MetaNeighbor : a method to rapidly assess cell type identity using both functional and random gene sets

Megan Crow, Stephan Fischer, Sara Ballouz, Manthan Shah, Jesse Gillis

2020-04-28

Contents

1 Introduction

The purpose of this method is to measure the similarity of cells across single cell RNA-sequencing (scRNA-seq) datasets by sampling from both random and functionally defined gene sets. MetaNeighbor works on the basis that cells of the same type should have more similar gene expression profiles than cells of different types.

In other words, when we compare the expression profile between T cells and hepatocytes for a given gene set, we should see higher correlations among all T cells than we do between T cells and hepatocytes. This is illustrated in the schematic below:

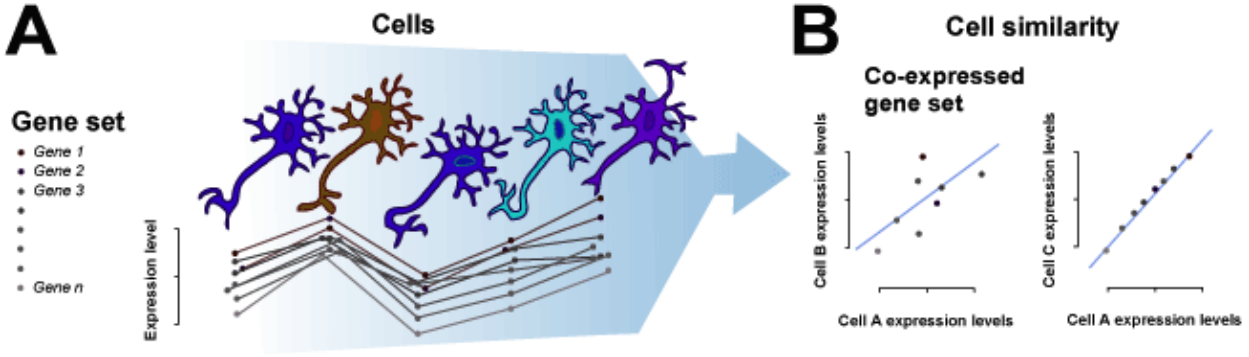


Figure 1. A. Relationship between gene set expression and cell type B. Cell similarity within and across cell types

In our approach, this is formalized through neighbor voting based on cell-cell similarities, which will be described in detail in the Methods section. In short, MetaNeighbor takes four inputs: a gene-by-sample expression matrix ("data"), a set of labels indicating each sample's dataset of origin ("experiment labels"), a set of labels indicating each sample's cell type ("cell type labels") and a set of genes ("gene sets"). The output is a performance vector ("AUROC scores"), which is the mean area under the receiver operator characteristic curve (AUROC) for the given task. This score reflects our ability to rank cells of the same known type higher than those of other types within a dataset, and can be interpreted as the probability that we will be correct about making a binary classification for a given cell (e.g. neuron vs. non-neuronal cell). An AUROC score of 0.5 means that we have performed as well as if we had randomly guessed the cell's identity.

This is a fully supervised analysis, and requires knowledge of the corresponding cell types across datasets. However, we have also used some heuristic measures to identify cell types across datasets when labels may be ambiguous or uninformative. We will walk through this exploratory analysis in Part 2 of the vignette.

2 Data type requirements

For MetaNeighbor to run, the input data should be a SummarizedExperiment object (SEO) with the following considerations:

1. The gene-by-sample matrix should be an assay of SEO.
2. Gene sets of interest should be provided as a list of vectors.
3. Additional data should have following vectors:
    i. `sample_id`: A character vector (length equal to the number of samples) containing a unique identifier for each sample.
    ii. `study_id`: A character vector (length equal to the number of samples) that indicates the source of each sample (ex."GSE60361" = Zeisel et al, "GSE71585" = Tasic et al, as in mn_data).
    iii. `cell_type`: A character vector (length equal to the number of samples) that indicates the cell type of each sample.
4. `cell_labels` should be provided as a sample-by-cell label matrix. This should be a binary (0,1) matrix where 1 indicates cell type membership and 0 indicates non-membership.

Additional requirements to be noted:

1. It is critical that genes within gene sets match the gene names of the expression matrix.
2. Gene sets should contain more than one gene.
3. The row names of the `cell_labels` object should match the column names of the expression matrix.

3 System requirements/estimated run times

Because there is a ranking step, the code is not ideal for scaling in the R environment as is. The major factor is the total number of samples. Speed-ups are possible with parallelization and installing libraries such as MRAN (https://mran.revolutionanalytics.com/). We also propose an approximate version of MetaNeighbor that skips the ranking step, effectively improving scalability by using less memory and running faster.

Laptop (OSX 10.10.4, 1.6 GHz, 8GB RAM, R 3.3.2, Rstudio 0.99.446) for the default version.

Desktop (Ubuntu 18.04.1 in an Oracle VM, 4 x 3.6 GHz, 10.9GB RAM, R 3.4.4) for the low-memory version.

