cellhub

Release 0.1

Sansom lab

Apr 22, 2022

CONTENTS

CHAPTER

ONE

INSTALLATION

1.1 Installation

1.1.1 Dependencies

Core dependencies include:

- Cellranger (from 10X Genomics) >= 6.0
- Python3
- R >= 4.0
- (Latex)

1.1.2 Installation

1. Install the cgat-core pipeline system following the instructions here https://github.com/cgat-developers/cgat-core/.
2. Clone and install the cellhub-devel repository e.g.:

```
git clone https://github.com/COMBATOxford/cellhub-devel.git
cd cellhub-devel
python setup.py develop
```

Note: Running "python setup.py develop" is necessary to allow pipelines to be launched via the "cellhub" command.

3. In the same virtual or conda environment as cgat-core install the required python packages:

```
pip install -r cellhub-devel/python/requirements.txt
```

4. To install the required R packages (the "BiocManager" and "devtools" libraries must be pre-installed):

```
Rscript cellhub-devel/R/install.packages.R
```

CHAPTER

TWO

OVERVIEW

2.1 Workflow Overview

2.1.1 Philosophy

Cellhub is designed to efficiently parallelise the processing of large datasets. Once processed different data slices can be easily extracted directly from the original matrices, aligned and exported for downstream analysis. At the heart of operations is an sqlite database which warehouses the experiment metadata and per-cell statistics.

The workflow can be divided into eight main steps.

2.1.2 1. Quantitation of per-channel libraries

The workflow begins with *pipeline_cellranger_multi.py*. Input 10X capture channel library identifiers "library_id" and their associated fastq files are specified in the pipeline_cellranger_multi.yml configuration file. The reads from the different libraries will be mapped in parallel.

Note: The channel library is considered to be the fundamental "batch" unit of a 10X experiment. Cells captured from the same channel are exposed to the same ambient RNA. Separate genomic libraries are prepared for each 10x channel.

- It is recommend to inspect patterns of ambient RNA using *pipeline_ambient_rna.py*.
- Per-channel velocity matrices can be prepared using *pipeline_velocity.py*.
- Cell identification can also be performed with *pipeline_emptydrops.py*.

2.1.3 2. Computation of per-cell statistics

Per-cell statistics are computed in parallel for each channel library using *pipeline_cell_qc.py*. The pipeline computes various statistics including standard metrics such as percentage of mitochondrial reads, numbers of UMIs and numbers of genes per cell. In addition it can compute scores for custom genesets.

Note: all per-channel matrices containing computed cell statistics are required to contain "library_id" and "barcode_id" columns.

Note: file names of the per-channel matrices are specified as "library_id.tsv.gz" (matrices for different analyses such as e.g. qcmetrics and scrublet scores are written to separate folders).

2.1.4 3. Cell demultiplexing [optional]

If samples have been multiplexed within channels either genetically or using hash tags a table of barcode_id -> sample_id assignments are prepared using pipeline_demux.py [not yet written].

2.1.5 4. Preparation of the cell database

The library and sample metadata, per cell statistics (and demultiplex assignments) are loaded into an sqlite database using *pipeline_celldb.py*. The pipeline creates a view called "final" which contains the qc and metadata needed for cell selection and downstream analysis.

Note: The user is required to supply a tab-separated sample metadata file (e.g. "samples.tsv") via a path in the pipeline_celldb.yml configuration file. It should have columns for library_id, sample_id as well as any other relevant experimental metadata such as condition, genotype, age, replicate, sex etc.

2.1.6 5. Fetching of cells for downstream analysis

Cells are fetched using *pipeline_fetch_cells.py*. The user specifies the cells that they wish to retrieve from the "final" table (see step 4) via an sql statement in the pipeline_fetch_cells.yml configuration file. The pipeline will extract the cells and metadata from the original matrices and combine them into market matrices and anndata objects for downstream analyses.

It is recommended to fetch cells into a new directory. By design fetching of a single dataset per-directory is supported.

The pipeline supports fetching of velocity information.

Note: The retrieved metadata will include a "sample_id" column. From this point onwards it is natural to think of the "sample_id" as the unit of interest. The "library_ids" remain in the metadata along with all the qc statistics to facilitate downstream investigation of batch effects and cell quality.

2.1.7 6. ADT normalization [optional]

If samples included the ADT modality, *pipeline_adt_norm.py* normalizes the antibody counts for the high-quality fetched cells in the previous step. Normalized ADT can be then used for downstream integration. The pipeline implements 3 normalization methodologies: DSB, median-based, and CLR. The user can specify the feature space.

2.1.8 7. Integration

Alignment of samples is performed with *pipeline_integration.py*. Currently the pipeline supports harmony, bbknn and scanorama. It will produce UMAPs summarising the alignments and will compute the LISI statistic.

Note: The user is required to supply a tsv file (e.g. "integration.tsv") containing the path to the pipeline_fetch_cells.py anndata object that holds the data which is to be integrated. The path to the tsv file is specified in the "pipeline_integration.yml" configuration file.

Warning: pipeline_integration.py will be moving to a new sansomlab/scxl repository (along with pipeline_scxl from sansomlab/tenx).

