cellhub

Release 0.1

Sansom lab

Sep 01, 2023

CONTENTS

Cellhub provides an end-to-end scaleable workflow for the pre-processing, warehousing and analysis of data from millions of single-cells. It aims to bring together best practice solutions, including for read alignment (Cellranger), ambient RNA correction (CellBender), de-multiplexing and de-hashing (GMMDemux), cell type prediction (singleR) and cluster analysis (Scanpy) into a cohesive set of easy to use analysis pipelines. It relies on the cgat-core workflow management system to leverage the power of high-performance compute clusters for high-throughput parallel processing. Pipelines cross-talk via a defined API allowing for easy extension or modification of the workflow. Cells and their associated qc-statistics and metadata are indexed in a central SQLite database from which arbitrary subsets can be fetched for downstream analysis in anndata format. The clustering pipeline allows rapid evaluation of different pre-processing and integration strategies at different clustering resolutions, through generation of pdf reports and cellxgene objects.

Cellhub was originally developed to support the single-cell component of the University of Oxford's COMBAT COVID-19 project.

Cellhub is currently alpha software. More detailed examples and tutorials will follow soon.

CHAPTER

ONE

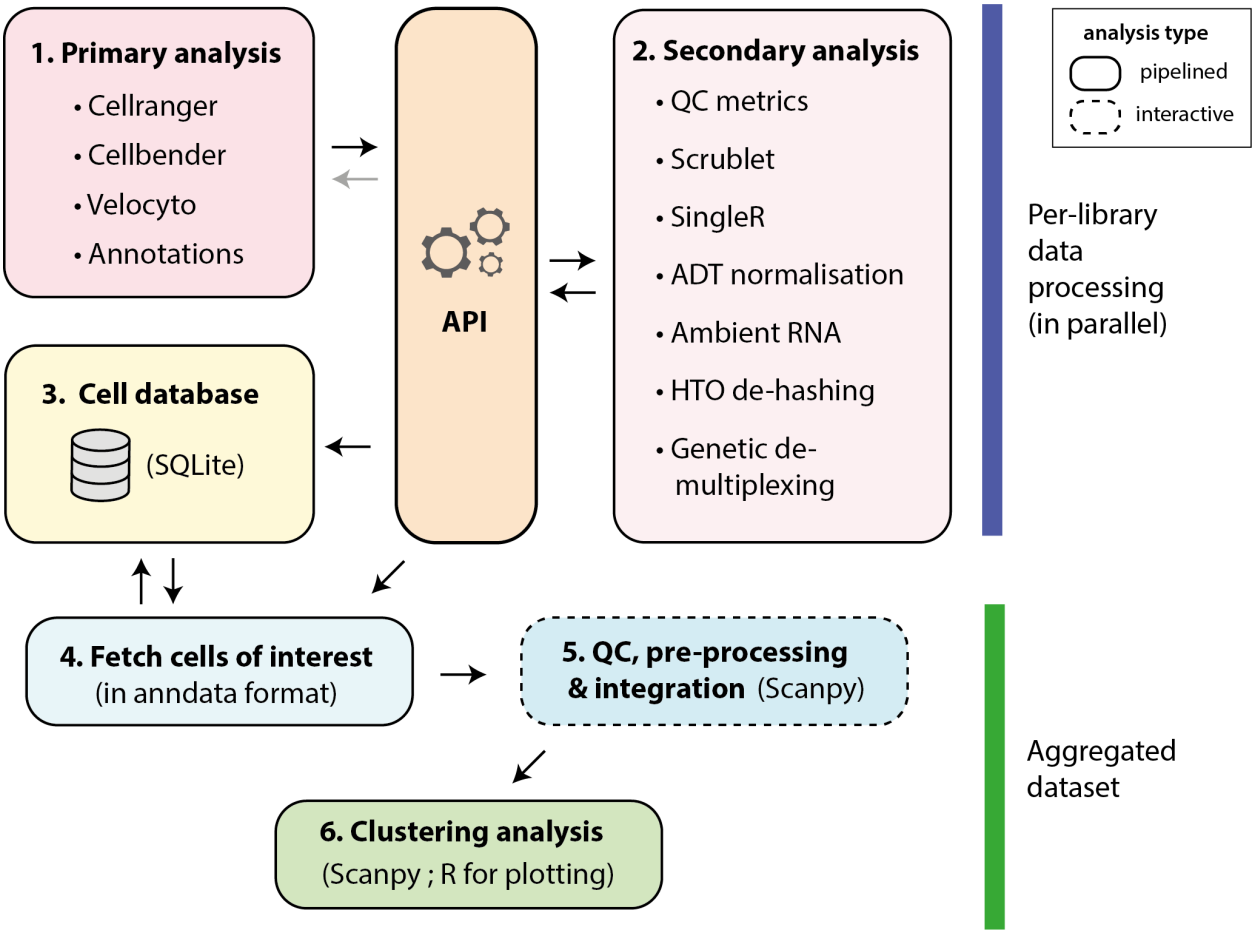
WORKFLOW OVERVIEW

1.1 Introduction

Cellhub is designed to efficiently parallelise the processing of large 10x datasets on compute clusters. Data for each of the sequencing libraries in the experiment is quantitated and quality controlled in parallel. Sample metadata, qc statistics and other per-cell information are stored in an SQL database. This information can be easily retrieved from a virtual table in the database for QC analysis. Data for arbitrary sets of cells of interest (specified using an SQL query) can be automatically and rapidly extracted in anndata format for downstream analysis.

By design, in the cellhub workflow, cell QC assessment, pre-processing and integration are performed interactively by the user. Often, different approaches to these steps are appropriate for different datasets and the user is directed to e.g. Luecken et al. 'Current best practices in single-cell RNA-seq analysis: a tutorial' and Lueken et al. 'Benchmarking atlas-level data integration in single-cell genomics' to get a feel for the issues.

Once the data has been integrated, the cellhub clustering pipeline can be used to execute detailed cell clustering analysis for a selection of resolutions. This pipeline will identify marker genes, perform pathway analysis and export pdf reports and a cellxgene object for inspection.



