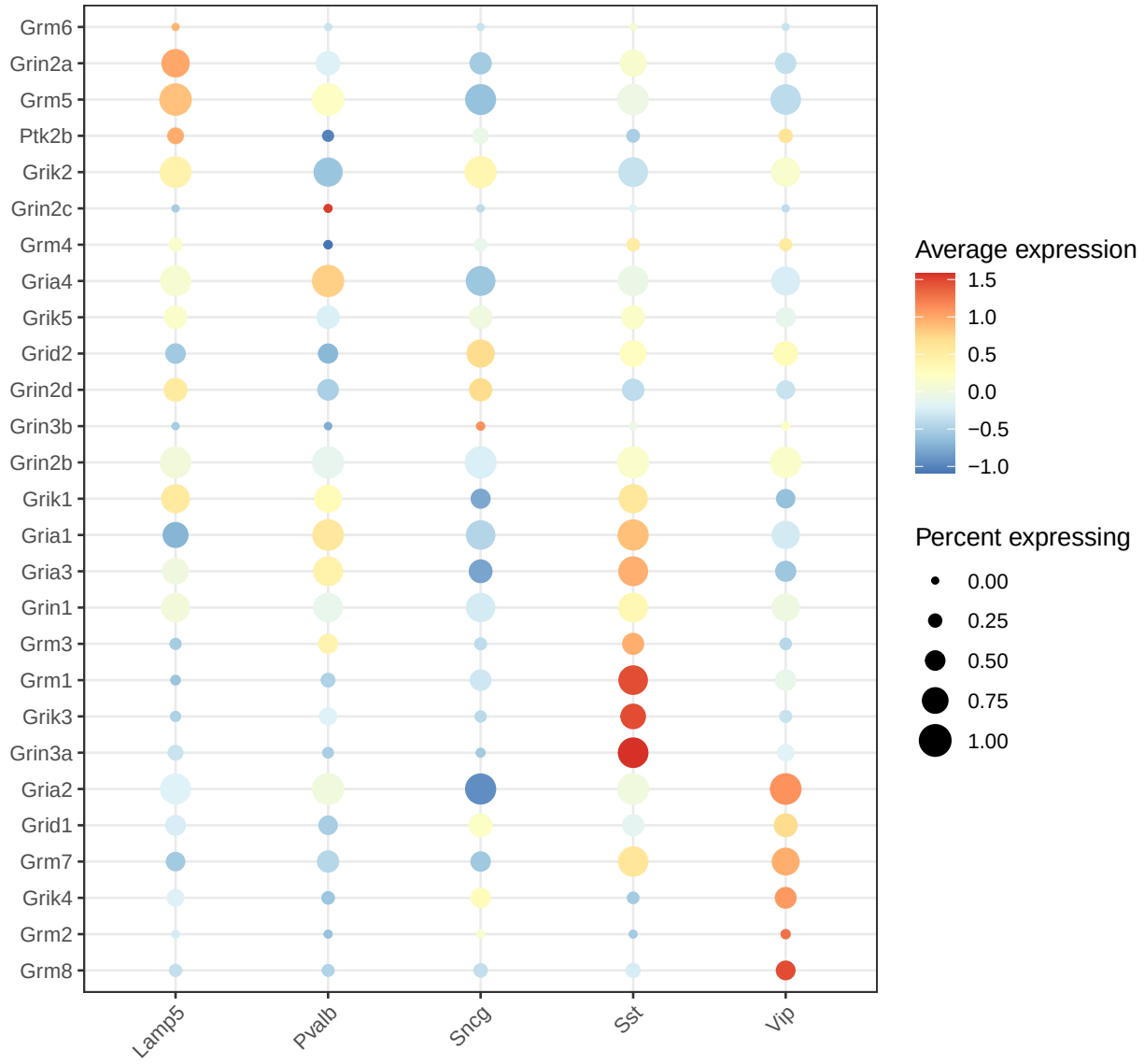
```
gs_size = sapply(go_sets, length)
aurocs_df = data.frame(go_term = rownames(aurocs), aurocs)
aurocs_df$average = rowMeans(aurocs)
aurocs_df$n_genes = gs_size[rownames(aurocs)]
head(aurocs_df[order(aurocs_df$average, decreasing = TRUE),],10)
```

go_term	Lamp5	Pvalb	Sncg	Sst	Vip	average	n_genes
GO:0008066%7Cglutamate receptor activity|MF	0.93	0.90	0.93	0.97	0.94	0.93	27
GO:0048846%7Caxon extension involved in axon guidance|BP	0.93	0.96	0.93	0.93	0.88	0.93	37
GO:1902284%7Cneuron projection extension involved in neuron projection guidance|BP	0.93	0.96	0.93	0.93	0.88	0.93	37

go_term	Lamp5	Pvalb	Sncg	Sst	Vip	average	n_genes
GO:0005245%7Cvoltage-gated calcium channel activity|MF	0.95	0.96	0.95	0.90	0.88	0.93	49
GO:1902667%7Cregulation of axon guidance|BP	0.93	0.96	0.94	0.93	0.87	0.93	45
GO:0032590%7Cdendrite membrane|CC	0.97	0.95	0.94	0.93	0.83	0.92	47
GO:0048841%7Cregulation of axon extension involved in axon guidance|BP	0.92	0.92	0.93	0.94	0.87	0.91	33
GO:0032228%7Cregulation of synaptic transmission, GABAergic|BP	0.89	0.95	0.89	0.93	0.89	0.91	34
GO:1902668%7Cnegative regulation of axon guidance|BP	0.90	0.95	0.92	0.92	0.86	0.91	28
GO:0021988%7Colfactory lobe development|BP	0.93	0.91	0.87	0.93	0.90	0.91	39