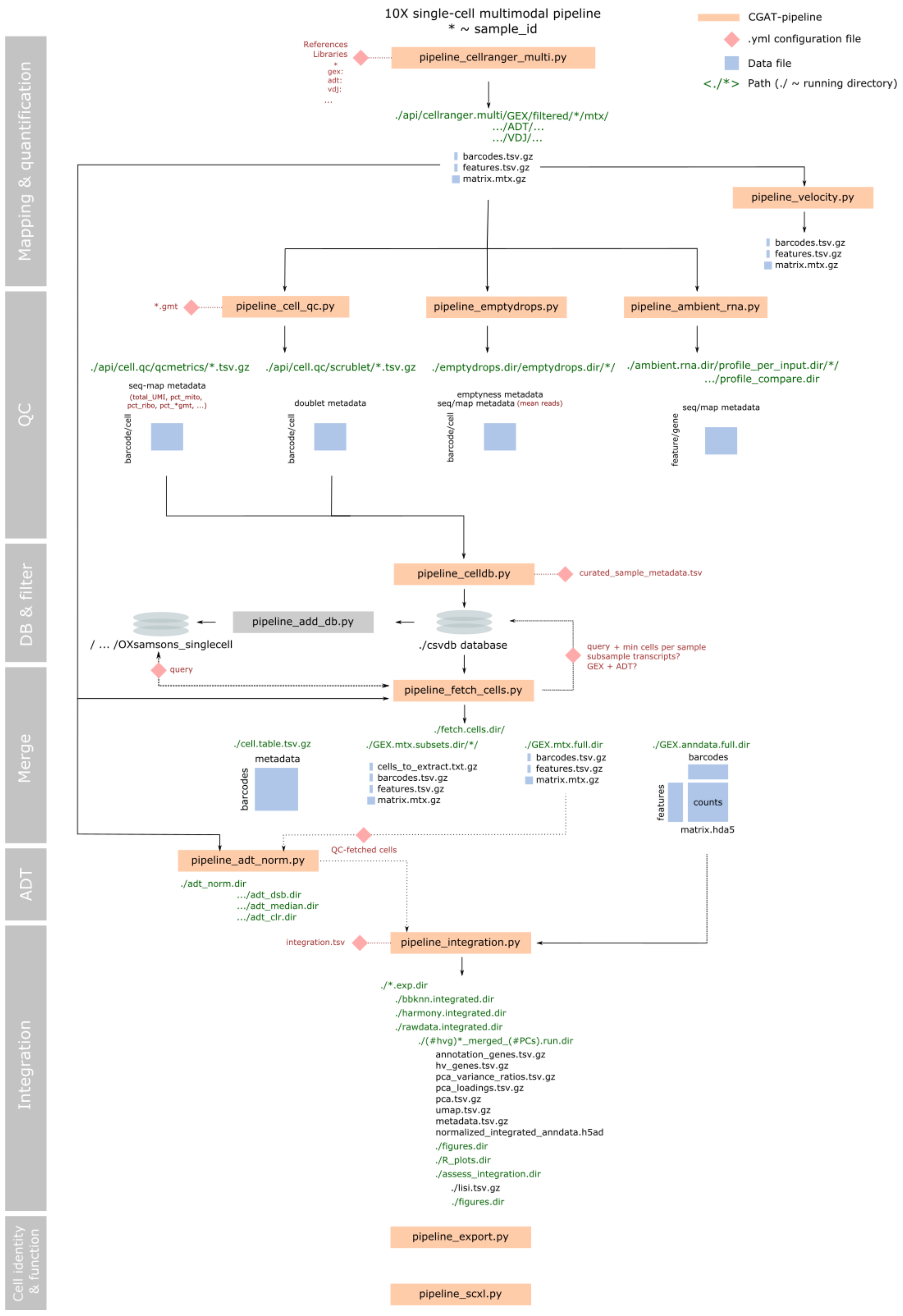
2.1.9 8. Export for seurat [optional]

The integration pipeline outputs an anndata object suitable for analysis with scanpy. A Seurat object can be prepared using *pipeline_export.py*.

Warning: pipeline_export.py will also be moving to the new sansomlab/scxl repository.

2.2 Workflow Diagram

The diagram is now a little out of date with respect to configuration of the pipeline inputs but provides a useful depiction of the overall workflow.


10X single-cell multimodal pipeline
* ~ sample_id
CGAT-pipeline
.yml configuration file
Data file
<./*> Path (./ ~ running directory)
Mapping & quantification
QC
DB & filter
Merge
ADT
Integration
Cell identity & function
References
Libraries
*
gex:
adt:
vdj:
...
pipeline_cellranger_multi.py
./api/cellranger.multi/GEX/filtered/*/mtx/
.../ADT/...
.../VDJ/...
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
pipeline_velocity.py
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
*.gmt
pipeline_cell_qc.py
pipeline_emptydrops.py
pipeline_ambient_rna.py
./api/cell.qc/qcmetrics/*.tsv.gz
./api/cell.qc/scrublet/*.tsv.gz
./emptydrops.dir/emptydrops.dir/*/
./ambient.rna.dir/profile_per_input.dir/*/
.../profile_compare.dir
seq-map metadata
(total_UMI, pct_mito, pct_ribo, pct_*gmt, ...)
barcode/cell
doublet metadata
barcode/cell
emptyness metadata
seq/map metadata (mean reads)
barcode/cell
seq/map metadata
feature/gene
pipeline_celldb.py
curated_sample_metadata.tsv
pipeline_add_db.py
/ ... /OXsamsons_singlecell
./csvdb database
query + min cells per sample
subsample transcripts?
GEX + ADT?
query
pipeline_fetch_cells.py
./fetch.cells.dir/
./cell.table.tsv.gz
metadata
barcodes
./GEX.mtx.subsets.dir/*/
cells_to_extract.txt.gz
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
./GEX.mtx.full.dir
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
./GEX.anndata.full.dir
barcodes
features
counts
matrix.hda5
QC-fetched cells
pipeline_adt_norm.py
./adt_norm.dir
.../adt_dsb.dir
.../adt_median.dir
.../adt_clr.dir
integration.tsv
pipeline_integration.py
./*.exp.dir
./bbknn.integrated.dir
./harmony.integrated.dir
./rawdata.integrated.dir
./(#hvg)*_merged_(#PCs).run.dir
annotation_genes.tsv.gz
hv_genes.tsv.gz
pca_variance_ratios.tsv.gz
pca_loadings.tsv.gz
pca.tsv.gz
umap.tsv.gz
metadata.tsv.gz
normalized_integrated_anndata.h5ad
./figures.dir
./R_plots.dir
./assess_integration.dir
./lisi.tsv.gz
./figures.dir
pipeline_export.py
pipeline_scxl.py

2.3 Usage

2.3.1 Configuring and running pipelines

Run the cellhub –help command to view the help documentation and find available pipelines to run cellhub.

The cellhub pipelines are written using cgat-core pipelining system. From more information please see the CGAT-core paper. Here we illustrate how the pipelines can be run using the cellranger_multi pipeline as an example.