Experiments	Cell types	Samples	Gene sets	Default: Time (s)	Low-memory: Time (s)
2	1	100	10	0.1	.
2	10	100	10	0.5	0.1
2	10	100	100	1.7	0.6
2	10	100	1000	17.5	6.2
2	1	1000	10	9	.
10	1	1000	10	9	.
2	10	1000	10	12	0.7
2	10	1000	100	93	6.4
2	10	1000	1000	979	63
2	10	10000	10	3653	10
2	10	10000	100	.	91
2	10	10000	1000	.	910

(a . indicates that we did not measure this combination of parameters)

4 Installation

nstall the MetaNeighbor package from Bioconductor by running the following command:

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

The development version of the package can be downloaded using following commands:

```
if (!require('devtools')) install.packages('devtools', quiet=TRUE)
devtools::install_git('https://github.com/gillislab/MetaNeighbor')
```

5 Methods

MetaNeighbor runs as follows: first, we build a network of rank correlations between all cells for a gene set. All values in the network are then re-ranked and standardized to lie between 0-1. Next, the neighbor voting predictor produces a weighted matrix of predicted labels by performing matrix multiplication between the network and the binary vector indicating cell type membership, then dividing each element by the null predictor (i.e., node degree). That is, each cell is given a score equal to the fraction of its neighbors (including itself), which are part of a given cell type. For cross-validation, we permute through all possible combinations of leave-one-dataset-out cross-validation, and we report how well we can recover cells of the same type as area under the receiver operator characteristic curve (AUROC). This is repeated for all folds of cross-validation, and the mean AUROC across folds is reported.

5.1 Part 1: Supervised MetaNeighbor

5.1.1 Quick start

To run through the analysis and plot results, load the package and run the following commands:

```
library(MetaNeighbor)
library(SummarizedExperiment)
data(mn_data)
data(GOmouse)
AUROC_scores = MetaNeighbor(dat = mn_data,
                            experiment_labels = as.numeric(factor(mn_data$study_id)),
                            celltype_labels = metadata(colData(mn_data))[["cell_labels"]],
                            genesets = GOmouse,
                            bplot = TRUE)
```

5.1.2 More detail

We have provided sample data and sample gene sets within the MetaNeighbor package. In this sample data, we have included the cortical interneurons from two public datasets, GSE60361 and GSE71585 (RPKM). A subset of ~3000 genes and 10 genesets have been included for demonstration. For this example, we will be testing how well we can identify the Sst Chodl subtype (Sst-Chodl from GSE71585 and Int1 from GSE60361) and the Smad3 subtype (Smad3 from GSE71585 and Int14 from GSE60361), relative to all other interneurons within their respective experiments.

MetaNeighbor requires a SummarizedExperimentObject as input, formatted as specified above.

There are two outputs of the method:

1. A matrix of AUROC scores representing the mean for each gene set tested for each celltype
2. A beanplot displaying density of AUROC scores for each cell type (by default the plot will be displayed and can be turned off by setting the argument **bplot=FALSE**)

5.1.2.1 Load package and data

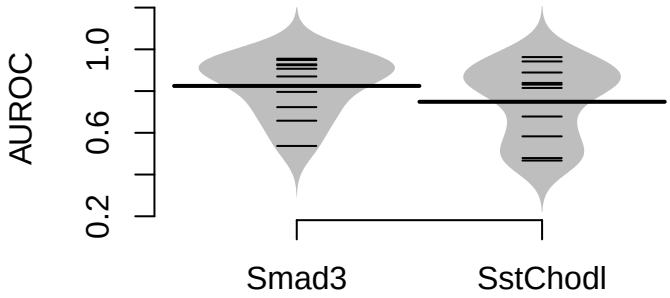
To run through the analysis, run the following commands:

```
library(MetaNeighbor)
library(SummarizedExperiment)
data(mn_data)
data(GOmouse)
```

5.1.2.2 Run MetaNeighbor and plot results

As MetaNeighbor runs, it outputs the name of the gene set that is being evaluated. When all gene sets have been tested, MetaNeighbor will return a gene set-by-cell type matrix of AUROC scores. A smoothed distribution of scores for each cell type will be plotted by default (turn off plotting by setting **bplot= FALSE**). Short horizontal lines inside the shape indicate AUROC values for individual gene sets, and the large horizontal line represents the mean.

```
AUROC_scores = MetaNeighbor(dat = mn_data,
                            experiment_labels = as.numeric(factor(mn_data$study_id)),
                            celltype_labels = metadata(colData(mn_data))[["cell_labels"]],
                            genesets = GOmouse,
                            bplot = TRUE)
```




```
head(AUROC_scores)
```

```
##            SstChodl Smad3
## GO:0016853    0.678 0.658
## GO:0005615    0.963 0.949
## GO:0005768    0.815 0.870
## GO:0007067    0.583 0.723
## GO:0065003    0.839 0.928
## GO:0042592    0.889 0.955
```

AUROC scores greater than 0.5 indicate improvement over randomly guessing the identity of the cell type of interest.

