scrtools Documentation

Release 0.3.0

Bo Li, Siranush Sarkizova, Joshua Gould, Marcin Tabaka, Orr Ash

Jul 01, 2018

CONTENTS

1 Version 0.3.0 *June 26, 2018* 3

Index 25

scrtools is a tool for analyzing transcriptomes of millions of single cells. It is a command line tool, a python package and a base for Cloud-based analysis workflows.

CHAPTER

ONE

VERSION 0.3.0 *JUNE 26, 2018*

scrtools supports fast preprocessing, batch-correction, dimension reduction, graph-based clustering, diffusion maps, force-directed layouts, and differential expression analysis, annotate clusters, and plottings.

1.1 Installation

Install **scrtools** locally via Miniconda:

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh .
bash Miniconda3-latest-Linux-x86_64.sh -p /users/foo/miniconda3
mv Miniconda3-latest-Linux-x86_64.sh /users/foo/miniconda3
conda create -n scrtools -y pip
source activate scrtools
conda install -y -c anaconda numpy
conda install -y -c anaconda cython
conda install -y -c conda-forge fftw
export CPATH=$CPATH:/users/foo/miniconda3/envs/scrtools/include
pip install pybind11
git clone https://github.com/nmslib/hnsw.git hnswlib
cd hnswlib/python_bindings
python setup.py install
cd ../..
git clone https://github.com/broadinstitute/scRNA-Seq.git scRNA-Seq
cd scRNA-Seq/scrtools
pip install .
cd ../..
```

1.1.1 Use scrtools in UGER

First, you need to request a RedHat7 server:

```
qrsh -q interactive -l h_vmem=4g -l os=RedHat7 -P regevlab
```

Then, if you have installed **scrtools**, you could activate the virtual environment:

```
source activate scrtools
```

Or, you can use an installed version by typing:

```
source /ahg/regevdata/users/libo/miniconda3/bin/activate scrtools
```

1.2 Use `scrtools` as a command line tool

`scrtools` can be used as a command line tool. Type:

```
scrtools -h
```

to see the help information:

```
Usage:
        scrtools <command> [<args>...]
        scrtools -h | --help
        scrtools -v | --version
```

`scrtools` has 8 sub-commands in 4 groups.

- Preprocessing:

  aggregate_matrix Aggregate cellranger-outputted channel-specific count matrices into a single count matrix. It also enables users to import metadata into the count matrix.

- Analyzing:

  cluster Perform first-pass analysis using the count matrix generated from 'aggregate_matrix'. This subcommand could perform low quality cell filtration, batch correction, variable gene selection, dimension reduction, diffusion map calculation, graph-based clustering, tSNE visualization. The final results will be written into h5ad-formatted file, which Seurat could load.

  de_analysis Detect markers for each cluster by performing differential expression analysis per cluster (within cluster vs. outside cluster). DE tests include Welch's t-test, Fisher's exact test, Mann-Whitney U test. It can also calculate AUROC values for each gene.

  annotate_cluster This subcommand is used to automatically annotate cell types for each cluster based on existing markers. Currently, it only works for human and mouse immune cells.

- Plotting:

  plot Make static plots, which includes plotting tSNEs by cluster labels and different groups.

  iplot Make interactive plots using plotly. The outputs are HTML pages. You can visualize diffusion maps with this sub-command.

- Subclustering:

  view View attribute (e.g. cluster labels) and their values. This subcommand is used to determine cells to run subcluster analysis.

  subcluster Perform sub-cluster analyses on a subset of cells from the analyzed data (i.e. 'cluster' output).

1.2.1 Quick guide

Suppose you have `example.csv` ready with the following contents:

```
Sample,Source,Platform,Donor,Reference,Location
sample_1,bone_marrow,NextSeq,1,GRCh38,/my_dir/sample_1/filtered_gene_bc_matrices_h5.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,/my_dir/sample_2/filtered_gene_bc_matrices_h5.h5
```

(continues on next page)

(continued from previous page)

```
sample_3,pbmc,NextSeq,1,GRCh38,/my_dir/sample_3/filtered_gene_bc_matrices_h5.h5
sample_4,pbmc,NextSeq,2,GRCh38,/my_dir/sample_4/filtered_gene_bc_matrices_h5.h5
```

You want to analyze all four samples but correct batch effects for bone marrow and pbmc samples separately. You can run the following commands:

```
scrtools aggregate_matrix --genome GRCh38 --attributes Source,Platform,Donor example.
↪csv example
scrtools cluster -p 20 --correct-batch-effect --batch-group-by Source -run-louvain --
↪run-tsne example_10x.h5 example
scrtools de_analysis --labels louvain_labels -p 20 --fisher example.h5ad example_de.
↪xlsx
scrtools annotate_cluster example.h5ad example.anno.txt
scrtools plot composition --cluster-labels louvain_labels --attribute Donor --style␣
↪normalized --not-stacked example.h5ad example.composition.png
scrtools plot scatter --basis tsne --attributes louvain_labels,Donor example.h5ad␣
↪example.scatter.png
scrtools iplot --attribute louvain_labels diffmap_pca example.h5ad example.diffmap.
↪html
```

The above analysis will give you tSNE, louvain cluster labels and diffusion maps in `example.h5ad`. You can investigate donor-specific effects by looking at `example.composition.png`. `example.scatter.png` plotted tSNE colored by louvain_labels and Donor info side-by-side. You can explore the diffusion map in 3D by looking at `example.diffmap.html`. This html maps all diffusion components into 3D using PCA.

If you want to perform subcluster analysis by combining cluster 1 and 3, run the following command:

```
scrtools subcluster -p 20 --correct-batch-effect example.h5ad 1,3 example_sub
```

1.2.2 `scrtools aggregate_matrix`

The first step for single cell analysis is to generate one count matrix from cellranger's channel-specific count matrices. `scrtools aggregate_matrix` allows aggregating arbitrary matrices with the help of a *CSV* file.

Type:

```
scrtools aggregate_matrix -h
```

to see the usage information:

```
Usage:
        scrtools aggregate_matrix <csv_file> <output_name> [--genome <genome> --
↪restriction <restriction>... --attributes <attributes> --google-cloud]
        scrtools aggregate_matrix -h
```

- Arguments:

  csv_file Input csv-formatted file containing information of each 10x channel. Each row must contain at least 3 columns — Sample, sample name; Location, location of the channel-specific count matrix in 10x format (e.g. /sample/filtered_gene_bc_matrices_h5.h5); Reference, genome reference used for 10x cellranger. See below for an example csv:

```
Sample,Source,Platform,Donor,Reference,Location
sample_1,bone_marrow,NextSeq,1,GRCh38,/my_dir/sample_1/filtered_gene_
↪bc_matrices_h5.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,/my_dir/sample_2/filtered_gene_
↪bc_matrices_h5.h5
sample_3,pbmc,NextSeq,1,GRCh38,/my_dir/sample_3/filtered_gene_bc_
↪matrices_h5.h5
sample_4,pbmc,NextSeq,2,GRCh38,/my_dir/sample_4/filtered_gene_bc_
↪matrices_h5.h5
```

output_name The output file name.

- Options:

  –genome <genome> Genome reference. [default: GRCh38]

  –restriction <restriction>… Select channels that satisfy all restrictions. Each restriction takes the format of name:value,…,value or name:~value,..,value, where ~ refers to not. You can specifiy multiple restrictions by setting this option multiple times.

  –attributes <attributes> Specify a comma-separated list of outputted attributes. These attributes should be column names in the csv file.

  –google-cloud If files are stored in google cloud. Assuming google cloud sdk is installed.

  -h, –help Print out help information.

- Outputs:

  output_name_10x.h5 A 10x-formatted HDF5 file containing the count matrix and associated attributes.

- Examples:

```
scrtools aggregate_matrix --genome GRCh38 --restriction Source:pbmc --restriction␣
↪Donor:1 --attributes Source,Platform,Donor example.csv example
```

1.2.3 scrtools cluster

Once we collected the count matrix `example_10x.h5`, we can perform single cell analysis using `scrtools cluster`.

Type:

```
scrtools cluster -h
```

to see the usage information:

```
Usage:
        scrtools cluster [options] <input_file> <output_name>
        scrtools cluster -h
```

- Arguments:

  input_file Input file in 10x format. If first-pass analysis has been performed, but you want to run some additional analysis, you could also pass a h5ad-formatted file.

  output_name Output file name. All outputs will use it as the prefix.

- Options:

  -p <number>, –threads <number> Number of threads. [default: 1]

  –genome <genome> Genome name. [default: GRCh38]

  –processed Input file is processed and thus no PCA & diffmap will be run.

  –output-filtration-results <spreadsheet> Output filtration results into <spreadsheet>.

  –output-loom Output loom-formatted file.

  –correct-batch-effect Correct for batch effects.

  –batch-group-by <expression> Batch correction assumes the differences in gene expression between channels are due to batch effects. However, in many cases, we know that channels can be partitioned into several groups and each group is biologically different from others. In this case, we will only perform batch correction for channels within each group. This option defines the groups. If <expression> is None, we assume all channels are from one group. Otherwise, groups are defined according to <expression>. <expression> takes the form of either 'attr', or 'attr1+attr2+…+attrn', or 'attr=value11,…,value1n_1;value21,…,value2n_2;…;valuem1,…,valuemn_m'. In the first form, 'attr' should be an existing sample attribute, and groups are defined by 'attr'. In the second form, 'attr1',…,'attrn' are n existing sample attributes and groups are defined by the Cartesian product of these n attributes. In the last form, there will be m + 1 groups. A cell belongs to group i (i > 0) if and only if its sample attribute 'attr' has a value among valuei1,…,valuein_i. A cell belongs to group 0 if it does not belong to any other groups.

  –min-genes <number> Only keep cells with at least <number> of genes. [default: 500]

  –max-genes <number> Only keep cells with less than <number> of genes. [default: 6000]

  –mito-prefix <prefix> Prefix for mitochondrial genes. [default: MT-]

  –percent-mito <ratio> Only keep cells with mitochondrial ratio less than <ratio>. [default: 0.1]

  –gene-percent-cells <ratio> Only use genes that are expressed in at <ratio> * 100 percent of cells to select variable genes. [default: 0.0005]

  –counts-per-cell-after <number> Total counts per cell after normalization. [default: 1e5]

  –random-state <seed> Random number generator seed. [default: 0]

  –run-uncentered-pca Run uncentered PCA.

  –no-variable-gene-selection Do not select variable genes.

  –no-submat-to-dense Do not convert variable-gene-selected submatrix to a dense matrix.

  –nPC <number> Number of PCs. [default: 50]

  –nDC <number> Number of diffusion components. [default: 50]

  –diffmap-alpha <alpha> Power parameter for diffusion-based pseudotime. [default: 0.5]

  –diffmap-K <K> Number of neighbors used for constructing affinity matrix. [default: 100]

  –calculate-pseudotime <roots> Calculate diffusion-based pseudotimes based on <roots>. <roots> should be a comma-separated list of cell barcodes.

  –run-louvain Run louvain clustering algorithm.

  –louvain-resolution <resolution> Resolution parameter for the louvain clustering algorithm. [default: 1.3]

–louvain-affinity <affinity> Affinity matrix to be used. Could be 'W_norm', 'W_diffmap', or 'W_diffmap_norm'. [default: W_norm]

–run-kmeans Run KMeans clustering algorithm on diffusion components.

–kmeans-n-clusters <number> Target at <number> clusters for K means. [default: 20]

–run-hdbscan Run hdbscan clustering algorithm on diffusion components.

–hdbscan-min-cluster-size <number> Minimum cluster size for hdbscan. [default: 50]

–hdbscan-min-samples <number> Minimum number of samples for hdbscan. [default: 50]

–run-approximated-louvain Run approximated louvain clustering algorithm.

–approx-louvain-ninit <number> Number of Kmeans tries. [default: 20]

–approx-louvain-nclusters <number> Number of clusters for Kmeans initialization. [default: 30]

–approx-louvain-resolution <resolution>. Resolution parameter for louvain. [default: 1.3]

–run-tsne Run multi-core tSNE for visualization.

–tsne-perplexity <perplexity> tSNE's perplexity parameter. [default: 30]

–run-fitsne Run FItSNE for visualization.

–run-umap Run umap for visualization.

–umap-on-diffmap Run umap on diffusion components.

–umap-K <K> K neighbors for umap. [default: 15]

–umap-min-dist <number> Umap parameter. [default: 0.1]

–umap-spread <spread> Umap parameter. [default: 1.0]

–run-fle Run force-directed layout embedding.

–fle-K <K> K neighbors for building graph for FLE. [default: 50]

–fle-n-steps <nstep> Number of iterations for FLE. [default: 10000]

–fle-affinity <affinity> Affinity matrix to be used. Could be 'W_diffmap', or 'W_diffmap_norm'. [default: W_diffmap]

-h, –help Print out help information.

- Outputs:

output_name.h5ad Output file in h5ad format. The clustering results are stored in the 'obs' field (e.g. 'louvain_labels' for louvain cluster labels). The PCA, tSNE and diffusion map coordinates are stored in the 'obsm' field.

output_name.loom Optional output. Only exists if '–output-loom' is set. output_name.h5ad in loom format for visualization.

- Examples:

```
scrtools cluster -p 20 --correct-batch-effect --run-louvain --run-tsne example_
↪10x.h5 example
```

1.2.4 `scrtools de_analysis`

Once we have the clusters, we can detect markers using `scrtools de_analysis`.

Type:

```
scrtools de_analysis -h
```

to see the usage information:

```
Usage:
        scrtools de_analysis [--labels <attr> -p <threads> --alpha <alpha> --fisher --
↪mwu --roc] <input_h5ad_file> <output_spreadsheet>
        scrtools de_analysis -h
```

- Arguments:

  input_h5ad_file Single cell data with clustering calculated. DE results would be written back.

  output_spreadsheet Output spreadsheet with DE results.

- Options:

  –labels <attr> <attr> used as cluster labels. [default: louvain_labels]

  –alpha <alpha> Control false discovery rate at <alpha>. [default: 0.05]

  –fisher Calculate Fisher's exact test.

  –mwu Calculate Mann-Whitney U test.

  –roc Calculate area under cuver in ROC curve.

  -p <threads> Use <threads> threads. [default: 1]

  -h, –help Print out help information.

- Outputs:

  input_h5ad_file DE results would be written back to the 'var' fields.

  output_spreadsheet An excel spreadsheet containing DE results. Each cluster has two tabs in the spreadsheet. One is for up-regulated genes and the other is for down-regulated genes.

- Examples:

  ```
  scrtools de_analysis --labels louvain_labels -p 20 --fisher --mwu --roc example.
  ↪h5ad example_de.xlsx
  ```

1.2.5 `scrtools annotate_cluster`

Once we have the DE results, we could optionally identify putative cell types for each cluster using `scrtools annotate_cluster`. Currently, this subcommand only works for human and mouse immune cells.

Type:

```
scrtools annotate_cluster -h
```

to see the usage information:

```
Usage:
        scrtools annotate_cluster [--json-file <file> --minimum-report-score <score> -
↪-do-not-use-non-de-genes] <input_h5ad_file> <output_file>
        scrtools annotate_cluster -h
```

- Arguments:

  input_h5ad_file Single cell data with DE analysis done by `scrtools de_analysis`.

  output_file Output annotation file.

- Options:

  –json-file <file> JSON file for markers. Could also be `human`/`mouse`. [default: human]

  –minimum-report-score <score> Minimum cell type score to report a potential cell type. [default: 0.5]

  –do-not-use-non-de-genes Do not count non DE genes as down-regulated.

  -h, –help Print out help information.

- Outputs:

  output_file This is a text file. For each cluster, all its putative cell types are listed in descending order of the cell type score. For each putative cell type, all markers support this cell type are listed. If one putative cell type has cell subtypes, all subtypes will be listed under this cell type.

- Examples:

  ```
  scrtools annotate_cluster example.h5ad example.anno.txt
  ```

1.2.6 `scrtools plot`

We can make a variety of figures using `scrtools plot`.

Type:

```
scrtools plot -h
```

to see the usage information:

```
Usage:
        scrtools plot [options] [--restriction <restriction>...] <plot_type> <input_
↪h5ad_file> <output_file>
        scrtools plot -h
```

- Arguments:

  plot_type Only 2D plots, chosen from 'composition', 'scatter', 'scatter_groups', 'scatter_genes', 'scatter_gene_groups', and 'heatmap'.

  input_h5ad_file Single cell data with clustering done by Scanpy in h5ad file format.

  output_file Output image file.

- Options:

  –dpi <dpi> DPI value for the figure. [default: 500]

–cluster-labels <attr> Use <attr> as cluster labels. This option is used in 'composition', 'scatter_groups', and 'heatmap'.

–attribute <attr> Plot <attr> against cluster labels. This option is only used in 'composition'.

–basis <basis> Basis for 2D plotting, chosen from 'tsne', 'fitsne', 'umap', 'pca', 'rpca', 'fle', or 'diffmap_pca'. This option is used in 'scatter', 'scatter_groups', 'scatter_genes', and 'scatter_gene_groups'. [default: tsne]

–attributes <attrs> <attrs> is a comma-separated list of attributes to color the basis. This option is only used in 'scatter'.

–restriction <restriction>… Multiple <restriction> strings for different attributes. Each <restriction> takes the format of 'attr:value,value'. Only used for scatter.

–group <attr> <attr> is used to make group plots. In group plots, the first one contains all components in the group and the following plots show each component separately. This option is iused in 'scatter_groups' and 'scatter_gene_groups'. If <attr> is a semi-colon-separated string, parse the string as groups.

–genes <genes> <genes> is a comma-separated list of gene names to visualize. This option is used in 'scatter_genes' and 'heatmap'.

–gene <gene> Visualize <gene> in group plots. This option is only used in 'scatter_gene_groups'.

–style <style> Composition plot styles. Can be either 'frequency', 'count', or 'normalized'. [default: frequency]

–not-stacked Do not stack bars in composition plot.

–log-y Plot y axis in log10 scale for composition plot.

–nrows <nrows> Number of rows in the figure. If not set, scrtools will figure it out automatically.

–ncols <ncols> Number of columns in the figure. If not set, scrtools will figure it out automatically.

–subplot-size <sizes> Sub-plot size in inches, w x h, separated by comma. Note that margins are not counted in the sizes. For composition, default is (6, 4). For scatter plots, default is (4, 4).

–left <left> Figure's left margin in fraction with respect to subplot width.

–bottom <bottom> Figure's bottom margin in fraction with respect to subplot height.

–wspace <wspace> Horizontal space between subplots in fraction with respect to subplot width.

–hspace <hspace> Vertical space between subplots in fraction with respect to subplot height.

–alpha <alpha> Point transparent parameter.

–legend-fontsize <fontsize> Legend font size.

–use-raw Use anndata stored raw expression matrix. Only used by 'scatter_genes' and 'scatter_gene_groups'.

–do-not-show-all Do not show all components in group for scatter_groups.

–show-zscore If show zscore in heatmap.

–heatmap-title <title> Title for heatmap.

-h, –help Print out help information.

Examples:

```
scrtools plot composition --cluster-labels louvain_labels --attribute Donor --style␣
↪normalized --not-stacked example.h5ad example.composition.png
scrtools plot scatter --basis tsne --attributes louvain_labels,Donor example.h5ad␣
↪example.scatter.png
scrtools plot scatter_groups --cluster-labels louvain_labels --group Donor example.
↪h5ad example.scatter_groups.png
scrtools plot scatter_genes --genes CD8A,CD4,CD3G,MS4A1,NCAM1,CD14,ITGAX,IL3RA,CD38,
↪CD34,PPBP example.h5ad example.genes.png
scrtools plot scatter_gene_groups --gene CD8A --group Donor example.h5ad example.gene_
↪groups.png
scrtools plot heatmap --cluster-labels louvain_labels --genes CD8A,CD4,CD3G,MS4A1,
↪NCAM1,CD14,ITGAX,IL3RA,CD38,CD34,PPBP --heatmap-title 'markers' example.h5ad␣
↪example.heatmap.png
```

1.2.7 `scrtools iplot`

We can also make interactive plots in html format using `scrtools iplot`. These interactive plots are very helpful if you want to explore the diffusion maps.

Type:

```
scrtools iplot -h
```

to see the usage information:

```
Usage:
        scrtools iplot --attribute <attr> [options] <basis> <input_h5ad_file> <output_
↪html_file>
        scrtools iplot -h
```

- Arguments:

  basis Basis can be either 'tsne', 'fitsne', 'umap', 'diffmap', 'pca', 'rpca' or 'diffmap_pca'.

  input_h5ad_file Single cell data with clustering done in h5ad file format.

  output_html_file Output interactive plot in html format.

- Options:

  –attribute <attr> Use attribute <attr> as labels in the plot.

  –is-real <attr> is real valued.

  –is-gene <attr> is a gene name.

  –log10 If take log10 of real values.

  -h, –help Print out help information.

- Examples:

```
scrtools iplot --attribute louvain_labels tsne example.h5ad example.tsne.html
scrtools iplot --attribute louvain_labels diffmap_pca example.h5ad example.
↪diffmap.html
```

1.2.8 scrtools view

We may want to further perform sub-cluster analysis on a subset of cells. This sub-command helps us to define the subset.

Type:

```
scrtools view -h
```

to see the usage information:

```
Usage:
        scrtools view [--show-attributes --show-gene-attributes --show-values-for-
↪attributes <attributes>] <input_h5ad_file>
        scrtools view -h
```

- Arguments:

  input_h5ad_file Analyzed single cell data in h5ad format.

- Options:

  –show-attributes Show the available sample attributes in the input dataset.

  –show-gene-attributes Show the available gene attributes in the input dataset.

  –show-values-for-attributes <attributes> Show the available values for specified attributes in the input dataset. <attributes> should be a comma-separated list of attributes.

  -h, –help Print out help information.

- Examples:

```
scrtools view --show-attributes example.h5ad
scrtools view --show-gene-attributes example.h5ad
scrtools view --show-values-for-attributes louvain_labels,Donor example.h5ad
```

1.2.9 scrtools subcluster

If there is a subset of cells that we want to further cluster, we can run `scrtools subcluster`. This sub-command will outputs a new h5ad file that you can run `de_analysis`, `plot` and `iplot` on.

Type:

```
scrtools subcluster -h
```

to see the usage information:

```
Usage:
        scrtools subcluster [options] --subset-selection <subset-selection>... <input_
↪file> <output_name>
        scrtools subcluster -h
```

- Arguments:

  input_file Single cell data with clustering done in h5ad format.

  output_name Output file name. All outputs will use it as the prefix.

- Options:

  –subset-selection <subset-selection>… Specify which cells will be included in the subcluster analysis. Each <subset_selection> string takes the format of 'attr:value,…,value', which means select cells with attr in the values. If multiple <subset_selection> strings are specified, the subset of cells selected is the intersection of these strings.

  -p <number>, –threads <number> Number of threads. [default: 1]

  –correct-batch-effect Correct for batch effects.

  –output-loom Output loom-formatted file.

  –random-state <seed> Random number generator seed. [default: 0]

  –run-uncentered-pca Run uncentered PCA.

  –no-variable-gene-selection Do not select variable genes.

  –no-submat-to-dense Do not convert variable-gene-selected submatrix to a dense matrix.

  –nPC <number> Number of PCs. [default: 50]

  –nDC <number> Number of diffusion components. [default: 50]

  –diffmap-alpha <alpha> Power parameter for diffusion-based pseudotime. [default: 0.5]

  –diffmap-K <K> Number of neighbors used for constructing affinity matrix. [default: 100]

  –calculate-pseudotime <roots> Calculate diffusion-based pseudotimes based on <roots>. <roots> should be a comma-separated list of cell barcodes.

  –run-louvain Run louvain clustering algorithm.

  –louvain-resolution <resolution> Resolution parameter for the louvain clustering algorithm. [default: 1.3]

  –louvain-affinity <affinity> Affinity matrix to be used. Could be 'W_norm', 'W_diffmap', or 'W_diffmap_norm'. [default: W_norm]

  –run-kmeans Run KMeans clustering algorithm on diffusion components.

  –kmeans-n-clusters <number> Target at <number> clusters for K means. [default: 20]

  –run-hdbscan Run hdbscan clustering algorithm on diffusion components.

  –hdbscan-min-cluster-size <number> Minimum cluster size for hdbscan. [default: 50]

  –hdbscan-min-samples <number> Minimum number of samples for hdbscan. [default: 50]

  –run-approximated-louvain Run approximated louvain clustering algorithm.

  –approx-louvain-ninit <number> Number of Kmeans tries. [default: 20]

  –approx-louvain-nclusters <number> Number of clusters for Kmeans initialization. [default: 30]

  –approx-louvain-resolution <resolution>. Resolution parameter for louvain. [default: 1.3]

  –run-tsne Run multi-core tSNE for visualization.

  –tsne-perplexity <perplexity> tSNE's perplexity parameter. [default: 30]

  –run-fitsne Run FItSNE for visualization.

  –run-umap Run umap for visualization.

  –umap-on-diffmap Run umap on diffusion components.

  –umap-K <K> K neighbors for umap. [default: 15]

–umap-min-dist <number> Umap parameter. [default: 0.1]

–umap-spread <spread> Umap parameter. [default: 1.0]

–run-fle Run force-directed layout embedding.

–fle-K <K> K neighbors for building graph for FLE. [default: 50]

–fle-n-steps <nstep> Number of iterations for FLE. [default: 10000]

–fle-affinity <affinity> Affinity matrix to be used. Could be 'W_diffmap', or 'W_diffmap_norm'. [default: W_diffmap]

-h, –help Print out help information.

- Outputs:

  output_name.h5ad Output file in h5ad format. The clustering results are stored in the 'obs' field (e.g. 'louvain_labels' for louvain cluster labels). The PCA, tSNE and diffusion map coordinates are stored in the 'obsm' field.

  output_name.loom Optional output. Only exists if '–output-loom' is set. output_name.h5ad in loom format for visualization.

- Examples:

```
scrtools subcluster --subset_selection louvain_labels:1,3  --subset_selection␣
↪Donor:1 -p 20 --correct-batch-effect example.h5ad example_sub
```

1.3 API

`scrtools` can also be used as a python package. Import `scrtools` by:

```
import scrtools
```

1.3.1 Tools:

Aggregate channel-specific count matrices

tools.aggregate_10x_matrices(csv_file, …)	Aggregate channel-specific 10x count matrices into one big count matrix.

tools.aggregate_10x_matrices

`tools.`**`aggregate_10x_matrices`**(*csv_file*, *genome*, *restrictions*, *attributes*, *output_file*, *google_cloud=False*)

Aggregate channel-specific 10x count matrices into one big count matrix.

This function takes as input a csv_file, which contains at least 3 columns — Sample, sample name; Location, folder that contains the count matrices (e.g. filtered_gene_bc_matrices_h5.h5); Reference, genome reference used for 10x cellranger. It outputs a 10x-formatted HDF5 file for the big count matrix.

Parameters

- **`csv_file`** (*str*) – The CSV file containing information about each 10x channel.
- **`genome`** (*str*) – The genome each sample comes from.

- **restrictions** (*list[str]*) – A list of restrictions used to select channels, each restriction takes the format of name:value,. . . ,value or name:~value,..,value, where ~ refers to not.
- **attributes** (*list[str]*) – A list of attributes need to be incorporated into the output count matrix.
- **output_file** (*str*) – The output count matrix file, normally the file name ends with '_10x.h5'.
- **google_cloud** (*bool*, optional (default: *False*)) – If the channel-specific count matrices are stored in a google bucket.

Returns

Return type None

Examples

```
>>> tools.aggregate_matrix('example.csv', 'GRCh38', ['Source:pbmc', 'Donor:1'], [
↪'Source', 'Platform', 'Donor'], 'example_10x.h5')
```

Preprocess

tools.read_input(input_file[, is_raw, genome])	Load either 10x-formatted raw count matrix or h5ad-formatted processed expression matrix into memory.
tools.update_var_names(data, genome)	
tools.filter_data(data[, mito_prefix, . . .])	
tools.log_norm(data, norm_count)	Normalization and then take log
tools.run_pca(data[, standardize, . . .])	
tools.run_rpca(data[, scale, max_value, . . .])	smooth outliers, then no center/scale data
tools.get_anndata_for_subclustering(data, . . .)	

tools.read_input

tools.**read_input** (*input_file*, *is_raw=True*, *genome='GRCh38'*)

Load either 10x-formatted raw count matrix or h5ad-formatted processed expression matrix into memory.

This function is used to load input data into memory. If the input is 10x-formatted raw count matrix, the whole matrix will be loaded into the memory. Otherwise, only the sample and gene attributes are loaded.

Parameters

- **input_file** (*str*) – Input file name.
- **is_raw** (*bool*, optional (default: *True*)) – If input file is 10x-formatted raw count matrix.
- **genome** (*str*, optional (default: *GRCh38*)) – The genome used to produce raw count matrices.

Returns An *anndata* object contains the count matrix.

Return type *anndata* object

Examples

```
>>> adata = tools.read_input('example_10x.h5', is_raw = True, genome = 'mm10')
```

tools.update_var_names

`tools.update_var_names`(*data*, *genome*)

tools.filter_data

`tools.filter_data`(*data*, *mito_prefix='MT-'*, *filt_xlsx=None*, *min_genes=500*, *max_genes=6000*, *percent_mito=0.1*, *percent_cells=0.0005*)

tools.log_norm

`tools.log_norm`(*data*, *norm_count*)
Normalization and then take log

tools.run_pca

`tools.run_pca`(*data*, *standardize=True*, *max_value=10*, *nPC=50*, *random_state=0*)

tools.run_rpca

`tools.run_rpca`(*data*, *scale=False*, *max_value=10.0*, *nPC=50*, *random_state=0*)
smooth outliers, then no center/scale data

tools.get_anndata_for_subclustering

`tools.get_anndata_for_subclustering`(*data*, *subset_selections*)

Batch correction

tools.set_group_attribute(data, attribute_string)
tools.estimate_adjustment_matrices(data)
tools.filter_genes_dispersion(data, . . . [, . . .])
tools.collect_variable_gene_matrix(data, . . .)
tools.correct_batch_effects(data)

tools.set_group_attribute

`tools.set_group_attribute`(*data*, *attribute_string*)

tools.estimate_adjustment_matrices

`tools.`**`estimate_adjustment_matrices`**(*data*)

tools.filter_genes_dispersion

`tools.`**`filter_genes_dispersion`**(*data, consider_batch, min_disp=0.5, max_disp=None, min_mean=0.0125, max_mean=7*)

tools.collect_variable_gene_matrix

`tools.`**`collect_variable_gene_matrix`**(*data, gene_subset*)

tools.correct_batch_effects

`tools.`**`correct_batch_effects`**(*data*)

Diffusion map

`tools.run_diffmap`(data, rep_key[, n_jobs, . . .])	
`tools.run_pseudotime_calculation`(data, roots)	

tools.run_diffmap

`tools.`**`run_diffmap`**(*data, rep_key, n_jobs=1, n_components=100, alpha=0.5, K=100, random_state=0, knn_method='hnsw', eigen_solver='randomized', M=15, efC=100, efS=100*)

tools.run_pseudotime_calculation

`tools.`**`run_pseudotime_calculation`**(*data, roots*)

Cluster algorithms

`tools.run_louvain`(data[, affinity, . . .])	
`tools.run_hdbscan`(data, rep_key[, n_jobs, . . .])	
`tools.run_kmeans`(data, rep_key, n_clusters)	
`tools.run_approximated_louvain`(data, rep_key)	

tools.run_louvain

`tools.`**`run_louvain`**(*data, affinity='W_norm', resolution=1.3, random_state=0*)

tools.run_hdbscan

`tools.`**`run_hdbscan`**(*data, rep_key, n_jobs=1, min_cluster_size=50, min_samples=25*)

tools.run_kmeans

tools.**run_kmeans**(*data*, *rep_key*, *n_clusters*, *n_init=10*, *n_jobs=1*, *random_state=0*)

tools.run_approximated_louvain

tools.**run_approximated_louvain**(*data*, *rep_key*, *n_jobs=1*, *resolution=1.3*, *random_state=0*, *n_clusters=30*, *n_init=20*)

Visualization algorithms

tools.run_tsne(data, rep_key, n_jobs[, …])	
tools.run_fitsne(data, rep_key, n_jobs[, …])	
tools.run_umap(data, rep_key[, …])	
tools.run_force_directed_layout(data, …[, …])	

tools.run_tsne

tools.**run_tsne**(*data*, *rep_key*, *n_jobs*, *n_components=2*, *perplexity=30*, *early_exaggeration=12*, *learning_rate=1000*, *random_state=0*)

tools.run_fitsne

tools.**run_fitsne**(*data*, *rep_key*, *n_jobs*, *n_components=2*, *perplexity=30*, *early_exaggeration=12*, *random_state=0*)

tools.run_umap

tools.**run_umap**(*data*, *rep_key*, *n_components=2*, *n_neighbors=15*, *min_dist=0.1*, *spread=1.0*, *random_state=0*)

tools.run_force_directed_layout

tools.**run_force_directed_layout**(*data*, *file_name*, *n_jobs*, *affinity='W_diffmap'*, *K=50*, *layout='fa'*, *n_steps=10000*, *memory=20*)

Differential expression analysis

tools.run_de_analysis(input_file, …)	

tools.run_de_analysis

tools.**run_de_analysis**(*input_file*, *output_excel_file*, *labels*, *n_jobs*, *alpha*, *run_fisher*, *run_mwu*, *run_roc*)

1.3.2 Annotate clusters:

annotate_cluster. *annotate_clusters*(data, ...)	

annotate_cluster.annotate_clusters

annotate_cluster.**annotate_clusters**(*data*, *json_file*, *thre*, *fout=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *ignoreNA=False*)

1.3.3 Plotting:

Static plots

plotting.plot_composition(data, cluster, attr)	Generate a composition plot, which shows the percentage of cells from each condition for every cluster.
plotting.plot_scatter(data, basis, attrs[, ...])	
plotting.plot_scatter_groups(data, basis, ...)	
plotting.plot_scatter_genes(data, basis, genes)	
plotting.plot_scatter_gene_groups(data, ...)	
plotting.plot_heatmap(data, cluster, genes)	

plotting.plot_composition

plotting.**plot_composition**(*data*, *cluster*, *attr*, *style='frequency'*, *stacked=True*, *logy=False*, *subplot_size=(6, 4)*, *left=0.15*, *bottom=None*, *wspace=0.2*, *hspace=None*)

Generate a composition plot, which shows the percentage of cells from each condition for every cluster.

This function is used to generate composition plots, which are bar plots showing the cell compositions (from different conditions) for each cluster. This type of plots is useful to fast assess library quality and batch effects.

Parameters

- **data** (*anndata* object) – Single cell expression data as an anndata object.
- **cluster** (*str*) – A string represents cluster labels, e.g. louvain_labels.
- **attr** (*str*) – A sample attribute representing the condition, e.g. Donor.
- **style** (*str*, optional (default: *frequency*)) – Composition plot style. Can be either *frequency*, *count*, or 'normalized'. Within each cluster, the *frequency* style show the ratio of cells from each condition over all cells in the cluster, the *count* style just shows the number of cells from each condition, the *normalized* style shows the percentage of cells from the condition in this cluster over the total number of cells from the condition for each condition.
- **stacked** (*bool*, optional (default: *True*)) – If stack the bars from each condition.
- **logy** (*bool*, optional (default: *False*)) – If show the y-axis in log10 scale
- **subplot_size** (*tuple*, optional (default: *(6, 4)*)) – The plot size (width, height) in inches.
- **left** (*float*, optional (default: *0.15*)) – This parameter sets the figure's left margin as a fraction of subplot's width (left * subplot_size[0]).

- **bottom** (*float*, optional (default: *0.15*)) – This parameter sets the figure's bottom margin as a fraction of subplot's height (bottom * subplot_size[1]),
- **wspace** (*float*, optional (default: *0.2*)) – This parameter sets the width between subplots and also the figure's right margin as a fraction of subplot's width (wspace * subplot_size[0]).
- **hspace** (*float*, optional (defualt: *0.15*)) – This parameter sets the height between subplots and also the figure's top margin as a fraction of subplot's height (hspace * subplot_size[1]).

Returns A *matplotlib.figure.Figure* object containing the composition plot.

Return type *Figure* object

Examples