Following installation, to find the available pipelines run:

```
cellhub -h
```

Next generate a configuration yml file:

```
cellhub cellranger_multi config -v5
```

To fully run the example cellhub pipeline run:

```
cellhub cellranger_multi make full -v5
```

However, it may be best to run the individual tasks of the pipeline to get a feel of what each task is doing. To list the pipline tasks and their current status, use the 'show' command:

```
cellhub cellranger_multi show
```

Individual tasks can then be executed by name, e.g.

```
cellhub cellranger_multi make cellrangerMulti -v5
```

Note: If any upstream tasks are out of date they will automatically be run before the named task is executed.

2.3.2 Getting Started

To get started please see the *IFNb example*.

CHAPTER

THREE

EXAMPLES

3.1 IFNb PBMC example

3.1.1 Setting up

1. Clone the example template to a local folder

Clone the folders and files for the example into a local folder.

cp -r /path/to/cellhub/examples/ifnb_pbmc/* .

This will create 3 folders:

- "cellhub" where the preprocessing pipelines will be run and where the cellhub database will be created
- "integration" where data for cells of interest (e.g. passing qc) will be fetched and integrated
- "scxl" where we can perform downstream analysis with pipeline_scxl.py.

3.1.2 Running the pre-processing pipelines and creating the database

2. Running Cellranger

The first step is to configure and run pipeline_cellranger_multi. We already have a pre-configured yml file so we can skip this step but the syntax is included for reference here and also for the other steps:

```
# enter the cellhub directory

cd cellhub

# cellhub cellranger_multi config
```

Edit the pipeline_cellranger_multi.yml file as appropriate to point to folders containing fastq files extracted from the original BAM files submitted by Kang et. al. to GEO (GSE96583). The GEO identifiers are: unstimulated (GSM2560248) and stimulated (GSM2560249). The fastqs can be extracted with the 10X bamtofastq tool.

We run the pipeline as follows:

```
cellhub cellranger_multi make full -v5 -p20
```

If you have not run "python setup.py devel" pipelines can instead be launched directly. In this case the equivalent command would be:

```
python path/to/cellhub-devel/cellhub/pipeline_cellranger_multi.py make full -v5 -p20 -
↪-pipeline-log=pipeline_cellranger_multi.py
```

Note: when launching pipelines directly if the "–pipeline-log" parameter is not specified the log file will be written to "pipeline.log".

3. Running the cell qc pipeline

Next we run the cell qc pipeline:

```
# cellhub cell_qc config

cellhub cell_qc make full -v5 -p20
```

4. Running emptydrops and investigating ambient RNA (optional)

If desired we can run emptydrops:

```
# cellhub emptydrops config

cellhub emptydrops make full -v5 -p20
```

And investigate the ambient rna:

```
# cellhub ambient_rna config

cellhub ambient_rna make full -v5 -p20
```

5. Loading the cell statistics into the celldb

The cell QC statistics and metadata ("samples.tsv") are next loaded into a local sqlite database:

```
# cellhub celldb config

cellhub celldb make full -v5 -p20
```

3.1.3 Performing downstream analysis

6. Fetching cells for downstream analysis

We use pipeline_fetch_cells to retrieve the cells we want for downstream analysis. (QC thresholds and e.g. desired samples are specified in the pipeline_fetch_cells.yml) file:

```
# change into the integration directory
cd ../integration

# cellhub fetch_cells config
cellhub fetch_cells make full -v5 -p20
```

7. Integration

Pipeline_integration supports integration of the data with harmony, bbknn and scanorama. In this example the location of the data is specified in the "integration.tsv" file as per the path given in the "pipeline_integration.yml" file.

```
# cellhub integration config

cellhub integration make full -v5 -p20
```

Warning: pipeline_integration.py will be moving to a new sansomlab/scxl repository (along with pipeline_scxl from sansomlab/tenx).

8. Export for downstream analysis

Finally we can export the integrated anndata object to e.g. a Seurat object for downstream analysis:

```
# cellhub export config

cellhub export make full -v5 -p20
```

Warning: pipeline_export.py will be moving to the new sansomlab/scxl repository

9. Clustering analysis with pipeline_scxl

Cluster analysis can be performed with pipeline_scxl.py. A suitable configuration file for working with the harmony aligned seurat object is provided in the examples/infb_pbmc/scxl/ folder:

```
# change into the sxcl directory
cd ../scxl.dir

# link in the exported Seurat object from step 8
mkdir integrated.seurat.dir
ln -s ../integration/export.dir/pbmc.exp.dir/seurat_object.rds integrated.seurat.dir/
↪begin.rds

# a suitable yml file has been provided so we can now launch the pipeline
python /path/to/tenx/pipelines/pipeline_scxl.py make full -v5 -p200
```

CHAPTER

FOUR

CODING GUIDELINES

4.1 Coding Guidelines

4.1.1 Repository layout

Table 1: Repository layout

Folder	Contents
cellhub	The cellhub Python module which contains the set of CGAT-core pipelines
cellhub/tasks	The cellhub tasks submodule which contains helper functions and pipeline task definitions
Python	Python worker scripts that are executed by pipeline tasks
R	R worker scripts that are executed by pipeline tasks
docsrc	The documentation source files in restructured text format for compilation with sphinx
docs	The compiled documentation
examples	Configuration files for example datasets
conda	Conda environment, TBD
tests	TBD

4.1.2 Coding style

Currently we are working to improve and standardise the coding style:

- Python code should be pep8 compliant. Compliance checks will be enforced soon.
- R code should follow the tidyverse style guide. Please do not use right-hand assignment.
- Arguments to Python scripts should be parsed with argparse.
- Arguments to R scripts should be parsed with optparse or supplied via yaml files.
- Logging in Python scripts should be performed with the standard library “logging” module.
- Logging in R scripts should be performed with the “futile.logger” library.
- If you need to write to stdout from R scripts use message() or warning(). Do not use print() or cat().

4.1.3 Writing pipelines

The pipelines live in the "cellhub" python module.

Auxiliary task functions live in the "cellhub/task" python sub-module.

Note: Tasks of more than a few lines should be abstracted into appropriately named sub-modules.

In the notes below "xxx" denotes the name of a pipeline such as e.g. "cell_qc".