As shown in the image above, the workflow can be divided into six parts. Interoperability between the pipelines is enabled by a structured API, which facilities processing of complex experiment designs (including genetic/hash tag demultiplexing), correction of ambient RNA and easy addition of new workflows.

1.2 1. Primary analysis

These pipelines process the raw data to generate count matrices or fetch annotations from externals sources. They register their outputs on the API.

- The workflow typically begins with *pipeline_cellranger.py*. The pipeline maps reads from the different libraries in parallel using "cellranger count" and "cellranger vdj" commands.
- CellBender can be run to correct ambient RNA with *pipeline_cellbender.py*. Typically, this pipeline will be run in a separate directory so that the downstream results can be compared with those from the uncorrected cellranger run.
- Velocyto can be run to compute spliced and unspliced counts for RNA-velocity analysis with *pipeline_velocyto.py*.
- Cell identification can be performed with *pipeline_emptydrops.py* for comparison with the Cellranger calls. Currently results are not available for use downstream.

1.3 2. Secondary analysis

These pipelines start from the outputs of the primary analysis pipelines on the API.

- Per-cell QC 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. The pipeline also runs the Scrublet algorithm for doublet prediction.
- Per-cell celltype predictions are computed in parallel for each channel library using *pipeline_singleR.py*
- The per-library patterns of ambient RNA can be inspected using *pipeline_ambient_rna.py*.
- Cells multiplexed with hash-tags can be de-hashed using *pipeline_dehash.py*
- 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.

1.4 3. Loading results into the cell database

The library and sample metadata, per cell statistics (and demultiplex assignments) etc, 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.

1.5 4. 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 an anndata object for downstream analysis.

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 Velocyto results for RNA-velocity analysis.

Note: The retrieved metadata will include a "sample_id" column. From this point onwards it may be 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.

1.6 5. Assessment of cell quality, pre-processing and integration

These steps are performed manually.

- Per cell QC statistics and singleR scores can be easily retrieved from the celldb or anndata object for inspection with R or python.
- It is recommended to perform pre-processing using Scanpy. Strategies for HVG selection and modelling of covariates should be considered by the data analyst on a case by case basis.
- Integration is normally performed in python with e.g. scVI, harmonpy or BBKNN. Different integration algorithms are needed for different contexts.

1.7 6. Clustering analysis

Clustering analysis is performed with pipeline_cluster.py. The pipeline starts from an anndata provided by the user in the format described in the pipeline documentation: *pipeline_cluster*.

CHAPTER

TWO

INSTALLATION

2.1 Dependencies

Core dependencies include:

- Cellranger (from 10X Genomics) >= 6.0
- Python3
- The cgat-core pipeline framework
- Python packages as per python/requirements.txt
- R >= 4.0
- Various R libraries (see R/install.packages.R)
- Latex
- The provided cellhub R library

Optional dependencies include:

- CellBender
- GMM-Demux
- pegasuspy (for demuxEM)

2.2 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/sansomlab/cellhub.git
cd cellhub
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/python/requirements.txt
```

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

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

5. Install the cellhub R library:

```
R CMD INSTALL R/cellhub
```

Note: On some systems, automatic detection of the HDF5 library by the R package "hdf5r" (a dependency of loomR) is problematic. This can be worked around by explicitely passing the path to your HDF5 library h5cc or h5pcc binary, e.g.

```
install.packages("hdf5r", configure.args="--with-hdf5=/apps/eb/dev/skylake/software/HDF5/
↪1.10.6-gompi-2020a/bin/h5pcc")
```

CHAPTER

THREE

USAGE

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 pipeline as an example.

Following installation, to find the available pipelines run:

```
cellhub -h
```

Next generate a configuration yml file:

```
cellhub cellranger config -v5
```

To fully run the example cellhub pipeline run:

```
cellhub cellranger make full -v5
```

However, it may be best to begin by running 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 show
```

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

```
cellhub cellranger make cellrangerMulti -v5
```

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

3.2 Getting Started

To get started please see the *IFNb example*.

For an example configuration files for a multimodal immune profiling experiment please see the *PBMC 10k example*.

CHAPTER

FOUR

EXAMPLES

4.1 IFNb PBMC example

4.1.1 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 integration is to be performed.
- "cluster" where we can perform downstream analysis with "cellhub cluster".

The folders contain the necessary configuration files: please edit the cellhub/libraries.tsv file to point to the location of the FASTQ files on your system.

4.1.2 Running the pre-processing pipelines and creating the database

2. Running Cellranger

The first step is to configure and run pipeline_cellranger. This pipeline takes three inputs (i) Information about the biological samples, number of cells expected and the 10x chemistry version are specified in a "samples.tsv" file. (ii) Input 10X channel library identifiers "library_id", sample prefixes, library types and FASTQ paths are specified via a tab delimited "libraries.tsv" file. (iii) A pipeline_cellranger.yml file is used to configure general options such as computional resource specification and the location of the genomic references. For more details please see: *pipeline_cellranger.py*.

For this example, preconfigured "samples.tsv", "libraries.tsv" and "pipeline_cellranger.yml" files are provided in the cellhub directory.

Edit the "libraries.tsv" file "fastq_path" column 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.

The pipeline is run as follows:

```
cellhub cellranger make full -v5 -p20
```

Finally, the count matrices must be manually registered on the API for downstream analysis:

```
cellhub cellranger_multi make useCounts
```

Note: When processing other datasets the "samples.tsv" and "libraries.tsv" files must be created by the user. For more details on constructing these files please see *pipeline_cellranger.py*. A template 'pipeline_cellranger.yml' file can be obtained using the "config" command which is common to all cellhub piplines.:

cellhub cellranger config

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. Run pipeline_singleR

Single R is run on all the cells so that the results are avaliable to help with QC as well as downstream analysis:

```
# cellhub singleR config

cellhub singleR make full -v5 -p20
```

As noted: in the pipeline_singleR inputs section the celldex references need to be stashed before the pipeline is run.

