episcanpy Documentation

Release 0.2.0+66.g32c2282

Anna Danese

Aug 07, 2020

CONTENTS

EpiScanpy is a toolkit to analyse single-cell open chromatin (scATAC-seq) and single-cell DNA methylation (for example scBS-seq) data. **EpiScanpy** is the epigenomic extension of the very popular scRNA-seq analysis tool **Scanpy** (Genome Biology, 2018) [Wolf18]. For more information, read scanpy documentation.

The documentation for epiScanpy is available here. If epiScanpy is useful to your research, consider citing epiScanpy.

Report issues and access the code on GitHub.

Note: Also see the release notes of `scanpy`.

Also see the release notes of `anndata`.

CHAPTER

ONE

VERSION 0.2.0 AUGUST 7, 2020

This release deal with the compatibility problems with the latest version of scanpy. Additionally, it contains new features to build quick custom count matrices (bld_mtx_fly), to convert snap into h5ad files (snap2anndata) or build gene activity matrices (geneactivity).

CHAPTER

TWO

VERSION 0.1.8 NOVEMBER 5, 2019

Release new processing function & quality controls.

CHAPTER

THREE

VERSION 0.1.7 NOVEMBER 5, 2019

Release for SCOG epiScanpy Hackathon in Saarbrucken.

This version is not fully compatible with previous version.

CHAPTER

FOUR

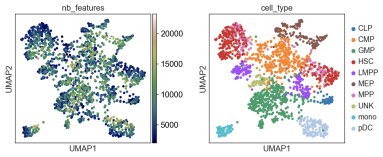
VERSION 0.1.0 MAY 10, 2019

Initial release.