5.2 Part 2: MetaNeighbor for Data Exploration

5.2.1 Quick start

To run through the analysis and plot results run the following commands:

```
library(MetaNeighbor)
data(mn_data)
var_genes = variableGenes(dat = mn_data, exp_labels = mn_data$study_id)
celltype_NV = MetaNeighborUS(var_genes = var_genes,
                             dat = mn_data,
                             study_id = mn_data$study_id,
                             cell_type = mn_data$cell_type)
top_hits = topHits(cell_NV = celltype_NV,
                   dat = mn_data,
                   study_id = mn_data$study_id,
                   cell_type = mn_data$cell_type,
                   threshold = 0.9)
top_hits
cols = rev(colorRampPalette(RColorBrewer::brewer.pal(11,"RdYlBu"))(100))
breaks = seq(0, 1, length=101)
gplots::heatmap.2(celltype_NV,
                  margins=c(8,8),
                  keysize=1,
```

```
                    key.xlab="AUROC",
                    key.title="NULL",
                    trace = "none",
                    density.info = "none",
                    col = cols,
                    breaks = breaks,
                    offsetRow=0.1,
                    offsetCol=0.1,
                    cexRow = 0.7,
                    cexCol = 0.7)
```

5.2.2 More detail

While ideally we would like to perform supervised analyses to investigate cell type identity, in some cases it is difficult to know how cell type labels compare across datasets. For this situation, we came up with a heuristic to allow researchers to make an educated guess about overlaps without requiring in-depth knowledge of marker genes. This was based on our observation that large, high variance gene sets tended to provide improved scores for known cell types.

Exploratory Metaneighbor requires an input SEO as specified above, as well as a vector containing a set of variable genes (`var_genes`). The function will use the set of variable genes to create a cell-cell similarity network.

The output of the function is a cell type-by-cell type mean AUROC matrix, which is built by treating each pair of cell types as testing and training data for MetaNeighbor, then taking the average AUROC for each pair (NB AUROC scores across testing and training folds will not be identical because each test cell type is scored out of its own dataset, and differences in dataset heterogeneity influence scores). When comparing datasets that contained similar cell types, we found that cell types that were the best hit for one another ("reciprocal top hits"), and cell types with scores >0.9 tended to be good candidates for downstream tests of cell type identity using Supervised MetaNeighbor. This rule will not hold when experiments contain wholly different cell types (e.g., comparing brain to pancreas will likely yield some spurious overlaps), or when datasets are very unbalanced with respect to one another.

5.2.2.1 Load package and data We have provided sample data and source code here.To begin the analysis, simply load the package from the above link into your R session.

```
library(MetaNeighbor)
data(mn_data)
```

5.2.2.2 Identify a highly variable gene set To begin, we will use the function variableGenes, which picks the top quartile of variable genes across all but the top decile of expression bins for each dataset, then provides the intersect across datasets as the output.

```
var_genes = variableGenes(dat = mn_data, exp_labels = mn_data$study_id)
head(var_genes)
```

```
## [1] "1110017D15Rik" "1190002N15Rik" "3110043O21Rik" "Aacs"
## [5] "Abcb10"        "Abcb6"
```

```
length(var_genes)
```

```
## [1] 331
```

In this case, we return a set of 331 highly variable genes. AUROC scores depend on both gene set size and variance. If the size of the returned set is small (<200 genes) the suggested AUROC cut-off of >0.9 may not be applicable, and it may be helpful to increase the gene set size. This may be done by taking a majority rule on genes included in the highly variable sets of each dataset in the analysis (i.e., include a gene if it is

highly variable in >50% of datasets) . This strategy is likely to be required with an increasing number of datasets included. However, we note that if few genes are returned when using a small number of datasets (2-3), this may indicate that the datasets have different cell type compositions, or have very different gene coverage. Under these circumstances, we do not recommend the use of MetaNeighbor.