```
>>> fig = plotting.plot_composition(data, 'louvain_labels', 'Donor', style =
↪'normalized', stacked = False)
```

plotting.plot_scatter

plotting.**plot_scatter**(*data*, *basis*, *attrs*, *restrictions=[]*, *nrows=None*, *ncols=None*, *subplot_size=(4, 4)*, *left=None*, *bottom=None*, *wspace=None*, *hspace=None*, *alpha=None*, *legend_fontsize=None*)

plotting.plot_scatter_groups

plotting.**plot_scatter_groups**(*data*, *basis*, *cluster*, *group*, *restrictions=[]*, *nrows=None*, *ncols=None*, *subplot_size=(4, 4)*, *left=None*, *bottom=None*, *wspace=None*, *hspace=None*, *alpha=None*, *legend_fontsize=None*, *showall=True*)

plotting.plot_scatter_genes

plotting.**plot_scatter_genes**(*data*, *basis*, *genes*, *nrows=None*, *ncols=None*, *subplot_size=(4, 4)*, *left=None*, *bottom=None*, *wspace=0.3*, *hspace=None*, *alpha=None*, *use_raw=False*)

plotting.plot_scatter_gene_groups

plotting.**plot_scatter_gene_groups**(*data*, *basis*, *gene*, *group*, *nrows=None*, *ncols=None*, *subplot_size=(4, 4)*, *left=None*, *bottom=None*, *wspace=0.3*, *hspace=None*, *alpha=None*, *use_raw=False*)

plotting.plot_heatmap

plotting.**plot_heatmap**(*data*, *cluster*, *genes*, *use_raw=False*, *showzscore=False*, *title="*, ***kwargs*)

Interactive plots

plotting.scatter(df, output_file)	

Continued on next page

Table 10 – continued from previous page

plotting.scatter_real(df, output_file[, log10])	
plotting.scatter3d(df, output_file)	
plotting.scatter3d_real(df, output_file[, log10])	

plotting.scatter

plotting.**scatter**(*df*, *output_file*)

plotting.scatter_real

plotting.**scatter_real**(*df*, *output_file*, *log10=False*)

plotting.scatter3d

plotting.**scatter3d**(*df*, *output_file*)

plotting.scatter3d_real

plotting.**scatter3d_real**(*df*, *output_file*, *log10=False*)

1.3.4 Miscellaneous:

misc.search_genes(data, gene_list[, measure])	Extract and display gene expressions for each cluster from an *anndata* object.
misc.search_de_genes(data, gene_list[, . . .])	Extract and display differential expression analysis results of markers for each cluster from an *anndata* object.

misc.search_genes

misc.**search_genes**(*data*, *gene_list*, *measure='percentage'*)

Extract and display gene expressions for each cluster from an *anndata* object.

This function helps to see marker expressions in clusters via the interactive python environment.

Parameters

- **data** (*anndata* object) – An *anndata* object containing the expression matrix and differential expression results.
- **gene_list** (*list[str]*) – A list of gene symbols.
- **measure** (*str*) – Can be either *percentage* or *mean_log_expression*. *percentage* shows the percentage of cells expressed the genes and *mean_log_expression* shows the mean log expression.

Returns A data frame containing marker expressions in each cluster.

Return type *pandas.DataFrame*

Examples

```
>>> results = misc.search_genes(data, ['CD3E', 'CD4', 'CD8'], measure =
↪'percentage')
```

misc.search_de_genes

misc.**search_de_genes**(*data*, *gene_list*, *test='fisher'*, *thre=1.5*)

Extract and display differential expression analysis results of markers for each cluster from an *anndata* object.

This function helps to see if markers are up or down regulated in each cluster via the interactive python environment. ++ indicates up-regulated and fold change >= threshold, + indicates up-regulated but fold change < threshold, – indicates down-regulated and fold change <= 1 / threshold, - indicates down-regulated but fold change > 1 / threshold, '?' indicates not differentially expressed.

Parameters

- **data** (*anndata* object) – An *anndata* object containing the expression matrix and differential expression results.
- **gene_list** (*list[str]*) – A list of gene symbols.
- **test** (*str*, optional (default: *fisher*)) – Differential expression test to look at, could be either *t*, *fisher* or *mwu*.
- **thre** (*float*, optional (default: *1.5*)) – Fold change threshold to determine if the marker is a strong DE (++ or –) or weak DE (+ or -).

Returns A data frame containing marker differential expression results for each cluster.

Return type *pandas.DataFrame*

Examples

```
>>> results = misc.search_de_genes(data, ['CD3E', 'CD4', 'CD8'], test = 'fisher',␣
↪thre = 2.0)
```

1.4 References

INDEX

A

aggregate_10x_matrices() (in module tools), 15
annotate_clusters() (in module annotate_cluster), 20

C

collect_variable_gene_matrix() (in module tools), 18
correct_batch_effects() (in module tools), 18

E

estimate_adjustment_matrices() (in module tools), 18

F

filter_data() (in module tools), 17
filter_genes_dispersion() (in module tools), 18

G

get_anndata_for_subclustering() (in module tools), 17

L

log_norm() (in module tools), 17

P

plot_composition() (in module plotting), 20
plot_heatmap() (in module plotting), 21
plot_scatter() (in module plotting), 21
plot_scatter_gene_groups() (in module plotting), 21
plot_scatter_genes() (in module plotting), 21
plot_scatter_groups() (in module plotting), 21

R

read_input() (in module tools), 16
run_approximated_louvain() (in module tools), 19
run_de_analysis() (in module tools), 19
run_diffmap() (in module tools), 18
run_fitsne() (in module tools), 19
run_force_directed_layout() (in module tools), 19
run_hdbscan() (in module tools), 18
run_kmeans() (in module tools), 19
run_louvain() (in module tools), 18
run_pca() (in module tools), 17
run_pseudotime_calculation() (in module tools), 18
run_rpca() (in module tools), 17
run_tsne() (in module tools), 19
run_umap() (in module tools), 19

S

scatter() (in module plotting), 22
scatter3d() (in module plotting), 22
scatter3d_real() (in module plotting), 22
scatter_real() (in module plotting), 22
search_de_genes() (in module misc), 23
search_genes() (in module misc), 22
set_group_attribute() (in module tools), 17

U

update_var_names() (in module tools), 17