4.1 Tutorials

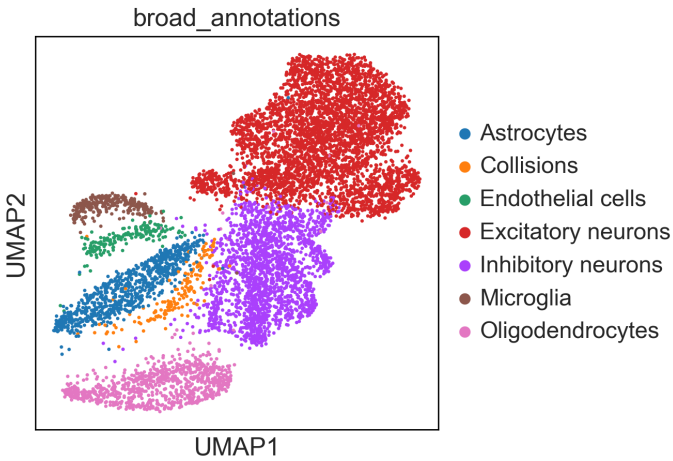
4.1.1 Single cell ATAC-seq

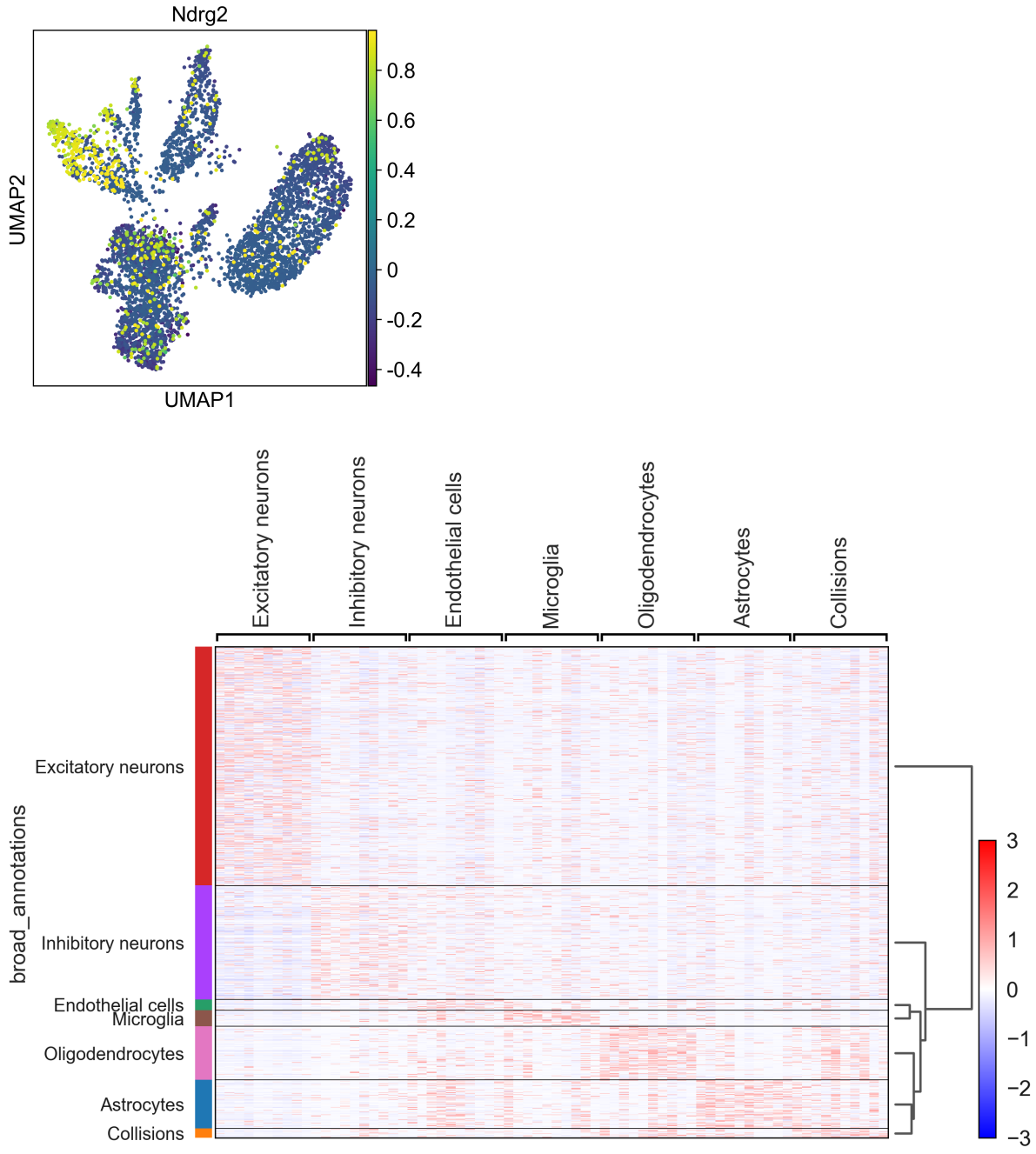
To get started, we recommend epiScanpy's analysis pipeline for scATAC-seq data from Buenrostro et al. [Buenrostro18]. , the dataset consist of ~3000cells of human PBMCs. This tutorial focuses on preprocessing, clustering, identification of cell types via known marker genes and trajectory inference. The tutorial can be found here.



If you want to see how to build count matrices from ATAC-seq bam files for different set of annotations (like enhancers). The tutorial can be found here.

Soon available, there will be a tutorial providing a function to very quickly build custom count matrices using standard 10x single cell ATAC output.





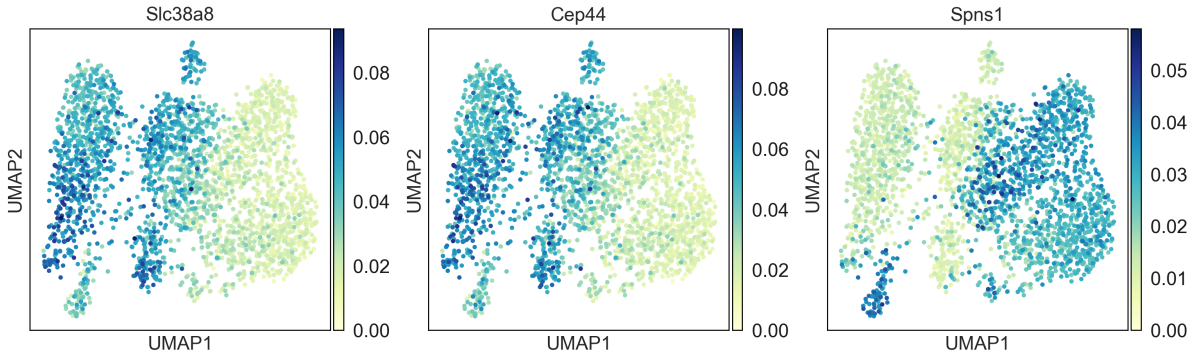
An additional tutorial on processing and clustering count matrices from the Cusanovich mouse scATAC-seq atlas [Cusanovich18].. Here.