6. 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
```

7. Run pipeline_annotation

This pipeline retrieves Ensembl and KEGG annotations needed for downstream analysis.:

```
# cellhub annotation config

cellhub annotation make full -v5 -p10
```

Please note that the specified Ensembl version should match that used for the cellranger reference trancriptome.

4.1.3 Performing cell QC

8. Assessment of cell quality

This step is left to the reader to perform manually because it needs to be carefully tailored to individual datasets.

4.1.4 Performing downstream analysis

9. Fetch cells for integration

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:

```
# It is recommended to fetch the cells in to a seperate directory for integration.
cd ../integration

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

10. Integration

Run the provided jupyter notebook to perform a basic Harmony integration of the data and to save it in the appropriate anndata format (see in the pipeline_cluster inputs section) is provided.

11. Clustering analysis

Cluster analysis is performed with pipeline cluster (a seperate directory is recommended for this so that multiple clustering runs can be performed as required).:

```
# change into the clustering directory
cd ../cluster.dir

# checkout the yml file
cellhub cluster config

# a suitable yml file has been provided so we can now launch the pipeline
cellhub cluster make full -v5 -p200
```

The pdf reports and excel files generated by the pipeline can be found in the "reports.dir" subfolder.

For interactive visulation, the results are provided in cellxgene format. To view the cellxgene.h5ad files, you will first need toinstall cellxgene with "pip install cellxgene". The cellxgene viewer can then be launched with:

```
# substitute "{x}" with the number integrated components used for the clustering run.
cellxgene --no-upgrade-check launch out.{x}.comps.dir/cellxgene.h5ad
```

4.2 10k Human PBMCs 5' v2.0 example

The 10x 10k Human PBMCs, 5' v2.0 dataset contains multimodal GEX, TCR and BCR data from a single human subject.

Example configuration files ("samples.tsv", "libraries.tsv" and "pipeline_cellranger.yml") for processing this dataset are provided in the "examples/10k_pbmc" folder.

Note: Due to a quirk with "cellranger vdj" fastq data for vdj libraries from different flow cells (for the same sample) needs to be presented in different folders. In other words, if fastqs for a vdj sample from different flow cells are presented in the same folder with different "sample" prefix the "cellranger vdj" command fails.

For downstream analysis please see the *IFNb example*.

CHAPTER

FIVE

PIPELINES

5.1 pipeline_adt_norm.py

5.1.1 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

5.1.2 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.

5.1.3 Code

cellhub.pipeline_adt_norm.**gexdepth**(*infile*, *outfile*)

This task will run R/adt_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/adt_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.**adt_plot_norm**(*infile*, *outfile*)

This task will run R/adt_plot_norm.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/adt_normalize.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/adt_get_median_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.

5.2 pipeline_ambient_rna.py

5.2.1 Overview

This pipeline performs the following steps: * Analyse the ambient RNA profile in each input (eg. channel's or library's raw cellranger 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

5.2.2 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

5.2.3 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.

5.3 pipeline_annotation.py

5.3.1 Overview

This pipeline retrieves annotation from Ensembl

5.3.2 Usage

The annotation pipeline should be run in the cellhub directory.

Configuration

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

Default configuration files can be generated by executing:

python <srcdir>/pipeline_annotation.py config

The ensembl version specified in the yaml file should match that used to build the reference transcriptome for the mapping algorithm (e.g. Cellranger)

Inputs

This pipeline has no inputs.

Dependencies

This pipeline requires:

5.3.3 Pipeline output

The pipeline produces the following outputs:

1. api/annotation/ensembl/ensembl.to.entrez.tsv.gz

- A mapping of ensembl_id to gene_name and entrez_id. Used by gsfisher for pathway analysis.

2. api/annotation/ensembl/ensembl.gene_name.map.tsv.gz

- A unique mapping of ensembl_id -> gene_name. Missing gene names are replaced with ensembl_ids. The gene names have been made unique.

3. api/annotation/kegg/kegg_pathways.rds

- Kegg pathways in rds format for gsfisher.

`cellhub.pipeline_annotation.`**`fetchEnsembl`**(*infile*, *outfile*)

Fetch the ensembl annotations from BioMart. This task requires internet access.

`cellhub.pipeline_annotation.`**`ensemblAPI`**(*infile*, *outfile*)

Add the Ensembl gene annotation results to the cellhub API.

`cellhub.pipeline_annotation.`**`fetchKegg`**(*infile*, *outfile*)

Fetch the Kegg pathway annotations. This task requires internet access.

`cellhub.pipeline_annotation.`**`keggAPI`**(*infile*, *outfile*)

Add the kegg pathways to the cellhub API

5.4 pipeline_cellbender.py

5.4.1 Overview

This pipeline uses CellBender to remove ambient UMI counts.

Configuration

This pipeline is normally run in a seperate e.g. "cellhub_cellbender" directory so that the downstream results can be compared with those from cellranger.

The pipeline requires a configured `pipeline_cellbender.yml` file. A default version can be obtained by executing:

```
cellhub cellbender config
```

Input

The location of the cellhub folder containing the cellranger results that will be used as the input for CellBender is be specified in the "pipeline_cellbender.yml" configuration file. Typically the user will have two parallel "cellhub" instances, e.g.:

1. "cellhub" <- containing a first cellhub run based on the Cellranger counts (counts registered with "cellhub cellranger make useCounts").
2. "cellhub_cellbender" <- containing a second cellhub run using CellBender to correct the Cellranger counts from the first run (counts registered with "cellhub cellbender make useCounts").

Running the pipeline

It is recommended to run the cellbender task on a gpu queue.

On the University of Oxford's BMRC cluster, this can be achieved with e.g.

```
cellhub cellbender make cellbender -v5 -p 200 --cluster-queue=short.qg --cluster-options
↪"-l gpu=1,gputype=p100"
```

5.4.2 Pipeline output

The pipeline registers cleaned CellBender h5 files on the local cellhub API. Currently this format is not fully compatible with the 10x h5 format. To work around this a custom loader is used, see the *cellhub.tasks.cellbender module documentation* for more details.

5.4.3 Code

`cellhub.pipeline_cellbender.`**`cellbender`**(*infile*, *outfile*)

This task will run the CellBender command. Please visit cellbender.readthedocs.io for further details.