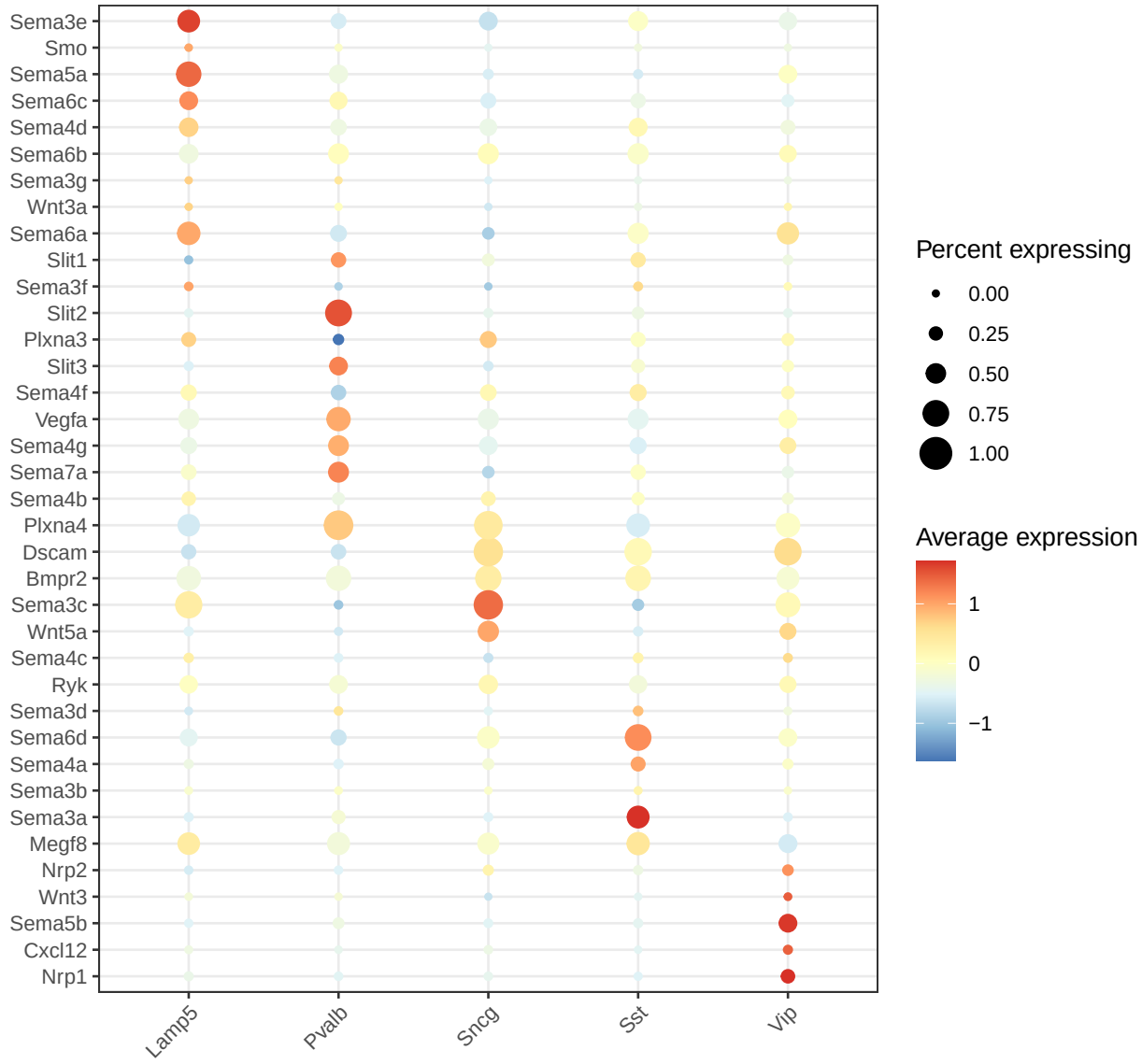
The top gene sets have extremely high contribution to replicability (average AUROC > 0.9) and, unsurprisingly, are all related to neuronal functions. To understand how these gene sets contribute to making cell types distinct, we can plot the individual genes from high scoring gene sets:

```
plotDotPlot(dat = all_data,
            experiment_labels = all_data$study_id,
            celltype_labels = all_data$new_label,
            gene_set = go_sets[["GO:0008066|glutamate receptor activity|MF"]])
```



The different types of inhibitory neurons tend to use different variants of glutamatergic receptor subunits (e.g., Grin2a, Grm5 for Lamp5).

```
plotDotPlot(dat = all_data,
            experiment_labels = all_data$study_id,
            celltype_labels = all_data$new_label,
            gene_set = go_sets[["GO:0048846|axon extension involved in axon guidance|BP"]])
```



The different types of inhibitory neurons tend to use different variants of semaphorins (e.g., Sema3e, Sema5a for Lamp5), which are involved in axon guidance.