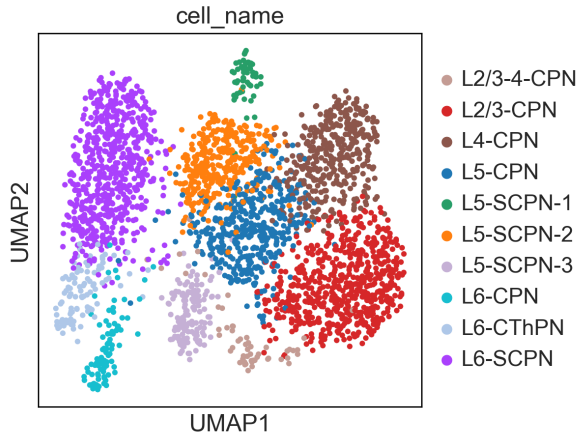
4.1.2 Single cell DNA methylation

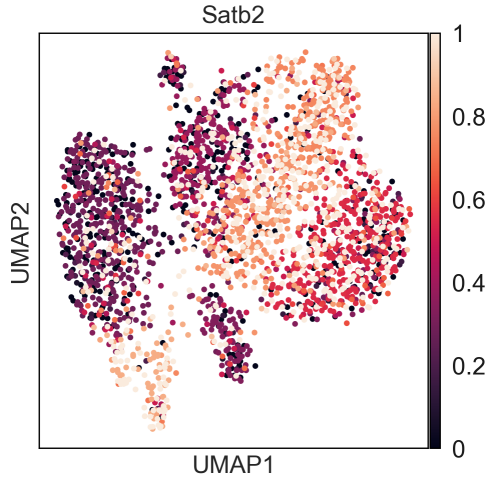
Here you can find a tutorial for the preprocessing, clustering and identification of cell types for single-cell DNA methylation data using the publicly available data from Luo et al. [Luo17].

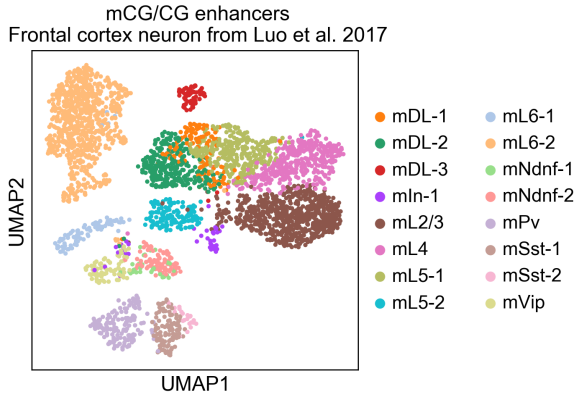
The first tutorial shows how to build the count matrices for the different feature spaces (windows, promoters) in different cytosine contexts. Here is the tutorial.

Then, there is a second tutorial on how to use them and compare the results. The data used comes from mouse brain (frontal cortex). It will be available very soon.









4.2 Usage Principles

Import the epiScanpy API as:

```
import episcanpy.api as epi
import anndata as ad
```

4.2.1 Workflow

The first step is to build the count matrix. Because single-cell epigenomic data types have different characteristics (count data in ATAC-seq versus methylation level in DNA methylation, for example), epiScanpy implements -omic specific approaches to build the count matrix. All the functions to build the count matrices (for ATAC, methylation or other) will use `epi.ct` (ct = count).