1. Paths should never be hardcoded in the pipelines - rather they must be read from the yaml files.
2. Yaml configuration files should be named pipeline_xxx.yml
3. The output of individual pipelines should be written to a subfolder name "xxx.dir" to keep the root directory clean (it should only contain these directories and the yml configuration files!).
4. Pipelines that generate cell-level information for down-stream analysis must read their inputs from the api and register their outputs to the API, see *API*. If you need information from an upstream pipeline that is not present on the API please raise an issue.
5. We are documenting the pipelines using the sphinx "autodocs" module so please maintain informative rst docstrings.

4.1.4 Cell barcodes

- We use cell barcodes in the format "UMI-library_id".
- Barcodes are set upstream e.g. in the market matrix files that are exposed on the API by pipeline_cellranger_multi.py.
- Downstream pipelines shold not need to manipulate barcodes. If you find barcodes being served on the API in an incorrect format please raise an issue.
- We use the field/column name "barcode_id" for the fields/columns that contains the barcodes in tables and databases.

4.1.5 Yaml configuration file naming

The cgat-core system only supports configuration files name "pipeline.yml".

We work around this by overriding the cgat-core functionality using a helper function in cellhub.tasks.control as follows:

```
import Pipeline as P
import cellhub.tasks.control as C

# Override function to collect config files
P.control.write_config_files = C.write_config_files
```

Default yml files must be located at the path cellhub/yaml/pipeline_xxx.yml

4.1.6 Writing and compiling the documentation

The source files for the documentation are found in:

```
docsrc
```

The documentation source files for the pipelines can be found in:

```
docsrc/pipelines
```

To build the documentation cd to the docsrc folder and run:

```
make github
```

This will build the documentation and copy the latex output to the "docs" folder. You then need to cd to the "docs" folder and run:

```
make
```

To compile the pdf.

When the repo is made public we will switch to using html documentation on readthedocs. Unfortunately there is no straightforward solution for private html hosting.

4.2 API

4.2.1 Overview

The pre-processing pipelines must define and register their outputs via a common api. The *cellhub.tasks.api module* provides the code for doing this. The api comprises of an "api" folder into which pipeline outputs are symlinked (by the "register_dataset" "api" class method).

The api provides a stable and sanitised interface from which the downstream pipelines can access the information.

4.2.2 Example

The “api” folder for the IFNB example dataset looks like this.

```
api
├── cell.qc
│   ├── qcmetrics
│   │   └── filtered
│   │       ├── GSM2560248.tsv.gz -> ../../../../cell.qc.dir/qcmetric.dir/GSM2560248.tsv.gz
│   │       ├── GSM2560249.tsv.gz -> ../../../../cell.qc.dir/qcmetric.dir/GSM2560249.tsv.gz
│   │       └── manifest.yml
│   └── scrublet
│       └── filtered
│           ├── GSM2560248.tsv.gz -> ../../../../cell.qc.dir/scrublet.dir/GSM2560248.tsv.gz
│           ├── GSM2560249.tsv.gz -> ../../../../cell.qc.dir/scrublet.dir/GSM2560249.tsv.gz
│           └── manifest.yml
└── cellranger.multi
    └── GEX
        ├── filtered
        │   ├── GSM2560248
        │   │   └── mtx
        │   │       ├── barcodes.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/GSM2560248/GEX/barcodes.tsv.gz
        │   │       ├── features.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/GSM2560248/GEX/features.tsv.gz
        │   │       ├── manifest.yml
        │   │       └── matrix.mtx.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/GSM2560248/GEX/matrix.mtx.gz
        │   └── GSM2560249
        │       └── mtx
        │           ├── barcodes.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/GSM2560249/GEX/barcodes.tsv.gz
        │           ├── features.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/GSM2560249/GEX/features.tsv.gz
        │           ├── manifest.yml
        │           └── matrix.mtx.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/GSM2560249/GEX/matrix.mtx.gz
        └── unfiltered
            ├── GSM2560248
            │   └── mtx
            │       ├── barcodes.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/unfiltered/GEX/barcodes.tsv.gz
            │       ├── features.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/unfiltered/GEX/features.tsv.gz
            │       ├── manifest.yml
            │       └── matrix.mtx.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/unfiltered/GEX/matrix.mtx.gz
            └── GSM2560249
                └── mtx
                    ├── barcodes.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/unfiltered/GEX/barcodes.tsv.gz
                    ├── features.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/unfiltered/GEX/features.tsv.gz
                    ├── manifest.yml
                    └── matrix.mtx.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/unfiltered/GEX/matrix.mtx.gz
```

4.2.3 Usage

Please see the *cellhub.tasks.api module documentation* for more details.

..note:: If you are writing a pipeline and information that you need from an upstream pipeline has not been exposed on the api please raise an issue.

CHAPTER

FIVE

TASKS MODULE DOCUMENTATION

5.1 API

5.1.1 Overview

This module contains the code for registering and accessing pipeline outputs from a common location.

There are classes that provide methods for:

(1) registering pipeline outputs to the common service endpoint

(2) discovering the information avaliable from the service endpoint (not yet written)

(3) accessing information from the service endpoint (not yet written)

The service endpoint is the folder "api". We use a rest-like syntax for providing access to the pipline outputs.

5.1.2 Usage

Registering ouputs on the service endpoint

Please see *pipeline_cellranger_multi.py* or *pipeline_cell_qc.py* source code for examples.

As an example the code used for registering the qcmetrics outputs is reproduced with some comments here:

```
import cellhub.tasks.api as api

file_set={}

...

# the set of files to be registered is defined as a dictionary
# the keys are arbitrary and will not appear in the api

file_set[library_id] = {"path": tsv_path,
                        "description":"qcmetric table for library " +
↪                        library_id,
                        "format":"tsv"}

# an api object is created, passing the pipeline name
x = api.api("cell.qc")

# the dataset to be deposited is added
x.define_dataset(analysis_name="qcmetrics",
```

(continues on next page)

(continued from previous page)

```
                  data_subset="filtered",
                  file_set=file_set,
                  analysis_description="per library tables of cell GEX qc statistics",
                  file_format="tsv")

# the dataset is linked in to the API
x.register_dataset()
```

Discovering avaliable datasets

TBD.

Accessing datasets

For now datasets can be accessed directly via the "api" endpoint. However in future it will likely be recommended to access them via a subclass of the "read" class (not yet written) which will provide sanity checking.

5.1.3 Class and method documentation

class `cellhub.tasks.api.`**api**(*pipeline=None*, *endpoint='api'*)

Bases: `object`

A class for defining and registering datasets on the cellhub api.