5.2.2.3 Run MetaNeighbor for data exploration

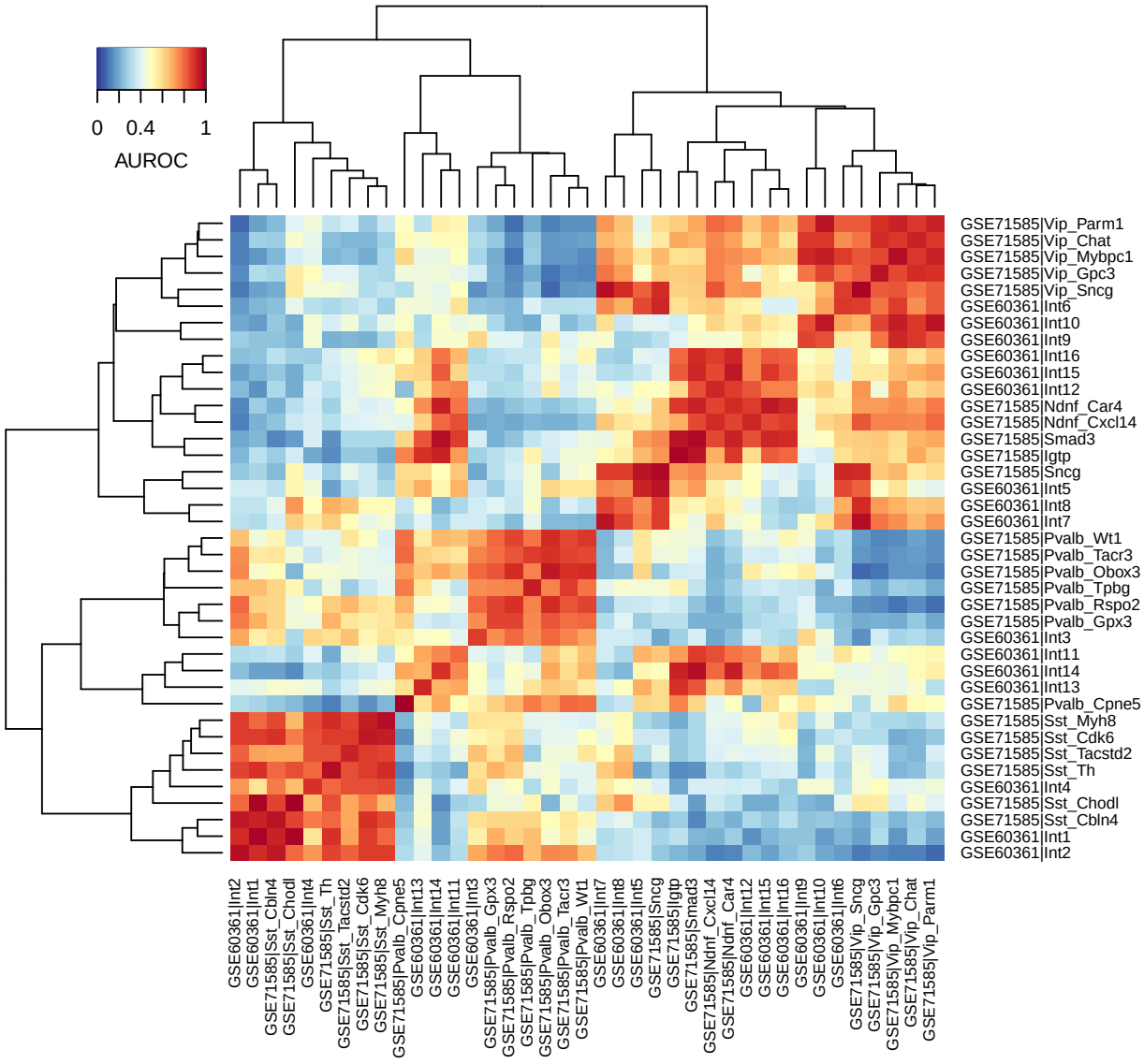
Once we have a set of highly variable genes, we can simply run an exploratory version of MetaNeighbor using the function:

```
celltype_NV = MetaNeighborUS(var_genes = var_genes,
                             dat = mn_data,
                             study_id = mn_data$study_id,
                             cell_type = mn_data$cell_type)
```

5.2.2.4 Plot results

Results can be plotted as follows:

```
cols = rev(colorRampPalette(RColorBrewer::brewer.pal(11,"RdYlBu"))(100))
breaks = seq(0, 1, length=101)
gplots::heatmap.2(celltype_NV,
                  margins=c(8,8),
                  keysize=1,
                  key.xlab="AUROC",
                  key.title="NULL",
                  trace = "none",
                  density.info = "none",
                  col = cols,
                  breaks = breaks,
                  offsetRow=0.1,
                  offsetCol=0.1,
                  cexRow = 0.7,
                  cexCol = 0.7)
```



This plot shows the AUROC scores between each testing and training pair. Red indicates a higher score and blue indicates a lower score. Note that the diagonal is not equal to one. This is because of the scoring system: cell types that are 'promiscuous' (i.e., have broad similarity to many types) will tend to have higher node degrees in the network. Differences in degree across cell types will affect scores as predictions are standardized by this factor. Within-dataset scores are shown for visualization purposes only, and should not be used for replicability inference.

5.2.2.5 Identify reciprocal top hits and high scoring cell type pairs

To find reciprocal top hits and those with AUROC>0.9 we use the following code:

```
top_hits = topHits(cell_NV = celltype_NV,
                   dat = mn_data,
                   study_id = mn_data$study_id,
                   cell_type = mn_data$cell_type,
                   threshold = 0.9)
top_hits
```

```
##    Study_ID|Celltype_1  Study_ID|Celltype_2 Mean_AUROC         Match_type
```

```
##  1        GSE60361|Int1   GSE71585|Sst_Chodl       0.99 Reciprocal_top_hit
##  2        GSE60361|Int7    GSE71585|Vip_Sncg       0.97 Reciprocal_top_hit
##  3       GSE60361|Int14       GSE71585|Smad3       0.97 Reciprocal_top_hit
##  4        GSE60361|Int5        GSE71585|Sncg       0.96 Reciprocal_top_hit
##  5       GSE60361|Int10   GSE71585|Vip_Parm1       0.95 Reciprocal_top_hit
##  6        GSE60361|Int2   GSE71585|Sst_Cbln4       0.95 Reciprocal_top_hit
##  7       GSE60361|Int15   GSE71585|Ndnf_Car4       0.94 Reciprocal_top_hit
##  8       GSE60361|Int10  GSE71585|Vip_Mybpc1       0.93          Above_0.9
##  9       GSE60361|Int14        GSE71585|Igtp       0.92          Above_0.9
## 10        GSE71585|Sncg        GSE60361|Int6       0.92          Above_0.9
## 11  GSE71585|Ndnf_Car4       GSE60361|Int16       0.92          Above_0.9
## 12      GSE60361|Int12 GSE71585|Ndnf_Cxcl14       0.91 Reciprocal_top_hit
## 13 GSE71585|Vip_Mybpc1        GSE60361|Int9       0.91          Above_0.9
## 14   GSE71585|Vip_Sncg        GSE60361|Int8       0.91          Above_0.9
## 15       GSE60361|Int3 GSE71585|Pvalb_Obox3       0.81 Reciprocal_top_hit
```

These top hits can then be used for supervised analysis, making putative cell type labels for each unique grouping (see Part 1).