The first step is to load an annotation and then build the count matrix that will be either methylation or ATAC-seq specific. For example using `epi.ct`, e.g.:

```
epi.ct.load_features(file_features, **tool_params)  # to load annotation files
epi.ct.build_count_mtx(cell_file_names, omic="ATAC") # to build the ATAC-seq count␣
↪matrix
```

If you have an already build matrix, you can load it with any additional metadata (such as cell annotations or batches).

The count matrix, either the one that has been constructed or uploaded, with any additional informations (such as cell annotations or batches) are stored as an `AnnData` object. All functions for quality control and preprocessing are called using `epi.pp` (pp = preprocessing).

To visualise how common features are and what is the coverage distribution of the count matrix features, use:

```
epi.pp.commoness_features(adata, **processing_params)
epi.pp.coverage_cells(adata, **processing_params)
```

To remove low quality cells you can use the following functions:

```
epi.pp.filter_cells(adata, min_features=10)
epi.pp.filter_features(adata, min_cells=10)
```

To reduce the feature space to the most variable features: :: epi.pl.cal_var(adata) epi.pp.select_var_feature(adata, max_score=0.2, nb_features=50000)

The next step, is the calculation of tSNE, UMAP, PCA etc. For that, we take advantage of the embedding into Scanpy and we use mostly Scanpy functions, which are called using `sc.tl` (tl = tool) [Wolf18]. For that, see Scanpy usage principles: <https://scanpy.readthedocs.io/en/latest/basic_usage.html>`__. For example, to obtain cell-cell distance calculations or low dimensional representation we make use of the `adata` object, and store *n_obs* observations (cells) of *n_vars* variables (expression, methylation, chromatin features). For each tool, there typically is an associated plotting function in `sc.tl` and `sc.pl` (pl = plot)

```
epi.pp.pca(adata, n_comps=100, svd_solver='arpack')
epi.pp.neighbors(adata,  n_neighbors=15)

epi.tl.tsne(adata, **tool_params)
epi.pl.tsne(adata, **plotting_params)
```

There are also epiScanpy specific tools and plotting functions that can be accessed using `epi.tl` and `epi.pl`

```
epi.tl.silhouette(adata, **tool_params)
epi.pl.silhouette(adata, **plotting_params)
epi.pl.prct_overlap(adata, **plotting_params)
```

4.2.2 Data structure

Similarly to Scanpy, the methylation and ATAC-seq matrices are stored as Anndata objects. For more information on the datastructure see here`here <https://anndata.readthedocs.io/en/latest/>`__

4.3 System Requirements

4.3.1 Hardware requirements

`epiScanpy` package requires only a standard computer with enough RAM to support the in-memory operations.

4.3.2 Software requirements

OS Requirements This package is supported for *macOS* and *Linux*. The package has been tested on the following systems: + macOS: Mojave & Catalina (10.14 to 10.15.4)

4.3.3 Python Dependencies

EpiScanpy require a working version of `Python` (>= 3.6)

Additionally, this package `epiScanpy` depends on other Python dependencies and packages.:

```
anndata
matplotlib
numpy
pandas
pyliftOver
pysam
scanpy
scipy
scikit-learn
```

(continues on next page)

(continued from previous page)

```
seaborn
bamnostic
```

4.4 Installation

4.4.1 Anaconda

If you do not have a working Python 3.5 or 3.6 installation, consider installing Miniconda (see Installing Miniconda). Then run:

```
conda install seaborn scikit-learn statsmodels numba
conda install -c conda-forge python-igraph louvain
conda create -n scanpy python=3.6 scanpy
```

Finally, run:

```
conda install -c annadanese episcanpy
```

Pull epiScanpy from PyPI (consider using `pip3` to access Python 3):

```
pip install episcanpy
```

4.4.2 Github

you can also install epiScanpy directly from Github:

```
pip install git+https://github.com/colomemaria/epiScanpy
```

4.5 API

Import epiScanpy's high-level API as:

```
import episcanpy.api as epi
```

4.5.1 Count Matrices: CT

Loading data, loading annotations, building count matrices, filtering of lowly covered methylation variables. Filtering of lowly covered cells.