When initialising an instance of the class, the pipeline name is passed e.g.:

```
x = cellhub.tasks.api.register("cell_qc")
```

Note: pipeline names are sanitised to replace spaces, underscores and hypens with periods.

define_dataset(*analysis_name=None*, *analysis_description=None*, *data_subset=None*, *data_id=None*, *data_format=None*, *file_set=None*)

Define the dataset.

The "data_subset", "data_id" and "data_format" parameters are optional.

The file_set is a dictionary that contains the files to be registered:

```
{ "name": { "path": "path/to/file",
            "format": "file-format",
            "description": "free-text" }
```

the top level "name" keys are arbitrary and not exposed in the API

e.g. for cell ranger output the file_set dictionary looks like this:

```
{"barcodes": {"path":"path/to/barcodes.tsv",
              "format": "tsv",
              "description": "cell barcode file"},
{"features": {"path":"path/to/features.tsv",
              "format": "tsv",
              "description": "features file"},
```

(continues on next page)

(continued from previous page)

```
{"matrix": {"path":"path/to/matrix.mtx",
              "format": "market-matrix",
              "description": "Market matrix file"}
}
```

register_dataset()

Register the dataset on the service endpoint. The method:

1. creates the appropriate folders in the "api" endpoint folder
2. symlinks the source files to the target location
3. constructs and deposits the manifest.yml file

The location at which datasets will be registered is defined as:

```
api/pipeline.name/analysis_name/[data_subset/][data_id/][data_format/]
```

(data_subset, data_id and data_format are [optional])

show()

Print the api object for debugging.

reset_endpoint()

Clean the dataset endpoint

CHAPTER

SIX

PIPELINE DOCUMENTATION

6.1 Pipeline ambient rna

6.1.1 Overview

This pipeline performs the following steps: * Analyse the ambient RNA profile in each input (eg. channel's or library's raw cellrange matrices) * Compare ambient RNA profiles across inputs

Configuration

The pipeline requires a configured `pipeline.yml` file. Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_ambient_rna.py config
```

Input files

An tsv file called 'input_libraries.tsv' is required. This file must have column names as explained below. Must not include row names. Add as many rows as iput channels/libraries for analysis.

This file must have the following columns:

- library_id - name used throughout. This could be the channel_pool id eg. A1
- raw_path - path to the raw_matrix folder from cellranger count
- exp_batch - might or might not be useful. If not used, fill with "1"
- channel_id - might or might not be useful. If not used, fill with "1"
- seq_batch - might or might not be useful. If not used, fill with "1"
- (optional) excludelist - path to a file with cell_ids to excludelist

You can add any other columns as required, for example pool_id

Dependencies

This pipeline requires: * cgat-core: https://github.com/cgat-developers/cgat-core * R dependencies required in the r scripts

6.1.2 Pipeline output

The pipeline returns: * per-input html report and tables saved in a 'profile_per_input' folder * ambient rna comparison across inputs saved in a 'profile_compare' folder

6.1.3 Code

`cellhub.pipeline_ambient_rna.`**`ambient_rna_per_input`**`(`*infile*, *outfile*`)`
Explore count and gene expression profiles of ambient RNA droplets per input - The output is saved in profile_per_input.dir/<input_id> - The output consists on a html report and a ambient_genes.txt.gz file - See more details of the output in the ambient_rna_per_library.R

`cellhub.pipeline_ambient_rna.`**`ambient_rna_compare`**`(`*infiles*, *outfile*`)`
Compare the expression of top ambient RNA genes across inputs - The output is saved in profile_compare.dir - Output includes and html report and a ambient_rna_profile.tsv - See more details of the output in the ambient_rna_compare.R

`cellhub.pipeline_ambient_rna.`**`plot`**`(`*infile*, *outfile*`)`
Draw the pipeline flowchart

`cellhub.pipeline_ambient_rna.`**`full`**`()`
Run the full pipeline.

6.2 Pipeline Cellranger Multi

6.2.1 Overview

This pipeline performs the following functions:

- Alignment and quantitation (using cellranger count or cellranger multi)

6.2.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline_cellranger_multi.yml` file.

Default configuration files can be generated by executing:

python <srcdir>/pipeline_cellranger_multi.py config

Dependencies

This pipeline requires: * cgat-core: https://github.com/cgat-developers/cgat-core * cellranger: https://support.10xgenomics.com/single-cell-gene-expression/

6.2.3 Pipeline output

The pipeline returns: * the output of cellranger multi

6.2.4 Code

`cellhub.pipeline_cellranger_multi.`**`taskSummary`**`(`*infile*, *outfile*`)`
    Make a summary of optional tasks that will be run

`cellhub.pipeline_cellranger_multi.`**`makeConfig`**`(`*outfile*`)`
    Read parameters from yml file for the whole experiment and save config files as csv.

`cellhub.pipeline_cellranger_multi.`**`cellrangerMulti`**`(`*infile*, *outfile*`)`
    Execute the cellranger multi pipleline for first sample.

`cellhub.pipeline_cellranger_multi.`**`postProcessMtx`**`(`*infile*, *outfile*`)`
    Post-process the cellranger multi matrices to split the counts for the GEX, ADT and HTO modalities into seperate market matrices.

The cellbarcode are reformatted to the "UMI-LIBRARY_ID" synta.

The input cellranger.multi.dir folder layout is:

unfiltered "outs":

```
library_id/outs/multi/count/raw_feature_bc_matrix/
```

filtered "outs":

```
library_id/outs/per_sample_outs/sample|library_id/count/sample_feature_bc_matrix
```

This task produces

unfiltered: :: out.dir/library_id/unfiltered/mtx/[GEX|ADT|HTO]/

filtered: :: out.dir/library_id/filtered/sample_id/mtx/[GEX|ADT|HTO]/

`cellhub.pipeline_cellranger_multi.`**`mtxAPI`**`(`*infile*, *outfile*`)`
    Register the post-processed mtx files on the API endpoint

`cellhub.pipeline_cellranger_multi.`**`h5API`**`(`*infile*, *outfile*`)`
    Put the h5 files on the API

The input cellranger.multi.dir folder layout is:

unfiltered "outs":