`cellhub.pipeline_cellbender.`**`h5API`**(*infile*, *outfile*)

Put the h5 files on the API

Inputs:

The input cellbender.dir folder layout is:

unfiltered "outs":

```
library_id/cellbender.h5
```

filtered "outs":

```
library_id/cellbender_filtered.h5
```

`cellhub.pipeline_cellbender.`**`mtx`**(*infile*, *outfile*)

Convert cellbender h5 to mtx format

`cellhub.pipeline_cellbender.`**`mtxAPI`**(*infile*, *outfile*)

Put the mtx files on the API

Inputs:

The input cellbender.dir folder layout is:

unfiltered "outs":

```
library_id/cellbender.h5
```

filtered "outs":

```
library_id/cellbender_filtered.h5
```

`cellhub.pipeline_cellbender.full()`

Run the full pipeline.

`cellhub.pipeline_cellbender.useCounts(`*infile*, *outfile*`)`

Set the cellbender counts as the source for downstream analysis. This task is not run by default.

5.5 pipeline_celldb.py

5.5.1 Overview

This pipeline uploads the outputs from the preprocessing pipelines into a SQLite database. Cell identifiers are joined with sample metadata, qc statistics and other per-cell information in a virtual table called "final" in the database.

Configuration

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

Default configuration files can be generated by executing:

cellhub celldb config

The user must edit the final_sql_query parameter in the configuration file to create the "final" database view appropriately.

Input files

1. A user-supplied 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. The pipeline requires the outputs of pipeline_cell_qc to be registered on the API
3. Optionally, the pipeline will load the results of pipeline_singleR and pipeline_dehash from the API if these tables are set to "active" in the configuration file.

Pipeline output

The pipeline returns an SQLite database that contains a "final" view which links cell identifiers with cell QC information, scrublet scores, user-provided metadata, cell type predictions (optional) and de-multiplexing assignments (optional). In the database cell identify is encoded by the "barcode" and "library_id" fields which are automatically indexed (as a multi-column index).

5.5.2 Code

`cellhub.pipeline_celldb.`**`connect`**`()`

Helper function to connect to DB

`cellhub.pipeline_celldb.`**`load_samples`**`(`*outfile*`)`

load the sample metadata table

`cellhub.pipeline_celldb.`**`load_gex_qcmetrics`**`(`*outfile*`)`

load the gex qcmetrics into the database

`cellhub.pipeline_celldb.`**`load_gex_scrublet`**`(`*outfile*`)`

Load the scrublet scores into database.

`cellhub.pipeline_celldb.`**`load_singleR`**`(`*outfile*`)`

Load the singleR predictions into the database.

`cellhub.pipeline_celldb.`**`load_gmm_demux`**`(`*outfile*`)`

Load the gmm demux dehashing calls into the database.

`cellhub.pipeline_celldb.`**`load_demuxEM`**`(`*outfile*`)`

load the demuxEM dehashing calls into the database

`cellhub.pipeline_celldb.`**`load_souporcell`**`(`*outfile*`)`

load the souporcell cluster result into the database

`cellhub.pipeline_celldb.`**`final`**`(`*outfile*`)`

Construct a "final" view on the database from which the cells can be selected and fetched by pipeline_fetch_cells.py

5.6 pipeline_cellranger.py

5.6.1 Overview

This pipeline performs the following functions:

- Alignment and quantitation of 10x GEX, CITE-seq and VDJ sequencing data.

5.6.2 Usage

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

Configuration

The pipeline requires a configured:file:*pipeline_cellranger.yml* file.

Default configuration files can be generated by executing:

    python <srcdir>/pipeline_cellranger.py config

Inputs

In addition to the "pipeline_cellranger.yml" file, the pipeline requires two inputs:

1. a "samples.tsv" file describing the samples
2. a "libraries.tsv" table containing the sample prefixes, feature type and fastq paths.

(i) samples.tsv

A table describing the samples and libraries to be analysed.

It must have the following columns:

- "sample_id" a unique identifier for the biological sample being analysed
- "library_id" is a unique identifier for the sequencing libraries generated from a single channel on a single 10x chip. Use the same "library ID" for Gene Expression, Antibody Capture, VDJ-T and VDJ-B libraries that are generated from the same channel.

Additional arbitrary columns describing the sample metadata should be included to aid the downstream analysis, for example

- "condition"
- "replicate"
- "timepoint"
- "genotype"
- "age"
- "sex"

For HTO hashing experiments include a column containing details of the hash tag, e.g.

- "hto_id"

(ii) libraries.tsv

A table that links individual sequencing libraries, library types and FASTQ file locations.

It must have the following columns:

- "library_id": Must match the library_ids provided in the "samples.tsv" file, for details see above.
- "feature_type": One of "Gene Expression", "Antibody Capture", "VDJ-T" or "VDJ-B". Case sensitive.
- "fastq_path": the location of the folder containing the fastq files
- "sample": this will be passed to the "–sample" parameter of the cellranger pipelines (see: https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/using/fastq-input). It is only used to match the relevant FASTQ files: it does not have to match the "sample_id" provided in the "samples.tsv" table, and is not used in downstream analysis.
- "chemistry": The 10x reaction chemistry, the options are:
  - 'auto' for autodetection,
  - 'threeprime' for Single Cell 3',
  - 'fiveprime' for Single Cell 5',
  - 'SC3Pv1',
  - 'SC3Pv2',
  - 'SC3Pv3',
  - 'SC5P-PE',
  - 'SC5P-R2' for Single Cell 5', paired-end/R2-only,
  - 'SC-FB' for Single Cell Antibody-only 3' v2 or 5'.
  - "expect_cells": An integer specifying the expected number of cells

It is recommended to include the following columns

- "chip": a unique ID for the 10x Chip
- "channel_id": an integer denoting the channel on the chip
- "date": the date the 10x run was performed

Note: Use of the cellranger "–lanes": parameter is not supported. This means that data from all the lanes present in the given location for the given "sample" prefix will be run. This applies for both Gene Expression and VDJ analysis. If you need to analyse data from a single lane, link the data from that lane into a different location.