Building count matrices

Quickly build a count matrix from tsv/tbi file.

ct.bld_mtx_fly(tsv_file, annotation[, ...])	Building count matrix on the fly.

episcanpy.ct.bld_mtx_fly

Load features

In order to build a count matrix for either methylation or open chromatin data, loading the segmentation of the genome of interest or the set of features of interest is a prerequirement.

ct.load_features(file_features[, ...])	The function load features is here to transform a bed file into a usable set of units to measure methylation levels.
ct.make_windows(size[, chromosomes, max_length])	Generate windows/bins of the given size for the appropriate genome (default choice is human).
ct.size_feature_norm(loaded_feature, size)	If the features loaded are too smalls or of different sizes, it is possible to normalise them to a unique given size by extending the feature coordinate in both directions.
ct.plot_size_features(loaded_feature[, ...])	Plot the different feature sizes in an histogram.
ct.name_features(loaded_features)	Extract the names of the loaded features, specifying the chromosome they originated from.

episcanpy.ct.load_features

episcanpy.ct.make_windows

episcanpy.ct.size_feature_norm

episcanpy.ct.plot_size_features

episcanpy.ct.name_features

Reading methylation file

Functions to read methylation files, extract methylation and buildthe count matrices:

ct.build_count_mtx(cells, annotation[, ...])	Build methylation count matrix for a given annotation.
ct.read_cyt_summary(sample_name, meth_type, ...)	Read file from which you want to extract the methylation level and (assuming it is like the Ecker/Methylpy format) extract the number of methylated read and the total number of read for the cytosines covered and in the right genomic context (CG or CH) :param sample_name: name of the file to read to extract key information.
ct.load_met_noimput(matrix_file[, path, save])	read the raw count matrix and convert it into an AnnData object.

episcanpy.ct.build_count_mtx

episcanpy.ct.read_cyt_summary

episcanpy.ct.load_met_noimput

Reading open chromatin(ATAC) file

ATAC-seq specific functions to build count matrices and load data:

ct.bld_atac_mtx(list_bam_files, loaded_feat)	Build a count matrix one set of features at a time.
ct.save_sparse_mtx(initial_matrix[, ...])	Convert regular atac matrix into a sparse Anndata:

episcanpy.ct.bld_atac_mtx

episcanpy.ct.save_sparse_mtx

General functions

Functions non -omic specific:

ct.save_sparse_mtx(initial_matrix[, ...])	Convert regular atac matrix into a sparse Anndata:

4.5.2 Preprocessing: PP

Imputing missing data (methylation), filtering lowly covered cells or variables, correction for batch effect.

pp.coverage_cells(adata[, key_added, log, ...])	Histogram of the number of open features (in the case of ATAC-seq data) per cell.
pp.correlation_pc(adata, variable[, pc, ...])	Correlation between a given PC and a covariate.
pp.coverage_features(adata[, binary, log, ...])	Display how often a feature is measured as open (for ATAC-seq).
pp.density_features(adata[, threshold, ...])	Display how often a feature is measured as open (for ATAC-seq).
pp.select_var_feature(adata[, min_score, ...])	This function computes a variability score to rank the most variable features across all cells.
pp.cal_var(adata[, show, color, save])	Show distribution plots of cells sharing features and variability score.
pp.variability_features(adata[, min_score, ...])	This function computes a variability score to rank the most variable features across all cells.
pp.binarize(adata[, copy])	convert the count matrix into a binary matrix.
pp.lazy(adata[, pp_pca, svd_solver, nb_pcs, ...])	Automatically computes PCA coordinates, loadings and variance decomposition, a neighborhood graph of observations, t-distributed stochastic neighborhood embedding (tSNE) Uniform Manifold Approximation and Projection (UMAP)
pp.load_metadata(adata, metadata_file[, ...])	Load observational metadata in adata.obs.