```
library_id/outs/multi/count/raw_feature_bc_matrix/
```

filtered "outs":

```
library_id/outs/per_sample_outs/sample|library_id/count/sample_feature_bc_matrix
```

`cellhub.pipeline_cellranger_multi.`**`postProcessVDJ`**`(`*infile*, *outfile*`)`

Post-process the cellranger contig annotations.

The cellbarcodes are reformatted to the "UMI-LIBRARY_ID" syntax.

The input cellranger.multi.dir folder layout is:

unfiltered "outs":

```
library_id/outs/multi/vdj_[b|t]/
```

filtered "outs":

```
library_id/outs/per_sample_outs/sample|library_id/vdj_[b|t]/
```

This task produces

unfiltered: :: out.dir/library_id/unfiltered/vdj_[t|b]/

filtered: :: out.dir/library_id/filtered/sample_id/vdj_[t|b]/

`cellhub.pipeline_cellranger_multi.`**`vdjAPI`**`(`*infile*, *outfile*`)`

Register the post-processed VDJ contigfiles on the API endpoint

`cellhub.pipeline_cellranger_multi.`**`full`**`()`

Run the full pipeline.

6.3 Pipeline Cell QC

6.3.1 Overview

This pipeline performs the following steps:

- Calculates per-cell QC metrics: ngenes, total_UMI, pct_mitochondrial, pct_ribosomal, pct_immunoglobin, pct_hemoglobin, and any specified geneset percentage
- Runs scrublet to calculate per-cell doublet score

Configuration

The pipeline requires a configured `pipeline.yml` file. Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_cell_qc.py config
```

Input files

A tsv file called 'libraries.tsv' is required. This file must have column names as explained below. Must not include row names. Add as many rows as input channels/librarys for analysis. This file must have the following columns: * library_id - name used throughout. This could be the channel_pool id eg. A1 * path - path to the filtered_matrix folder from cellranger count

Dependencies

This pipeline requires: * cgat-core: https://github.com/cgat-developers/cgat-core * R dependencies required in the r scripts

6.3.2 Pipeline output

The pipeline returns: * qcmetrics.dir folder with per-input qcmetrics.tsv.gz table * scrublet.dir folder with per-input scrublet.tsv.gz table

6.3.3 Code

`cellhub.pipeline_cell_qc.`**`qcmetrics`**`(`*infile*, *outfile*`)`
This task will run R/calculate_qc_metrics.R, It uses the input_libraries.tsv to read the path to the cellranger directory for each input Ouput: creates a cell.qc.dir folder and a library_qcmetrics.tsv.gz table per library/channel For additional input files check the calculate_qc_metrics pipeline.yml sections: - Calculate the percentage of UMIs for genesets provided - Label barcodes as True/False based on whether they are part or not of a set of lists of barcodes provided

`cellhub.pipeline_cell_qc.`**`qcmetricsAPI`**`(`*infiles*, *outfile*`)`
Add the QC metrics results to the API

`cellhub.pipeline_cell_qc.`**`scrublet`**`(`*infile*, *outfile*`)`
This task will run python/run_scrublet.py, It uses the input_libraries.tsv to read the path to the cellranger directory for each input Ouput: creates a scrublet.dir folder and a library_scrublet.tsv.gz table per library/channel It also creates a doublet score histogram and a double score umap for each library/channel Check the scrublet section in the pipeline.yml to specify other parameters

`cellhub.pipeline_cell_qc.`**`scrubletAPI`**`(`*infiles*, *outfile*`)`
Add the scrublet results to the API

`cellhub.pipeline_cell_qc.`**`plot`**`(`*infile*, *outfile*`)`
Draw the pipeline flowchart

`cellhub.pipeline_cell_qc.`**`full`**`()`
Run the full pipeline.

6.4 Pipeline Emptydrops

6.4.1 Overview

This pipeline performs the following task:

- run emptydrops on the raw output of cellranger

6.4.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline.yml` file.

Default configuration files can be generated by executing:

> python <srcdir>/pipeline_emptydrops.py config

Input files

The pipeline is run from the cellranger count output (raw_feature_bc_matrix folder).

The pipeline expects a tsv file containing a column named path and a column named sample_id.

'raw path' should contain the path to each cellranger path to raw_feature_bc_matrix. 'sample_id' is the desired name for each sample (output folder will be named like this).

Dependencies

This pipeline requires: * cgat-core: https://github.com/cgat-developers/cgat-core * R + packages

6.4.3 Pipeline output

The pipeline returns: A list of barcodes passing emptydrops cell identification and a table with barcode ranks including all barcodes (this can be used for knee plots).

6.4.4 Code

`cellhub.pipeline_emptydrops.`**`emptyDrops`**`(`*infile*, *outfile*`)`
: Run Rscript to run EmptyDrops on each library

`cellhub.pipeline_emptydrops.`**`meanReads`**`(`*infile*, *outfile*`)`
: Calculate the mean reads per cell

`cellhub.pipeline_emptydrops.`**`full`**`()`
: Run the full pipeline.

6.5 Pipeline Velocity

6.5.1 Overview

This pipeline performs the following steps:

- sort bam file by cell barcode
- estimate intronic and exonic reads using velocyto (on selected barcodes)

6.5.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline_velocity.yml` file.

Default configuration files can be generated by executing:

> python <srcdir>/pipeline_velocity.py config

Input files

The pipeline is run from bam files generated by cellranger count.

The pipeline expects a tsv file containing the path to each cellranger bam file (path) and the respective sample_id for each sample. In addition a list of barcodes is required, this could be the filtered barcodes from cellranger or a custom input (can be gzipped file). Any further metadata can be added to the file. The required columns are sample_id, barcodes and path.

Dependencies

This pipeline requires: * cgat-core: https://github.com/cgat-developers/cgat-core * samtools * veloctyo

6.5.3 Pipeline output

The pipeline returns: * a loom file with intronic and exonic reads for use in scvelo analysis

6.5.4 Code

`cellhub.pipeline_velocity.`**`checkInputs`**`(`*outfile*`)`
: Check that input_samples.tsv exists and the path given in the file is a valid directorys.