Note: To combine sequencing data from different flow cells, add additional rows to the table. Rows with identical "library_id" and "feature_type" are automatically combined by the pipelines. If you are doing this for VDJ data, the data from the different flows cells must be in different folders as explained in the note below.

Note: For V(D)J analysis, if you need to combine FASTQ files that have a different "sample" prefix (i.e. from different flow cells) the FASTQ files with different "sample" prefixes must be presented in separate folders. This is because despite the docs indicating otherwise (https://support.10xgenomics.com/single-cell-vdj/software/pipelines/latest/using/vdj), "cellranger vdj" does not support this:

–sample prefix1,prefix2 –fastqs all_data/,all_data/

but it does support:

–sample prefix1,prefix2 –fastqs flow_cell_1/,flow_cell_2/.

Dependencies

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

Pipeline logic

The pipeline is designed to:

- map libraries in parallel to speed up analysis
- submit standalone cellranger jobs rather than to use the cellranger cluster mode which can cause problems on HPC clusters that are difficult to debug
- map ADT data with GEX data: so that the ADT analysis takes advantage of GEX cell calls
- map VDJ-T and VDJ-B libraries using the "cellranger vjd" command.

Note: 10x recommends use of "cellranger multi" for mapping libraries from samples with GEX and VDJ. This is so that barcodes present in the VDJ results but not the GEX cell calls can be removed from the VDJ results. Here for simplicity and to maximise parallelisation we use "cellranger vdj": it is trivial to remove VDJ barcodes without a GEX overlap downstream.

Pipeline output

The pipeline returns: * the output of cellranger

5.6.3 Code

`cellhub.pipeline_cellranger.`**`count`**(*infile*, *outfile*)

Execute the cellranger count pipeline

`cellhub.pipeline_cellranger.`**`mtxAPI`**(*infile*, *outfile*)

Register the count market matrix (mtx) files on the API endpoint

Inputs:

The input cellranger count folder layout is:

unfiltered "outs": ::
library_id/outs/raw_feature_bc_matrix [mtx] library_id/outs/raw_feature_bc_matrix.h5

filtered "outs": ::
library_id/outs/filtered_feature_bc_matrix library_id/outs/filtered_feature_bc_matrix.h5

`cellhub.pipeline_cellranger.`**`h5API`**(*infile*, *outfile*)

Put the h5 files on the API

`cellhub.pipeline_cellranger.`**`tcr`**(*infile*, *outfile*)

Execute the cellranger vdj pipeline for the TCR libraries

`cellhub.pipeline_cellranger.`**`registerTCR`**(*infile*, *outfile*)

Register the TCR contigfiles on the API endpoint

`cellhub.pipeline_cellranger.`**`mergeTCR`**(*infiles*, *outfile*)

Merge the TCR contig annotations

cellhub.pipeline_cellranger.**registerMergedTCR**(*infile*, *outfile*)

Register the merged TCR contigfiles on the API endpoint

cellhub.pipeline_cellranger.**bcr**(*infile*, *outfile*)

Execute the cellranger vdj pipeline for the BCR libraries

cellhub.pipeline_cellranger.**registerBCR**(*infile*, *outfile*)

Register the individual BCR contigfiles on the API endpoint

cellhub.pipeline_cellranger.**mergeBCR**(*infiles*, *outfile*)

Merge the BCR contigfiles

cellhub.pipeline_cellranger.**registerMergedBCR**(*infile*, *outfile*)

Register the merged VDJ-B contigfiles on the API endpoint

cellhub.pipeline_cellranger.**full**()

Run the full pipeline.

cellhub.pipeline_cellranger.**useCounts**(*infile*, *outfile*)

Set the cellranger counts as the source for downstream analysis. This task is not run by default.

5.7 pipeline_cell_qc.py

5.7.1 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

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

5.7.2 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.

5.8 pipeline_cluster.py

5.8.1 Overview

This pipeline performs clustering of integrated single cell datasets. Starting from an anndata object with integrated coordinates (e.g. from Harmony or scVI) the pipeline:

- Computes the neighbour graph using the HNSW alogrithm
- Performs UMAP compuation and Leiden clustering (with ScanPy)
- Visualises QC statistics on the UMAPs and by cluster
- Visualises singleR results
- Finds cluster marker genes (using the ScanPy 'rank_genes_groups' function)

- Performs pathway analysis of the cluster phenotypes (with gsfisher)
- Prepares marker gene and summary reports
- Export an anndata for viewing with cellxgene

For plotting of data in R, the pipeline saves the input anndata in loom format and reads data in R with the loomR library.

5.8.2 Usage

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

Configuration

It is recommended to perform the clustering in a new directory.

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

Default configuration files can be generated by executing:

    python <srcdir>/pipeline_cluster.py config

Inputs

The pipeline starts from an anndata object with the following structure.

- anndata.var.index -> gene names - these will be displayed in the plots
- anndata.var.gene_ids -> ensembl gene ids, optional but required for pathway analysis
- anndata.X -> scaled data (a dense matrix)
- anndata.layers["counts"] -> raw counts (a sparse matrix)
- anndata.layers["log1p"] -> total count normalised, log1p transformed data (a sparse matrix)
- anndata.obs -> metadata (typically passed through from original cellhub object)
- anndata.obsm["X_rdim_name"] -> containing the integrated coordinates (where "rdim_name" matches the "runspec_rdim_name" parameter). TODO: rename this parameter.

It is strongly recommended to retain the information for all of the genes in all of the matrices (i.e. do not subset to HVGs!). This is important for marker gene discovery and pathway analysis.

5.8.3 Pipeline output

The pipeline produces the following outputs:

1. Summary report
   - Containing the overview UMAP plots, visualisation of QC and SingleR information
   - The result of the per-cluster pathway analysis
2. Marker report
   - Containing heatmaps, violin plots, expression dotplots, MA and volcano plots for each cluster.
3. xlsx spreadsheets for the marker gene and pathway results
4. Anndata objects ready to be viewed with cellxgene

5.8.4 Code

cellhub.pipeline_cluster.**taskSummary**(*infile*, *outfile*)

Make a summary of optional tasks that will be run

cellhub.pipeline_cluster.**preflight**(*infile*, *outfile*)

Preflight sanity checks.

cellhub.pipeline_cluster.**metadata**(*infile*, *outfile*)

Export the metadata (obs) from the source anndata for use in the plotting tasks

cellhub.pipeline_cluster.**loom**(*infile*, *outfile*)

Export the data to the loom file format. This is used as an exchange format for plotting in R.

cellhub.pipeline_cluster.**neighbourGraph**(*infile*, *outfile*)

Compute the neighbor graph. A miniminal anndata is saved for UMAP computation and clustering.

cellhub.pipeline_cluster.**scanpyCluster**(*infile*, *outfile*)

Discover clusters using ScanPy.

cellhub.pipeline_cluster.**cluster**(*infile*, *outfile*)

Post-process the clustering result.

cellhub.pipeline_cluster.**compareClusters**(*infile*, *outfile*)

Draw a dendrogram showing the relationship between the clusters.

cellhub.pipeline_cluster.**clustree**(*infile*, *outfile*)

Run clustree.

cellhub.pipeline_cluster.**paga**(*infile*, *outfile*)

Run partition-based graph abstraction (PAGA) see: https://genomebiology.biomedcentral.com/articles/10.1186/s13059-019-1663-x

cellhub.pipeline_cluster.**UMAP**(*infile*, *outfile*)

Compute the UMAP layout.

cellhub.pipeline_cluster.**plotRdimsFactors**(*infiles*, *outfile*)

Visualise factors of interest on the UMAP.

cellhub.pipeline_cluster.**plotRdimsClusters**(*infile*, *outfile*)

Visualise the clusters on the UMAP

cellhub.pipeline_cluster.**plotRdimsSingleR**(*infile*, *outfile*)

Plot the SingleR primary identity assignments on a UMAP

cellhub.pipeline_cluster.**plotSingleR**(*infile*, *outfile*)

Make singleR heatmaps for the references present on the cellhub API.

cellhub.pipeline_cluster.**summariseSingleR**(*infile*, *outfile*)

Collect the single R plots into a section for the Summary report.

cellhub.pipeline_cluster.**plotGroupNumbers**(*infiles*, *outfile*)

Plot statistics on cells by group, e.g. numbers of cells per cluster.

Plots are defined on a case-by-case basis in the yaml.

cellhub.pipeline_cluster.**clusterStats**(*infile*, *outfile*)

Compute per-cluster statistics (e.g. mean expression level).

cellhub.pipeline_cluster.**findMarkers**(*infile*, *outfile*)

Find per-cluster marker genes. Just execute the rank_genes_groups routine, no filtering here.

cellhub.pipeline_cluster.**summariseMarkers**(*infiles*, *outfile*)

Summarise the differentially expressed marker genes. P-values are adjusted per-cluster are filtering out genes with low expression levels and fold changes. Per-cluster gene universes are prepared for the pathway analysis.

cellhub.pipeline_cluster.**topMarkerHeatmap**(*infiles*, *outfile*)

Make the top marker heatmap.

cellhub.pipeline_cluster.**dePlots**(*infile*, *outfile*)

Make per-cluster diagnoistic differential expression plots (MA and volcano plots).

cellhub.pipeline_cluster.**markerPlots**(*infiles*, *outfile*)

Make the per-cluster marker plots TODO: add some version of split dot plots back..

cellhub.pipeline_cluster.**plotMarkerNumbers**(*infile*, *outfile*)

Summarise the numbers of marker genes for each cluster.

cellhub.pipeline_cluster.**markers**(*infile*, *outfile*)

Target to run marker gene plotting tasks.

cellhub.pipeline_cluster.**parseGMTs**(*param_keys=['gmt_pathway_files_']*)

Helper function for parsing the lists of GMT files

cellhub.pipeline_cluster.**genesetAnalysis**(*infile*, *outfile*)

Naive geneset over-enrichment analysis of cluster marker genes.

Testing is performed with the gsfisher package.

GO categories and KEGG pathways are tested by default.

Arbitrary sets of genes cat be supplied as GMT files (e.g. such as those from MSigDB).

cellhub.pipeline_cluster.**summariseGenesetAnalysis**(*infile*, *outfile*)

Summarise the geneset over-enrichment analyses of cluster marker genes.

Enriched pathways are saved as dotplots and exported in excel format.

cellhub.pipeline_cluster.**genesets**(*infile*, *outfile*)

Intermediate target to run geneset tasks

cellhub.pipeline_cluster.**plots**(*infile*, *outfile*)

Target to run all the plotting functions.

cellhub.pipeline_cluster.**latexVars**(*infiles*, *outfile*)

Prepare a file containing the latex variable definitions for the reports.

cellhub.pipeline_cluster.**summaryReportSource**(*infile*, *outfile*)

Write the latex source for the summary report.

cellhub.pipeline_cluster.**summaryReport**(*infile*, *outfile*)

Compile the summary report to PDF format.

cellhub.pipeline_cluster.**markerReportSource**(*infile*, *outfile*)

Write the latex source file for the marker report.

cellhub.pipeline_cluster.**markerReport**(*infile*, *outfile*)

Prepare a PDF report visualising the discovered cluster markers.

cellhub.pipeline_cluster.**export**(*infile*, *outfile*)

Link output files to a directory in the "reports.dir" folder.

Prepare folders containing the reports, differentially expressed genes and geneset tables for each analysis.

TODO: link in the cellxgene anndata files.

cellhub.pipeline_cluster.**report**()

Target for building the reports.

cellhub.pipeline_cluster.**cellxgene**(*infile*, *outfile*)

Export an anndata object for cellxgene.

5.9 pipeline_dehash.py

5.9.1 Overview

This pipeline dehashes cells which have multiplexed using hash tag oligos (HTOs).

5.9.2 Usage

See Installation and Usage for general information on how to use cgat pipelines.

Configuration

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

Default configuration files can be generated by executing:

cellhub celldb config

5.9.3 Code

cellhub.pipeline_dehash.**gmmDemux**(*infile*, *outfile*)

Run gmmDemux

cellhub.pipeline_dehash.**gmmAPI**(*infiles*, *outfile*)

Register the GMM-Demux results on the API

cellhub.pipeline_dehash.**hashCountCSV**(*infile*, *outfile*)

Make the hash count csv table for demuxEM

cellhub.pipeline_dehash.**demuxEM**(*infile*, *outfile*)

Run demuxEM

cellhub.pipeline_dehash.**parseDemuxEM**(*infile*, *outfile*)

Parse the ridiculous output format from demuxEM.

cellhub.pipeline_dehash.**demuxemAPI**(*infiles*, *outfile*)

Register the demuxEM results on the API

5.10 pipeline_emptydrops.py

5.10.1 Overview

This pipeline performs the following task:

- run emptydrops on the raw output of cellranger

5.10.2 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

5.10.3 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).

5.10.4 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.

5.11 pipeline_fetch_cells.py

5.11.1 Overview

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

5.11.2 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.

Dependencies

This pipeline requires:

5.11.3 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.`**`GEX`**`(`*infile*, *outfile*`)`

Extract the target cells into a single anndata. Note that this currently contains all the modalities

TODO: support down-sampling

`cellhub.pipeline_fetch_cells.`**`ADT`**`(`*infile*, *outfile*`)`

Extract the target cells into a single anndata. Note that this currently contains all the modalities

TODO: support down-sampling

5.12 pipeline_singleR.py

5.12.1 Overview

This pipeline runs singleR for cell prediction. Single R:

(1) runs at cell level (cells are scored independently)

(2) Uses a non-paramentric correlation test (i.e. monotonic transformations of the test data have no effect).

Given these facts, in cellhub we run singleR on the raw counts upstream to (a) help with cell QC and (b) save time in the interpretation phase.

This pipeline operates on the ensembl_ids.

5.12.2 Usage

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

Configuration

The pipeline should be run in the cellhub directory.

To obtain a configuration file run "cellhub singleR config".

Inputs

1. Per-sample market matrix files (from the cellhub API).
2. References for singleR obtained via the R bioconductor 'celldex' library. As downloading of the references is very slow, they need to be manually downloaded and "stashed" as rds files in an appropriate location using the R/scripts/singleR_stash_references.R scripts. This location is then specified in the yaml file.

5.12.3 Pipeline output

The pipeline saves the singleR scores and predictions for each of the specified references on the cellhub API.

5.12.4 Code

`cellhub.pipeline_singleR.`**`genSingleRjobs`**`()`
generate the singleR jobs

`cellhub.pipeline_singleR.`**`singleR`**`(`*infile*, *outfile*`)`
Perform cell identity prediction with singleR.

`cellhub.pipeline_singleR.`**`concatenate`**`(`*infile*, *outfile*`)`
Concatenate the label predictions across all the samples.

`cellhub.pipeline_singleR.`**`summary`**`(`*infile*, *outfile*`)`
Make a summary table that can be included in the cell metadata packages.

`cellhub.pipeline_singleR.`**`singleRAPI`**`(`*infiles*, *outfile*`)`
Add the singleR results to the cellhub API.

5.13 pipeline_velocyto.py

5.13.1 Overview

This pipeline performs the following steps:

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

5.13.2 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

5.13.3 Pipeline output

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

5.13.4 Code

`cellhub.pipeline_velocyto.`**checkInputs**(*outfile*)

Check that input_samples.tsv exists and the path given in the file is a valid directorys.

`cellhub.pipeline_velocyto.`**genClusterJobs**()

Generate cluster jobs for each sample

`cellhub.pipeline_velocyto.`**sortBam**(*infile*, *outfile*)

Sort bam file by cell barcodes

`cellhub.pipeline_velocyto.`**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_velocyto.`**full**()

Run the full pipeline.

CHAPTER

SIX

CELLHUB.TASKS

This sub-module provides helper classes and functions for configuring and running the pipelines.

6.1 parameters.py

6.1.1 Overview

Helper functions for configuring the pipeline parameters.

6.1.2 Functions

cellhub.tasks.parameters.**write_config_files**(*pipeline_path*, *general_path*)

Retrieve default yaml configuration file from:
cellhub/yaml/pipeline_[name].yml

cellhub.tasks.parameters.**get_parameter_file**(*pipeline_path*)

Return the local yml file path if the pipeline is being executed, otherwise return the location of the yml file in the repo.

Note that a local yaml file is mandatory for pipeline execution.

6.2 setup.py

A parent class to help setup pipeline tasks. It can be extended to meet the needs of the different pipelines. The class is used to obtain a task object that:

- defines job resource requirements
- provides access to variables (by name or via a .var dictionary)
- creates an outfolder based on the outfile name

class cellhub.tasks.setup.**setup**(*infile*, *outfile*, *PARAMS*, *memory='4G'*, *cpu=1*, *make_outdir=True*, *expose_var=True*)

Bases: `object`

A class for routine setup of pipeline tasks.

Parameters

- **infile** – The task infile path or None

- **outfile** – The task outfile path (typically ends with ".sentinel")
- **memory** – The total memory needed for execution of the task. If no unit is given, gigabytes are assumed. Recognised units are "M" for megabyte and "G" for gigabytes. 4 gigabytes can be requested by passing "4", "4GB" or "4096M". Default = "4GB".
- **cpu** – The number of cpu cores required (used to populate job_threads)
- **make_outdir** – True|False. Default = True.
- **expose_var** – True|False. Should the self.var dictionary be created from self.__dict__. Default = True.

job_threads

The number of threads that will be requested

job_memory

The amount of memory that will be requested per thread

resources

A dictionary with keys "job_threads" and "job_memory" for populating the P.run() kwargs, e.g. `P.run(statement, **t.resources)`

outname

The os.path.basename of outfile

outdir

The os.path.dirname of outfile

indir

If an infile path is given, the os.path.dirname of the infile.

inname

If an infile path is given, the os.path.basename of the infile.

log_file

If the outfile path ends with ".sentinel"

parse_mem(*memory*)

Return an integer that represents the amount of memory needed by the task in gigabytes.

set_resources(*PARAMS*, *memory='4G'*, *cpu=1*)

calculate the resource requirements and return a dictionary that can be used to update the local variables

6.3 api.py

6.3.1 Overview

The primary and secondary analysis pipelines define and register their outputs via a common api. The api comprises of an "api" folder into which pipeline outputs are symlinked (by the "register_dataset" "api" class method).

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 available 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.

6.3.2 Usage

Registering outputs on the service endpoint

- All matrices registered on the API that hold per-cell statistics must have "library_id" and "barcode" columns. The library identifiers must correspond with those given in the "libraries.tsv" file. The barcodes field should contain the untouched Cellranger barcodes.

Please see *pipeline_cellranger.py* or *pipeline_cell_qc.py* source code for examples of how to register results on the API.

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",
                 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 available datasets

At present the API can be browsed on the command line. Programmatic access is expected in a future update.

Accessing datasets

At present datasets are accessed directly via the "api" endpoint.

6.3.3 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",
            "link_name": "api link name", # optional
            "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"},
{"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

6.4 profile.py

Parse the pipeline.log file from a cgat-core pipeline and summarise information on task resource usage.

6.4.1 Usage

After running a pipline, the resources used by the pipeline tasks can be summarised with the "cellhub pipeline_name profile" command, e.g.

```
> cellhub cluster make full -v5 -p 100
> cellhub cluster profile
```

6.4.2 Code

cellhub.tasks.profile.**is_tool**(*name*)

Check whether *name* is on PATH and marked as executable.

6.5 cellbender.py

6.5.1 Overview

This module contains helper functions for pipeline_cellbender.py

cellhub.tasks.cellbender.**anndata_from_h5**(*file: str*, *analyzed_barcodes_only: bool = True*) → AnnData

Load an output h5 file into an AnnData object for downstream work.

Parameters

- **file** – The h5 file
- **analyzed_barcodes_only** – False to load all barcodes, so that the size of the AnnData object will match the size of the input raw count matrix. True to load a limited set of barcodes: only those analyzed by the algorithm. This allows relevant latent variables to be loaded properly into adata.obs and adata.obsm, rather than adata.uns.

Returns

The anndata object, populated with inferred latent variables
and metadata.

Return type
adata

cellhub.tasks.cellbender.**dict_from_h5**(*file: str*) → Dict[str, ndarray]

Read in everything from an h5 file and put into a dictionary.

CHAPTER

SEVEN

CONTRIBUTING

Contributions to the code and documentation are welcome from all.

7.1 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
cellhub/reports	The latex source files used for building the reports
Python	Python worker scripts that are executed by pipeline tasks
R/cellhub	The R cellhub library
R/scripts	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

7.2 Style guide

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

7.2.1 Python

- Python code should be pep8 compliant. Compliance checks will be enforced soon.
- Arguments to Python scripts should be parsed with argparse.
- Logging in Python scripts should be performed with the standard library "logging" module, written to stdout and redirected to a log file in the pipeline task.

7.2.2 R

- R code should follow the tidyverse style guide. Please do not use right-hand assignment.
- Arguments to R scripts should be parsed with optparse.
- Logging in R scripts should be performed with the "futile.logger" library, written to stdout and redirect to a log file specified in the pipeline task.
- Otherwise, to write to stdout from R scripts use message() or warning(). Do not use print() or cat().

7.3 Writing pipelines

The pipelines live in the "cellhub" python module.

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

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 are 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 public 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: please maintain informative rst doc-strings.

7.4 Writing pipeline tasks

Helper functions for writing pipelines and tasks live in the *cellhub.tasks module*.

In cellhub, pipeline tasks are written using the *cellhub.tasks.setup module* as follows:

```
import ruffus
import cgatcore.pipeline as P
import cgatcore.iotools as IOTools
import cellhub.tasks as T

PARAMS = P.get_parameters(....)

@files("a.txt", "results.dir/my_task.sentinel")
def my_task(infile, outfile):
    '''Example task'''

    t = T.setup(infile, outfile, PARAMS,
                memory="4GB", cpu=1, make_outdir=True)

    results_file = outfile.replace(".sentinel", ".tsv.gz")
```

(continues on next page)

(continued from previous page)

```
    statement = ''' python code.py
                    --arg1=%(infile)s
                    --args=%(parameter_x)s
                    --arg2=%(outdir)s
                    --arg3=%(results_file)s
                    &> %(log_file)s
                ''' % dict(PARAMS, **t.var, **locals())

    P.run(statement, **t.resources)

    IOTools.touch_file(outfile)
```

As shown in the example, the following conventions are adopted:

1. The task output is an empty sentinel file. It will only be written if the task returns without an error. This ensures that the pipeline does not proceed with partial results.
2. An instance, "t", of the cellhub.tasks.setup class is created. Based on the arguments provided, it is populated with useful variables (see above), including the parsed resource requirements. By default, the class constructor will create the output directory (if it does not already exist) based on the outfile name.
3. The stderr and stdout are captured to a log file. By default t.log_file is populated with the outfile name with ".sentinel" replaced by ".log".
4. The statement is substituted with variables from the PARAMS, t.var and locals() dictionaries as required. Note that variable names must be unique across the dictionaries provided.
5. The resources needed are passed to P.run() as kwargs via the t.resources dictionary.

7.5 Cell indexing

- Tables of per-cell information registered on the API must have columns "barcode" (for the original Cellranger assigned barcode, "-1" suffix not removed) and "library_id". These two columns are used by pipeline_celldb.py to join the tables in the database.
- Downstream of fetch_cells, when cells are aggregated across libraries, we use unique cell identifiers made by combining the barcode and library id in [AGCT]-library_id format (with the "-1" suffix now removed from the original barcode). The unique cell identifiers are used to populate the anndata.obs.index and the "barcode_id" column in tsv files where needed.

7.6 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

7.7 Compiling the documentation locally

To be described.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

N

O

P

Q

R

S

T

U

W