continues on next page

Table 6 – continued from previous page

pp.read_ATAC_10x(matrix[, cell_names, ...])	Load sparse matrix (including matrices corresponding to 10x data) as AnnData objects.
pp.filter_cells(adata[, min_counts, ...])	Filter cell outliers based on counts and numbers of genes expressed.
pp.filter_features(data[, min_counts, ...])	Filter features based on number of cells or counts.
pp.normalize_total(adata[, target_sum, ...])	Normalize counts per cell.
pp.pca(adata[, n_comps, zero_center, ...])	Principal component analysis [Pedregosa11].
pp.normalize_per_cell(adata[, ...])	Normalize total counts per cell.
pp.regress_out(adata, keys[, n_jobs, copy])	Regress out unwanted sources of variation.
pp.subsample(data[, fraction, n_obs, ...])	Subsample to a fraction of the number of observations.
pp.downsample_counts(adata[, ...])	Downsample counts from count matrix.
pp.neighbors(adata[, n_neighbors, n_pcs, ...])	Compute a neighborhood graph of observations [McInnes18].
pp.sparse(adata[, copy])	Transform adata.X from a matrix or array to a csc sparse matrix.
pp.sparse(adata[, copy])	Transform adata.X from a matrix or array to a csc sparse matrix.

episcanpy.pp.coverage_cells

episcanpy.pp.correlation_pc

episcanpy.pp.coverage_features

episcanpy.pp.density_features

episcanpy.pp.select_var_feature

episcanpy.pp.cal_var

episcanpy.pp.variability_features

episcanpy.pp.binarize

episcanpy.pp.lazy

episcanpy.pp.load_metadata

episcanpy.pp.read_ATAC_10x

episcanpy.pp.filter_cells

episcanpy.pp.filter_features

episcanpy.pp.normalize_total

episcanpy.pp.pca

episcanpy.pp.normalize_per_cell

episcanpy.pp.regress_out

episcanpy.pp.subsample

episcanpy.pp.downsample_counts

episcanpy.pp.neighbors

episcanpy.pp.sparse

Methylation matrices

Methylation specific count matrices.

pp.imputation_met(adata[, ...])	Impute missing values in methyaltion level matrices.
pp.load_met_noimput(matrix_file[, path, save])	read the raw count matrix and convert it into an AnnData object.
pp.readandimputematrix(file_name[, min_coverage])	Temporary function to load and impute methyaltion count matrix into an AnnData object

episcanpy.pp.imputation_met

episcanpy.pp.load_met_noimput

episcanpy.pp.readandimputematrix

4.5.3 Tools: TL

tl.rank_features(adata, groupby[, omic, ...])	It is a wrap-up function of scanpy sc.tl.rank_genes_groups function.
tl.lazy(adata[, pp_pca, copy])	Automatically computes PCA coordinates, loadings and variance decomposition, a neighborhood graph of observations, t-distributed stochastic neighborhood embedding (tSNE) Uniform Manifold Approximation and Projection (UMAP)
tl.load_markers(path, marker_list_file)	Convert list of known cell type markers from literature to a dictionary Input list of known marker genes First row is considered the header
tl.identify_cluster(adata, cell_type, ...[, ...])	Use markers of a given cell type to plot peak openness for peaks in promoters of the given markers Input cell type, cell type markers, peak promoter intersections
tl.top_feature_genes(adata, gtf_file[, ...])	Deprecated - Please use epi.tl.var_features_to_genes instead.
tl.var_features_to_genes(adata, gtf_file[, ...])	Once you called the most variable features.
tl.geneactivity(adata, gtf_file[, ...])	merge values of peaks/windows/features overlapping genebodies + 2kb upstream.
tl.diffmap(adata[, n_comps, copy])	Diffusion Maps [Coifman05] [Haghverdi15] [Wolf18].