`cellhub.pipeline_velocity.`**`genClusterJobs`**`()`
: Generate cluster jobs for each sample

`cellhub.pipeline_velocity.`**`sortBam`**`(`*infile*, *outfile*`)`
: Sort bam file by cell barcodes

`cellhub.pipeline_velocity.`**`runVelocyto`**`(`*infile*, *outfile*`)`
: Run velocyto on barcode-sorted bam file. This task writes a loom file into the pipeline-run directory for each sample.

`cellhub.pipeline_velocity.`**`full`**`()`
: Run the full pipeline.

6.6 Pipeline fetch cells

6.6.1 Overview

This pipeline fetches a given set of cells from market matrices or loom files into a single market matrix file.

6.6.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

It is recommended to fetch the cells into a new directory. Fetching of multiple datasets per-directory is (deliberately) not supported.

The pipeline requires a configured `pipeline_fetch_cells.yml` file.

Default configuration files can be generated by executing:

> python <srcdir>/pipeline_fetch_cells.py config

Inputs

The pipeline will fetch cells from a cellhub instance according to the parameters specified in the local pipeline_fetch_cell.yml file.

The location of the cellhub instances must be specificed in the yml:

```
cellhub:
    location: /path/to/cellhub/instance
```

The specifications of the cells to retrieve must be provided as an SQL statement (query) that will be executed against the "final" table of the cellhub database:

```
cellhub:
    sql_query: >-
        select * from final
        where pct_mitochondrial < 10
        and ngenes > 200;
```

The cells will then be automatically retrieved from the API.

Cell barcodes are set according to the "barcode_id" column which is set by pipeline_cellranger_multi.py and have the format "UMI-1-LIBRARY_ID"

Dependencies

This pipeline requires:

6.6.3 Pipeline output

The pipeline outputs a folder containing a single market matrix that contains the requested cells.

`cellhub.pipeline_fetch_cells.`**`fetchCells`**(*infile*, *outfile*)
: Fetch the table of the user's desired cells from the database effectively, cell-metadata tsv table.

`cellhub.pipeline_fetch_cells.`**`barcodeSubsets`**(*infile*, *output_files*, *sentinel*)
: Generate the sets of barcodes to retrieve from each of the source matrices. (These will be used for extraction of data from all of the specified modalities).

`cellhub.pipeline_fetch_cells.`**`GEXSubsets`**(*infile*, *outfile*)
: Extract the GEX cell subsets from the parent mtx.

`cellhub.pipeline_fetch_cells.`**`mergeGEXSubsets`**(*infiles*, *outfile*)
: Merge the GEX cell subsets into a single mtx.

`cellhub.pipeline_fetch_cells.`**`downsampleGEX`**(*infiles*, *outfile*)
: Down-sample transcripts given a cell-metadata variable

  TODO: incorporate down-sampling into the fetching of the cell subsets.

`cellhub.pipeline_fetch_cells.`**`exportAnnData`**(*infiles*, *outfile*)
: Export h5ad anndata matrices for downstream analysis with scanpy

`cellhub.pipeline_fetch_cells.`**`ADTSubsets`**(*infile*, *outfile*)
: Extract the ADT cell subsets from the parent mtx.

`cellhub.pipeline_fetch_cells.`**`mergeADTSubsets`**(*infiles*, *outfile*)
: Merge the ADT cell subsets into a single mtx.

`cellhub.pipeline_fetch_cells.`**`exportAnnDataADT`**(*infiles*, *outfile*)
: Export h5ad anndata matrices for downstream analysis with scanpy

6.7 Pipeline ADT normalization

6.7.1 Overview

This pipeline implements three normalization methods:

- DSB (https://www.biorxiv.org/content/10.1101/2020.02.24.963603v3)
- Median-based (https://bioconductor.org/books/release/OSCA/integrating-with-protein-abundance.html)
- CLR (https://satijalab.org/seurat/archive/v3.0/multimodal_vignette.html)