5.3 Part 3: low-memory version of MetaNeighbor for large datasets

MetaNeighbor's voting algorithm relies on a cell-cell correlation network. This procedure becomes very memory-intensive and time consuming when it is applied to datasets that contain a large number of samples (>10K). We propose an approximative version of MetaNeighbor that does not explicitly compute the cell-cell correlation network, resulting in significant improvements in memory usage and run times.

The low-memory (fast) version is used by passing the flag `fast_version = TRUE` to either MetaNeighbor or MetaNeighborUS. Other parameters are unchanged: switching from the exact to the fast version is very simple. If your dataset is particularly large, we encourage you to use SingleCellExperiment objects, a subclass of SummarizedExperiment that is able to store data in sparse representations. We also strongly encourage you to use R with OpenBLAS or MKL (see installation tutorial here). The low-memory version relies almost exclusively on matrix operations, OpenBLAS/MKL will considerably speed it up and automatically use all cores on your machine.

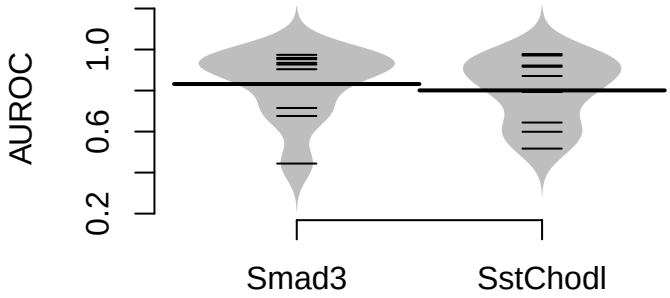
5.3.1 Run previous example with low-memory version

We start by running the examples from the previous sections using the `fast_version = TRUE` flag. Load MetaNeighbor and the tutorial data:

```
library(MetaNeighbor)
library(SummarizedExperiment)
data(mn_data)
data(GOmouse)
```

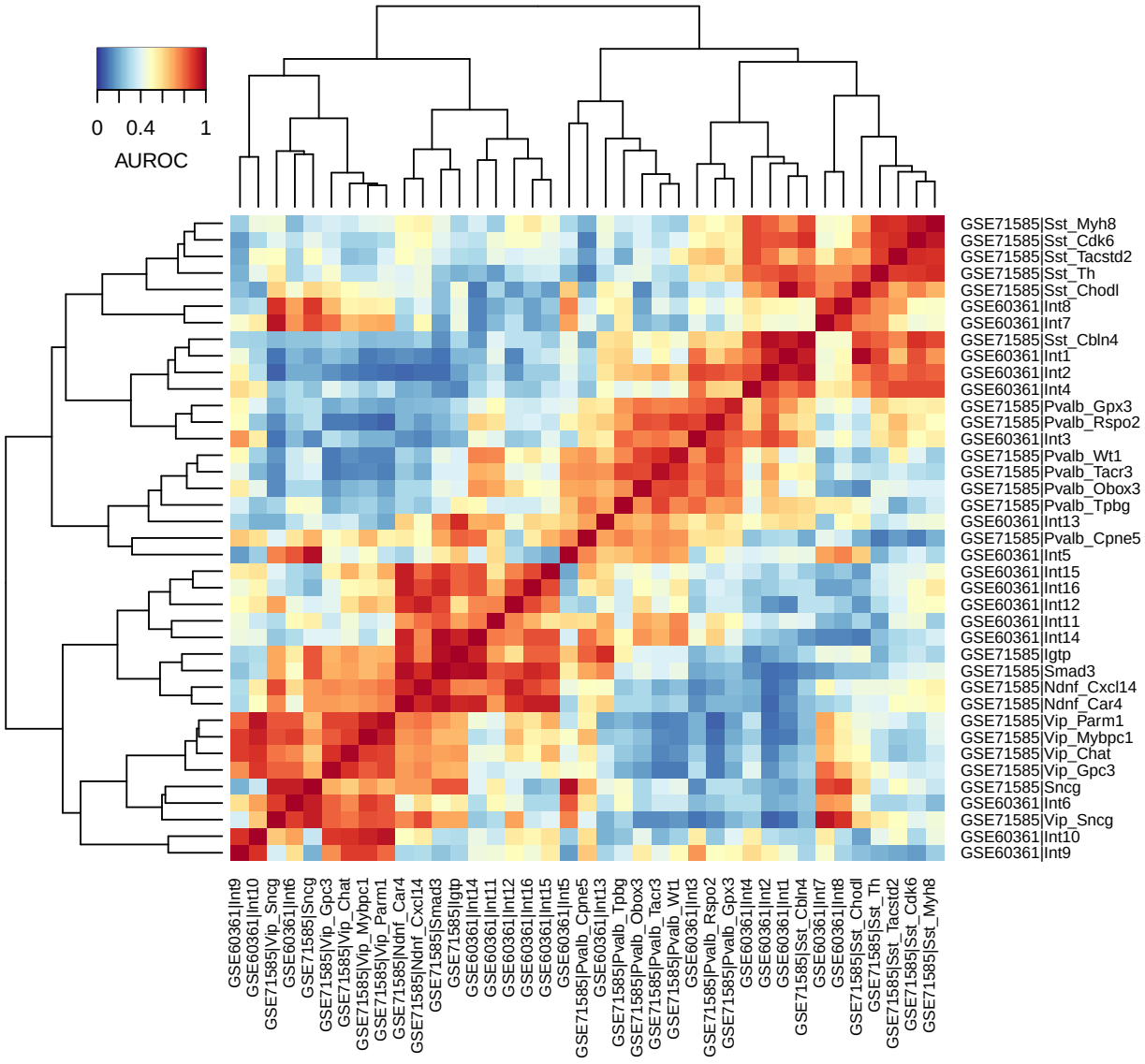
Re-run MetaNeighbor using the same command as above, with `fast_version = TRUE`:

```
AUROC_scores = MetaNeighbor(dat = mn_data,
                            experiment_labels = as.numeric(factor(mn_data$study_id)),
                            celltype_labels = metadata(colData(mn_data))[["cell_labels"]],
                            genesets = GOmouse,
                            bplot = TRUE,
                            fast_version = TRUE)
```



Re-run unsupervised MetaNeighbor, with `fast_version = TRUE`:

```
var_genes = variableGenes(dat = mn_data, exp_labels = mn_data$study_id)
celltype_NV = MetaNeighborUS(var_genes = var_genes,
                             dat = mn_data,
                             study_id = mn_data$study_id,
                             cell_type = mn_data$cell_type,
                             fast_version = TRUE)
cols = rev(colorRampPalette(RColorBrewer::brewer.pal(11,"RdYlBu"))(100))
breaks = seq(0, 1, length=101)
gplots::heatmap.2(celltype_NV,
                  margins=c(8,8),
                  keysize=1,
                  key.xlab="AUROC",
                  key.title="NULL",
                  trace = "none",
                  density.info = "none",
                  col = cols,
                  breaks = breaks,
                  offsetRow=0.1,
                  offsetCol=0.1,
                  cexRow = 0.7,
                  cexCol = 0.7)
```



Because of the approximations in the low-memory version, the AUROCs do not match exactly, but results from the low-memory version are in good qualitative agreement with the exact procedure.

5.3.2 Apply low-memory version to large datasets

The low-memory version is particularly useful when the number of samples across datasets exceeds 10,000. In this section, we show a simple example with 5 datasets from the human pancreas. For simplicity, we use parsed datasets from the Hemberg lab (datasets available here in RDS format). Every dataset is represented as a SingleCellExperiment object, a subclass of SummarizedExperiment, particularly useful for storing large datasets as it is able to handle sparse matrices or HDF5 matrices.

5.3.2.1 Example with 2 datasets: prepare the data Before we apply MetaNeighbor, we need to fuse the datasets as a unique SingleCellExperiment object. Start by downloading the Baron and Segerstolpe datasets, then run R and load the following libraries and data:

```
library(SingleCellExperiment)
library(Matrix)
```

```
baron <- readRDS('baron-human.rds')
segerstolpe <- readRDS('segerstolpe.rds')
```

In this analysis, we remove dead cells and doublets from the Segerstolpe dataset, and retain the genes that are common to the two datasets:

```
common_genes <- intersect(rownames(baron), rownames(segerstolpe))
baron <- baron[common_genes,]
segerstolpe <- segerstolpe[common_genes, !(segerstolpe$cell_type1 %in% c('not applicable', 'co-expressi
```

We create a SingleCellExperiment that is a fusion of the two datasets, then remove the single datasets:

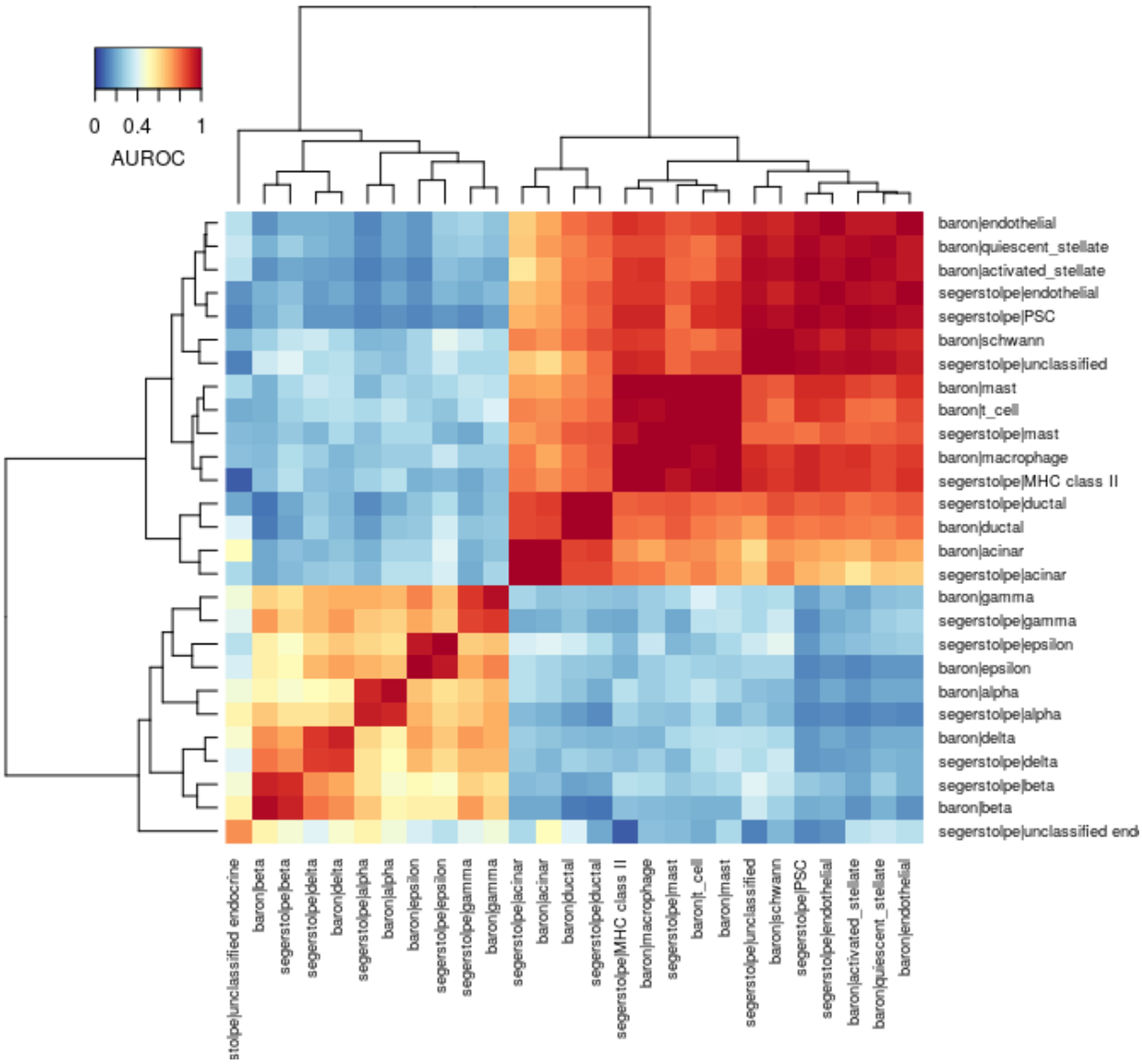
```
new_colData = data.frame(
  study_id = rep(c('baron', 'segerstolpe'), c(ncol(baron), ncol(segerstolpe))),
  cell_type = c(as.character(colData(baron)$cell_type1), colData(segerstolpe)$cell_type1)
)
pancreas <- SingleCellExperiment(
  Matrix(cbind(assay(baron, 1), assay(segerstolpe, 1)), sparse = TRUE),
  colData = new_colData
)
dim(pancreas)
```

The fused dataset has 10,739 samples across 18,936 genes.

5.3.2.2 Example with 2 datasets: match labels

Now that the dataset is ready, we can find variable genes and run the unsupervised version of MetaNeighbor:

```
var_genes = variableGenes(dat = pancreas, exp_labels = pancreas$study_id)
celltype_NV = MetaNeighborUS(var_genes = var_genes,
                             dat = pancreas,
                             study_id = pancreas$study_id,
                             cell_type = pancreas$cell_type,
                             fast_version = TRUE)
cols = rev(colorRampPalette(RColorBrewer::brewer.pal(11,"RdYlBu"))(100))
breaks = seq(0, 1, length=101)
gplots::heatmap.2(celltype_NV,
                  margins=c(8,8),
                  keysize=1,
                  key.xlab="AUROC",
                  key.title="NULL",
                  trace = "none",
                  density.info = "none",
                  col = cols,
                  breaks = breaks,
                  offsetRow=0.1,
                  offsetCol=0.1,
                  cexRow = 0.7,
                  cexCol = 0.7)
```



5.3.3 Apply MetaNeighbor to a collection of 5 datasets

Using a procedure similar to the above example, we generated a SingleCellExperiment with all 5 pancreas datasets. We encourage you to fuse the datasets on your own, then load the resulting SingleCellExperiment object:

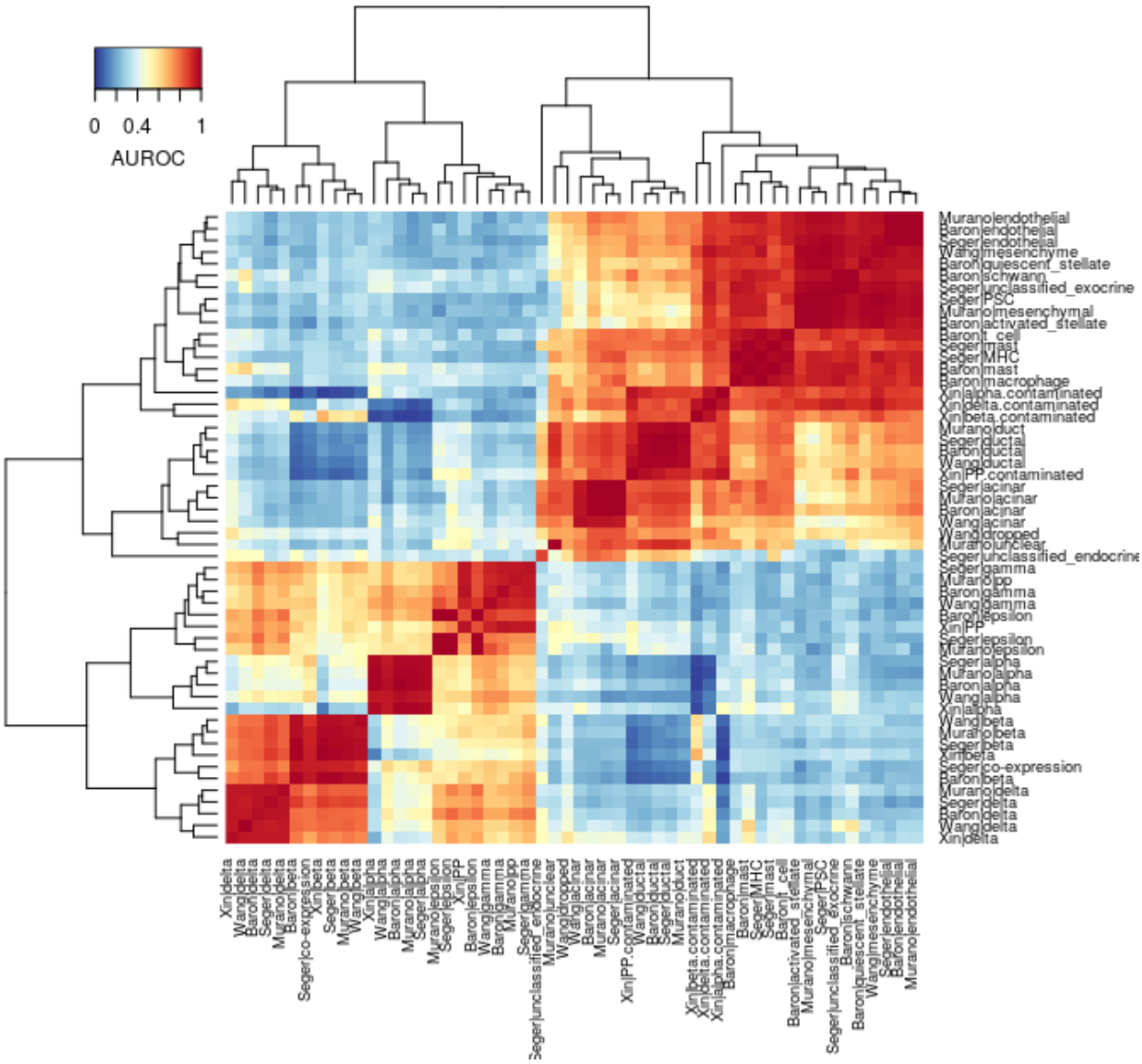
```
all_pancreas <- readRDS('all_pancreas.rds')
dim(all_pancreas)
```

Our fused dataset has 15,138 cells across 15,558 genes.

5.3.3.1 Match labels Select variable genes, then run unsupervised MetaNeighbor to match labels across studies:

```
var_genes = variableGenes(dat = all_pancreas, exp_labels = all_pancreas$Study_ID)
celltype_NV = MetaNeighborUS(var_genes = var_genes,
                             dat = all_pancreas,
                             study_id = all_pancreas$Study_ID,
```

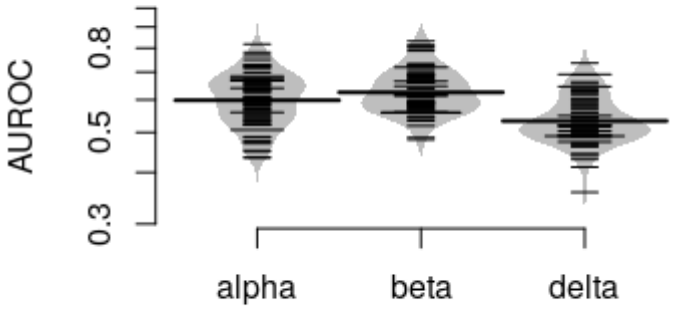
```
                              cell_type = all_pancreas$Celltype,
                              fast_version = TRUE)
cols = rev(colorRampPalette(RColorBrewer::brewer.pal(11,"RdYlBu"))(100))
breaks = seq(0, 1, length=101)
gplots::heatmap.2(celltype_NV,
                  margins=c(8,8),
                  keysize=1,
                  key.xlab="AUROC",
                  key.title="NULL",
                  trace = "none",
                  density.info = "none",
                  col = cols,
                  breaks = breaks,
                  offsetRow=0.1,
                  offsetCol=0.1,
                  cexRow = 0.7,
                  cexCol = 0.7)
```



5.3.3.2 Run MetaNeighbor on common labels

Run MetaNeighbor for labels that span all datasets with 71 gene sets (GO slim terms containing 50 to 1,000 genes):

```
data(GOhuman)
small_pancreas = all_pancreas[, all_pancreas$Celltype %in% c('alpha', 'beta', 'delta')]
celltype_matrix = model.matrix(~small_pancreas$Celltype - 1)
colnames(celltype_matrix) = levels(as.factor(small_pancreas$Celltype))
AUROC_scores = MetaNeighbor(dat = small_pancreas,
                            experiment_labels = as.numeric(factor(small_pancreas$Study_ID)),
                            celltype_labels = celltype_matrix,
                            genesets = GOhuman,
                            bplot = TRUE,
                            fast_version = TRUE)
```



6 FAQ and Contact Information

- If you use this package, please cite Crow et al (2018) Nature Communications.
- Data files used in Crow et al (2018) may be accessed here.
- A development version of MetaNeighbor was first available here (2016) but is no longer maintained.
- For any assistance reproducing analyses please contact mcrow@cshl.edu or jgillis@cshl.edu .