continues on next page

Table 8 – continued from previous page

tl.draw_graph(adata[, layout, init_pos, ...])	Force-directed graph drawing [Islam11] [Jacomy14] [Chippada18].
tl.tsne(adata[, n_pcs, use_rep, perplexity, ...])	t-SNE [Maaten08] [Amir13] [Pedregosa11].
tl.umap(adata[, min_dist, spread, ...])	Embed the neighborhood graph using UMAP [McInnes18].
tl.dpt(adata[, n_dcs, n_branchings, ...])	Infer progression of cells through geodesic distance along the graph [Haghverdi16] [Wolf19].
tl.louvain(adata[, resolution, ...])	Cluster cells into subgroups [Blondel08] [Levine15] [Traag17].
tl.leiden(adata[, resolution, restrict_to, ...])	Cluster cells into subgroups [Traag18].
tl.kmeans(adata, num_clusters)	Compute kmeans clustering using X_pca fits.
tl.hc(adata, num_clusters)	Compute hierarchical clustering using X_pca fits.
tl.getNClusters(adata, n_cluster[, ...])	Function will test different settings of louvain to obtain the target number of clusters.
tl.dendogram(adata, groupby[, n_pcs, ...])	Computes a hierarchical clustering for the given *groupby* categories.
tl.ARI(adata, label_1, label_2)	Compute Adjusted Rand Index.
tl.AMI(adata, label_1, label_2)	Compute adjusted Mutual Info.
tl.homogeneity(adata, label_1, label_2)	Compute homogeneity score.
tl.silhouette(adata_name, cluster_annot[, ...])	Compute silhouette scores.

episcanpy.tl.rank_features

episcanpy.tl.lazy

episcanpy.tl.load_markers

episcanpy.tl.identify_cluster

episcanpy.tl.top_feature_genes

episcanpy.tl.var_features_to_genes

episcanpy.tl.geneactivity

episcanpy.tl.diffmap

episcanpy.tl.draw_graph

episcanpy.tl.tsne

episcanpy.tl.umap

episcanpy.tl.dpt

episcanpy.tl.louvain

episcanpy.tl.leiden

episcanpy.tl.kmeans

episcanpy.tl.hc

episcanpy.tl.getNClusters

episcanpy.tl.dendogram

episcanpy.tl.ARI

episcanpy.tl.AMI

episcanpy.tl.homogeneity

episcanpy.tl.silhouette

4.5.4 Plotting: PL

The plotting module `episcanpy.plotting` largely parallels the `tl.*` and a few of the `pp.*` functions. For most tools and for some preprocessing functions, you'll find a plotting function with the same name.

`pl.pca`(adata, basis, *[, color, ...])	Scatter plot in PCA coordinates.
`pl.pca_overview`(adata[, color, use_raw, ...])	Plot PCA results.
`pl.pca_variance_ratio`(adata[, n_pcs, log, ...])	Plot the variance ratio.
`pl.tsne`(adata, basis, *[, color, ...])	Scatter plot in tSNE basis.
`pl.umap`(adata, basis, *[, color, ...])	Scatter plot in UMAP basis.
`pl.rank_feat_groups`(adata[, groups, ...])	Plot ranking of features.
`pl.rank_feat_groups_violin`(adata[, groups, ...])	Plot ranking of features for all tested comparisons.
`pl.rank_feat_groups_dotplot`(adata[, groups, ...])	Plot ranking of features using dotplot plot (see `dotplot()`)
`pl.rank_feat_groups_stacked_violin`(adata[, ...])	Plot ranking of features using stacked_violin plot (see `stacked_violin()`)
`pl.rank_feat_groups_matrixplot`(adata[, ...])	Plot ranking of features using matrixplot plot (see `matrixplot()`)
`pl.rank_feat_groups_heatmap`(adata[, groups, ...])	Plot ranking of features using heatmap plot (see `heatmap()`)
`pl.rank_feat_groups_tracksplot`(adata[, ...])	Plot ranking of features using heatmap plot (see `heatmap()`)
`pl.cal_var`(adata[, show, color, save])	Show distribution plots of cells sharing features and variability score.
`pl.violin`(adata, keys[, groupby, log, ...])	Violin plot.
`pl.scatter`(adata[, x, y, color, use_raw, ...])	Scatter plot along observations or variables axes.
`pl.ranking`(adata, attr, keys[, dictionary, ...])	Plot rankings.
`pl.clustermap`(adata[, obs_keys, use_raw, ...])	Hierarchically-clustered heatmap.
`pl.heatmap`(adata, var_names[, groupby, ...])	Heatmap of the expression values of genes.
`pl.dotplot`(adata, var_names[, groupby, ...])	Makes a *dot plot* of the expression values of *var_names*.
`pl.matrixplot`(adata, var_names[, groupby, ...])	Creates a heatmap of the mean expression values per cluster of each var_names If groupby is not given, the matrixplot assumes that all data belongs to a single category.

continues on next page

Table 9 – continued from previous page

pl.tracksplot(adata, var_names, groupby[, ...])	In this type of plot each var_name is plotted as a filled line plot where the y values correspond to the var_name values and x is each of the cells.
pl.dendrogram(adata, groupby[, ...])	Plots a dendrogram of the categories defined in *groupby*.
pl.correlation_matrix(adata, groupby[, ...])	Plots the correlation matrix computed as part of *sc.tl.dendrogram*.
pl.prct_overlap(adata, key_1, key_2[, norm, ...])	% or cell count corresponding to the overlap of different cell types between 2 set of annotations/clusters.
pl.overlap_heatmap(adata, key_1, key_2[, ...])	Heatmap of the cluster correspondance between 2 set of annaotations.
pl.cluster_composition(adata, cluster, condition)	
pl.silhouette(adata_name, cluster_annot[, ...])	Plot the product of tl.silhouette as a silhouette plot
pl.silhouette_tot(adata_name, cluster_annot)	Both compute silhouette scores and plot it.
pl.cal_var(adata[, show, color, save])	Show distribution plots of cells sharing features and variability score.
pl.variability_features(adata[, min_score, ...])	This function computes a variability score to rank the most variable features across all cells.

episcanpy.pl.pca

episcanpy.pl.pca_overview

episcanpy.pl.pca_variance_ratio

episcanpy.pl.tsne

episcanpy.pl.umap

episcanpy.pl.rank_feat_groups

episcanpy.pl.rank_feat_groups_violin

episcanpy.pl.rank_feat_groups_dotplot

episcanpy.pl.rank_feat_groups_stacked_violin

episcanpy.pl.rank_feat_groups_matrixplot

episcanpy.pl.rank_feat_groups_heatmap

episcanpy.pl.rank_feat_groups_tracksplot

episcanpy.pl.cal_var

episcanpy.pl.violin

episcanpy.pl.scatter

episcanpy.pl.ranking

episcanpy.pl.clustermap

episcanpy.pl.heatmap

episcanpy.pl.dotplot

episcanpy.pl.matrixplot

episcanpy.pl.tracksplot

episcanpy.pl.dendrogram

episcanpy.pl.correlation_matrix

episcanpy.pl.prct_overlap

episcanpy.pl.overlap_heatmap

episcanpy.pl.cluster_composition

episcanpy.pl.silhouette

episcanpy.pl.silhouette_tot

episcanpy.pl.variability_features

4.6 References

BIBLIOGRAPHY

[Angerer16] Angerer *et al.* (2016), *destiny – diffusion maps for large-scale single-cell data in R*, Bioinformatics.

[Cusanovich18] Cusanovich, D. A. et al. A Single-Cell Atlas of In Vivo Mammalian Chromatin Accessibility. Cell 174, 1309–1324.e18 (2018).

[Luo17] Luo, C. et al. Single-cell methylomes identify neuronal subtypes and regulatory elements in mammalian cortex. Science 357, 600–604 (2017).

[Wolf18] Wolf, F. A., Angerer, P. & Theis, F. J. SCANPY: large-scale single-cell gene expression data analysis. Genome Biol. 19, 15 (2018).

[Buenrostro18] Buenrostro J. D. et al. Integrated Single-Cell Analysis Maps the Continuous Regulatory Landscape of Human Hematopoietic Differentiation. Cell 173 (2018)

PYTHON MODULE INDEX

e

INDEX

E

M