Configuration

The pipeline requires a configured `pipeline_adt_norm.yml` file. Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_adt_norm.py config
```

Input files

This pipeline requires the unfiltered gene-expression and ADT count matrices and a list of high quality barcodes most likely representing single-cells.

This means that ideally this pipeline is run after high quality cells are selected via the pipeline_fetch_cells.py.

This pipeline will look for the unfiltered matrix in the api:

./api/cellranger.multi/ADT/unfiltered//*mtx*/.gz

./api/cellranger.multi/GEX/unfiltered//*mtx*/.gz

Dependencies

This pipeline requires: * cgat-core: https://github.com/cgat-developers/cgat-core * R dependencies required in the r scripts

6.7.2 Pipeline output

The pipeline returns a adt_norm.dir folder containing one folder per methodology adt_dsb.dir, adt_median.dir, and adt_clr.dir with per-sample folders conatining market matrices [features, qc-barcodes] with the normalized values.

6.7.3 Code

`cellhub.pipeline_adt_norm.`**`gexdepth`**(*infile*, *outfile*)
This task will run R/calculate_depth_dist.R, It will describe the GEX UMI distribution of the background and cell-containing barcodes. This will help to assess the quality of the ADT data and will inform about the definition of the background barcodes.

`cellhub.pipeline_adt_norm.`**`gexdepthAPI`**(*infiles*, *outfile*)
Add the umi depth metrics results to the API

`cellhub.pipeline_adt_norm.`**`adtdepth`**(*infile*, *outfile*)
This task will run R/calculate_depth_dist.R, It will describe the ADT UMI distribution of the background and cell-containing barcodes. This will help to assess the quality of the ADT data and will inform the definition of the background barcodes.

`cellhub.pipeline_adt_norm.`**`adtdepthAPI`**(*infiles*, *outfile*)
Add the umi depth metrics results to the API

`cellhub.pipeline_adt_norm.`**`plot_norm_adt`**(*infile*, *outfile*)
This task will run R/plot_norm_adt.R, It will create a visual report on the cell vs background dataset split and, if the user provided GEX and ADT UMI thresholds, those will be included.

`cellhub.pipeline_adt_norm.`**`dsb_norm`**(*infile*, *outfile*)
This task runs R/normalize_adt.R. It reads the unfiltered ADT count matrix and calculates DSB normalized ADT expression matrix which is then saved like market matrices per sample.

`cellhub.pipeline_adt_norm.`**`dsbAPI`**`(`*infile*, *outfile*`)`
Register the ADT normalized mtx files on the API endpoint

`cellhub.pipeline_adt_norm.`**`median_norm`**`(`*infile*, *outfile*`)`
This task runs R/get_median_adt_normalization.R, It reads the filtered ADT count matrix and performed median-based normalization. Calculates median-based normalized ADT expression matrix and writes market matrices per sample.

`cellhub.pipeline_adt_norm.`**`medianAPI`**`(`*infile*, *outfile*`)`
Register the ADT normalized mtx files on the API endpoint

`cellhub.pipeline_adt_norm.`**`clr_norm`**`(`*infile*, *outfile*`)`
This task runs R/get_median_clr_normalization.R, It reads the filtered ADT count matrix and performes CLR normalization. Writes market matrices per sample.

`cellhub.pipeline_adt_norm.`**`clrAPI`**`(`*infile*, *outfile*`)`
Register the CLR-normalized ADT mtx files on the API endpoint

`cellhub.pipeline_adt_norm.`**`plot`**`(`*infile*, *outfile*`)`
Draw the pipeline flowchart

`cellhub.pipeline_adt_norm.`**`full`**`()`
Run the full pipeline.

6.8 Pipeline Integration

6.8.1 Overview

This pipeline combines different integration methods, mainly using Satija lab's Seurat package (http://satijalab.org/seurat/).

For key parameters a range of choices can be specified. The pipeline will generate one report for each sample or experiment. All integration tasks and parameter combinations will be run in parallel on an HPC cluster.

The pipeline also assesses the integration by using metrics from the Seurat paper, the iLISI (harmony), the k-bet package and (optional) a similar entropy metric to what is suggested in the conos alignment method.

6.8.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline.yml` file.

Default configuration files can be generated by executing:

python <dropdir>/pipeline_integration.py config

Input files

The pipeline takes as input a file named input_samples.tsv. This file has to contain a column named sample_id representing the name for a sample or aggregated experiment. The other required column is 'path'. This has to be the path to the directory containing the aggregated matrix, metadata, barcode and features file for the respective sample (e.g. dropflow qc output)

sample_id path pbmc /path/to/matrix/

$ ls /path/to/matrix/ barcodes.tsv.gz features.tsv.gz matrix.mtx.gz metadata.tsv.gz

Dependencies

This pipeline requires:

- cgat-core: https://github.com/cgat-developers/cgat-core
- R & various packages.

6.8.3 Pipeline output

For each sample and each combination of parameters the following is generated within a folder for each tool+parameter:

- UMAP colored by the metadata column used for integration
- output of kbet, Seurats integration methods and harmony's iLISI in folder assess.integration.dir
- folder summary.plots.dir with summaries of each metric above
- folder summary.plots.dir also contains a csv file with the metrics
- (optional) within each folder with different cluster resolutions, a folder with entropy plots and UMAP with cluster assignments

`cellhub.pipeline_integration.`**`checkInputs`**`(`*outfile*`)`
: Check that input_samples.tsv exists and the path given in the file exists. Then make one folder for each experiments called *.exp.dir

`cellhub.pipeline_integration.`**`genClusterJobs`**`()`
: Generate cluster jobs with all paramter combinations.

`cellhub.pipeline_integration.`**`prepFolders`**`(`*infile*, *outfile*`)`
: Task to prepare folders for integration

`cellhub.pipeline_integration.`**`runScanpyIntegration`**`(`*infile*, *outfile*`)`
: Run scanpy normalization, hv genes and harmonypy on the data

`cellhub.pipeline_integration.`**`runScanpyUMAP`**`(`*infile*, *outfile*`)`
: Run scanpy UMAP and make plots

`cellhub.pipeline_integration.`**`plotUMAP`**`(`*infile*, *outfile*`)`
: Plot UMAP with different variables

`cellhub.pipeline_integration.`**`genJobsSummary`**`()`
: Job generator for summary jobs

`cellhub.pipeline_integration.`**`summariseUMAP`**`(`*infile*, *outfile*`)`
: Summarise UMAP from different methods

`cellhub.pipeline_integration.`**`runLISIpy`**`(`*infile*, *outfile*`)`
Assess the integration using iLISI (lisi on batch/dataset). Use the python implementation as part of the harmonypy package

`cellhub.pipeline_integration.`**`genJobsLISI`**`()`
Job generator for LISI summary jobs

`cellhub.pipeline_integration.`**`summariseLISI`**`(`*infile*, *outfile*`)`
summarise the results from iLISI analysis

`cellhub.pipeline_integration.`**`plot`**`(`*infile*, *outfile*`)`
Draw the pipeline flowchart

6.9 Pipeline Export

6.9.1 Overview

This pipeline performs the following task:

- convert anndata to seurat object

6.9.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline.yml` file.

Default configuration files can be generated by executing:

python <srcdir>/pipeline_emptydrops.py config

Input files

The pipeline is run from the anndata object created by pipeline_integration.py. A sample name, the path to the anndata object (path) and (optional) path to the folder with the original matrix data (matrixdir) are specified in input_samples.tsv. If matrixdir is specified then market matrix format input for the full dataset are expected. The alternative option is to supply the path to a anndata object instead of the matrixdir. The option use_full_anndata in pipeline.yml needs to be set if anndata should be used.

The column names in input_samples.tsv need to be sample_id, path and either matrixdir or anndata. The dim reduction to include in the output seurat object is specified in the pipeline.yml.

Dependencies

This pipeline requires: * cgat-core: https://github.com/cgat-developers/cgat-core * python - scanpy, anndata * R - Seurat

6.9.3 Pipeline output

The pipeline returns: A seurat object.

6.9.4 Code

`cellhub.pipeline_export.`**`checkInputs`**`(`*outfile*`)`
  Check that input_samples.tsv exists and the path given in the file is a valid directorys.

`cellhub.pipeline_export.`**`genClusterJobs`**`()`
  Generate cluster jobs for each sample

`cellhub.pipeline_export.`**`exportFromAnndata`**`(`*infile*, *outfile*`)`
  Run python script to extract data from anndata object

`cellhub.pipeline_export.`**`createSeuratObject`**`(`*infile*, *outfile*`)`
  Run R script to create seurat object

`cellhub.pipeline_export.`**`full`**`()`
  Run the full pipeline.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX