cellbrowser Documentation

Release v0.7.10+12.ge854919

Maximilian Haeussler, Lucas Seninge, Nikolai Markov

Jun 05, 2020

Contents

The UCSC Cell Browser is a fast, lightweight viewer for single-cell data. Cells are presented along with metadata and gene expression, with the ability to color cells by both of these attributes. Additional information, such as cluster marker genes and selected dataset-relevant genes, can also be displayed using the Cell Browser.

There is a UCSC Cell Browser website available at http://cells.ucsc.edu, which includes a handful of datasets from repositories like HCA, CIRM, and GEO as well as user contributed ones. We are happy to add your favorite dataset to this, you will just need to send us the files or a link to where we can download them to cells@ucsc.edu.

The documentation on this website describes how you can create a Cell Browser for your own data and make it available through your own web server.

The UCSC cell browser is funded by grants from the California Institute for Regenerative Medicine and the Chan-Zuckerberg Initiative.

To report issues or view the source code, see GitHub.

This is early research software. You are likely to run into bugs. If you do run into any trouble, please open a Github issue or email us at cells@ucsc.edu, we can usually fix them quickly.

CHAPTER 1

Installation

1.1 Installation with pip

To install the Cell Browser using pip, you will need Python2.5+ or Python3+ and pip. With these setup, on a Mac or any Linux system, simply run:

```
sudo pip install cellbrowser
```

On Linux, if you are not allowed to run the sudo command, you can install the Cell Browser into your user home directory:

```
pip install --user cellbrowser
export PATH=$PATH:~/.local/bin
```

You can add the second command to your ~/.profile or ~/.bashrc, this will allow you to run the Cell Browser commands without having to specify their location.

On OSX, if running `sudo pip` outputs *command not found*, you will need to setup pip first by running:

```
sudo easy_install pip
```

1.2 Installation with conda

If you would prefer to install the Cell Browser through bioconda, you can run:

```
conda install -c bioconda ucsc-cell-browser
```

There should be conda versions for release 0.4.23 onwards. The conda version is managed by Pablo Moreno at the EBI and is often a few releases behind. Please indicate in any bug reports if you used conda to install.

1.3 Installation with git clone

Pip is not required to install the Cell Browser. As an alternative to pip or conda, you can also git clone the repo and run the command line scripts under cellbrowser/src:

```
git clone https://github.com/maximilianh/cellBrowser.git --depth=10
cd cellBrowser/src
```

1.4 Installation with just wget or curl

You don't use pip, conda or git? You can also download the current master branch:

```
wget https://github.com/maximilianh/cellBrowser/archive/master.zip
unzip master.zip
cellBrowser-master/src/cbBuild
```

CHAPTER 2

Basic usage

2.1 Overview

The UCSC Cell Browser tool set consists of a number of different scripts to help you set up your own. The primiary utility being the Python script `cbBuild` that will import a set of existing single-cell data from a directory of tab-separated files and configuration files to generate a directory of html, json, and css files that can be viewed on the web. The rest of the utilities will produce output than can be fed directly into `cbBuild`.

The utilities `cbSeurat` and `cbScanpy` run a very basic single-cell pipeline on your expression matrix and will output all the files needed to create a cell browser visualization. The `cbImport*` (`cbImportCellranger`, `cbImportScanpy`, etc.) tools convert files produced by Cellranger, Seurat, and Scanpy into a set of files that you can create a Cell Browser visualization from. Both the pipeline and import tools are covered in more detail under their respective sections (With Scanpy, With Seurat, and With Cellranger). There is also a collection of small tools (`cbTool`) to combine cell annotation files from different pipelines or convert expression matrices.

2.2 Using cbBuild to set up a Cell Browser

The main utility for building your own cell browser is `cbBuild`. It takes in a gene expression matrix and a set related files and converts them JSON and binary files outputting them to directory which can be put onto a web server or used with the built-in webserver. At this time, there is no backend server needed for a cell browser. You can place the output of `cbBuild` on any static web server at your University or the ones you can rent from companies will do.

After the installation, you should be able to run the cbBuild command and see the usage message:

```
cbBuild
```

2.2.1 Example Minimal Cell Browser

Below are some instructions to set up a cell browser using a small example dataset based on data from Nowakowski et al. 2017. and the cortex-dev dataset on cells.ucsc.edu. The expression matrix only includes 100 genes, but it does show off many of the features of the cell browser.

First, download and extract it to the directory `mini` with:

```
curl -s https://cells.ucsc.edu/downloads/samples/mini.tgz | tar xvz
```

Next, build a browser consisting of html and other files into the directory ~/public_html/cells/ and serve that directory on port 8888:

```
cd mini
cbBuild -o ~/public_html/cells/ -p 8888
```

Lastly, point your web browser to `http://localhost:8888` to view your minimal cell browser. If you're running this on a server and not your own computer, replace localhost with the address of your server. To stop the cbBuild web server, press Ctrl-C. To keep it running in the background, press Ctrl-Z and put it into the background with `bg`. If you have stopped the web server, you can always run the same `cbBuild` command to restart it. Restarting the web server will not re-export the entire expression matrix again if there is already one under `~/public_html/cells/sample`.

The optional to specify the port, `-p PORT`, is optional. If you only want to build html files and serve them with your own web server, do not specify this option and `cbBuild` will only build the output files, but won't start a web server.

The example cellbrowser.conf explains all the various settings that are available in this config file. Things you can change include the colors for different metadata attributes, explain cluster acronyms used in your cluster names, add file names, add alternative dimensionality reduction layouts, add more marker gene tables, and more.

One of the most important settings in cellbrowser.conf is the dataset name. For example, in this 'mini' example, the dataset name is 'sample'. When you run `cbBuild`, its output files will be written to ~/public_html/cells/sample. You can go to another directory with a different cellbrowser.conf file and a different dataset name, and if you run the same cbBuild command as above, the cell browser output files will be copied into a new subdirectory within ~/public_html/cells/. A single cbBuild output directory can contain multiple datasets.

CHAPTER 3

Putting it onto the internet

3.1 Basics

Deploying your cell browser on the web is as simple as copying the output of `cbBuild`, including all files and directories, into to an empty directory on a web server. The cell browser should be able to be deployed on almost any web server, including:

- One provided by your university (often through a public_html directory in your home directory)
- Github.io provides free hosting for 1 GB of webspace and it is fast enough, see https://sansomlab.github.io/
- You can also use commercial cloud providers, such as Amazon S3, Google Cloud Storage, Microsoft Azure, though they will need a credit card.

Please consider also sending the output files to cells@ucsc.edu, we are more than happy to add it to our public Cell Browser website. You can choose a dataset prefix, and then can add myDataset.cells.ucsc.edu to your manuscript. Unfortunately, online backup solutions such as Dropbox, Box.com, iCloud, OneDrive or Google Drive will not work; they are intentionally designed to not be usable as web servers.

3.2 Adding multiple datasets to your cell browser

To add more datasets to the same cell browser, navigate to the other data directories and run cbBuild there with the same output directory. cbBuild will then modify the index.html in the output directory to show all datasets. Note that the directory that you provide via -o (or the CBOUT environment variable) is the html directory. The data for each individual dataset will be copied into subdirectories under this html directory, one directory per dataset.

3.3 Specifying a default output directory for `cbBuild`

The output directory for `cbBuild` can be controlled using environment or .conf variables. This allows you to run `cbBuild` in a directory without needing to specify an output directory using the "-o" option.

To control the output directory using an environment variable, add the following line to your ~/.bashrc to point to your html directory:

```
export CBOUT=/var/www
```

Replace `/var/www` with whatever you want the default output directory to be.

Alternatively, you can create a file called `.cellbrowser.conf` in your home directory and assign a value to htmlDir:

```
echo 'htmlDir = "/var/www"' >> ~/.cellbrowser.conf
```

Again, replace `/var/www` with your own dorectory.

3.4 Notes on setting up a permanent cell browser on a local machine

The port option, e.g. `-p 8888`, is optional. When this option is specified, it will start up its own web server. If you are running this on your local machine, a more permanent alternative to the -p option is to run a web server on your machine and then build directly into its web directory.

On a Mac, you can use the Apache that ships with OSX:

```
sudo /usr/sbin/apachectl start
sudo cbBuild -o /Library/WebServer/Documents/cells/
```

You should be able to access your viewer at http://localhost/cells

On Linux, you will need to install Apache2 (with `sudo yum install httpd` or `sudo apt-get install apache2`) and use the directory `/var/www/` instead:

```
sudo cbBuild -o /var/www/
```

We hope you do not use this software on Windows. Email cells@ucsc.edu if you have to.

CHAPTER 4

With text files

The generic way to set up your own cell browser is to start from tab-separated (tsv) or comma-separated (csv) format text files. The steps on this page assume that you have already gone through the process of clustering your cells.

4.1 The files you will need

You will need the first three files described below in tsv or csv format, the fourth is optional:

1. **Expression matrix**: one row per gene and one column per cell, ideally gzipped. The first column must be the gene identifier or gene symbol, or ideally geneId|symbol. Ensembl/GENCODE gene identifiers starting with ENSG and ENSMUSG will be translated automatically to symbols. The other columns are expression values as numbers, one per cell. The number type will be auto-detected (float or int). The first line of the file must be a header that includes the cell identifiers.

2. **Cell annotation metadata table**, one row per cell. No need to gzip this relatively small file. The first column is the name of the cell and it has to match the name in the expression matrix. There should be at least two columns: one with the name of the cell and one with the name of the cluster. To speed up processing of both your expression matrix and metadata file, these files should describe the same numbers of cells and be in the same order. This allows cbBuild process these files without needing to trim the matrix and reorder the metadata file. The metadata file also must have a header line.

3. **Cell coordinates**, often t-SNE or UMAP coordinates. This file always has three columns, (cellName, x, y). The cellName must be the same as in the expression matrix and cell annotation metadata file. If you have run multiple dimensionality reduction algorithms, you can specify multiple coordinate files in this format. The number rows in these coordinates doesn't need to match that of your expression matrix or metadata files, allowing you to specify only a subset of the cells. In this way, you can use a single dimensionality reduction algorithm, but include multiple subsets and view of the cells, e.g. one coordinates file per tissue. Note, if R has changed your cell identifiers (e.g. by adding dots), you may be able to fix them by running `cbTool metaCat`.

4. **Cluster-specific marker genes** (optional). The first column is the cluster name (from the cell annotation metadata file), the second column contains the gene symbol (or Ensembl gene ID, which will automatically be mapped to the gene symbol), and the third column is some numeric score (e.g. p-Value or FDR). You can add as many other columns as you like with additional information about this gene. You can also run

`cbMarkerAnnotate` on this file to add information from various gene-centric databases and link-outs to other resources to your existing table. See Annotate genes on how to add link to external gene-databases (like Allan Brain Atlas or OMIM) to your marker genes. If you used Seurat for your clustering, you can just provide the raw Seurat marker gene output. You can also specify multiple files of cluster-specific marker genes, e.g. in case that you are also doing differential gene expression analysis or have results from multiple algorithms.

Make sure that all your input files have Unix line endings and fix the line endings if necessary with mac2unix or dos2unix:

```
file *.txt *.csv *.tsv *.tab
```

4.2 Setting up your `cellbrowser.conf`

After you have all of these files in place, go to the directory go to the directory containing all of these files and run the following command to copy a sample `cellbrowser.conf` into your current directory:

```
cbBuild --init
```

This sample `cellbrowser.conf` includes the required settings plus some other useful settings. The current list of all possible cellbrowser.conf statements can be found in our example cellbrowser.conf.

In your cellbrowser.conf, replace the default values in the config statements:

- `exprMatrix` - expression matrix file name
- `meta` - cell annotation metadata file name
- `coords` - coordinate file names with a layout method label for each
- `markers` - cluster-specific marker gene file namew with a label for each
- `labelField` and `clusterField` - name of cluster field from header line of metadata file

From the directory where your `cellbrowser.conf` is located, run:

```
cbBuild -o /tmp/cb -p 8888
```

Point your internet browser to the name of the server (or localhost, if you're running this on your own machine) followed by :8888, e.g. http://localhost:8888.

The cell browser output directory (/tmp/cb in this example) can hold multiple datasets. If you have a second dataset in another directory that contains cellbrowser.conf, just make sure that the other cellbrowser.conf specifies a different dataset name with *name=xxx*. Then run *cbBuild -o /tmp/cb -p 8888* in the other directory to add the second dataset to your cell browser output directory /tmp/cb.

CHAPTER 5

With Seurat

There are a number of ways to create a cell browser using Seurat:

- **Import a Seurat rds file** - create a cell browser with the Unix command line tool `cbImportSeurat`.
- **Using RStudio and a Seurat object** - create a cell browser directly using the `ExportToCellbrowser()` R function.
- **Run our basic Seurat pipeline** - with just an expression matrix, you can run our `cbSeurat` pipeline to create a cell browser.

Each of these methods are described in more detail below.

5.1 Convert a Seurat `rds` file

First, create an .rds file in R as described in the Seurat tutorial:

```
saveRDS(pbmc, "pbmc3k_small.rds")
```

Next, on the Unix command line, use the `cbImportSeurat` script to convert this `rds` file into a cell browser:

```
cbImportSeurat -i pbmc3k_small.rds -o pbmc3kImport
```

This works with objects created by versions 2 and 3 of Seurat. Make sure that you have the same version of Seurat installed that was used to create the object.

The `-i` option specifies the input `rds` file and the `-o` option specifies a name for the output directory. You can use the `-n` option to change the dataset name in the cell browser; if it is not specified, it will default to the output directory name.

A Seurat object does not contain the marker genes by default, as FindAllMarkers() does not save its output. You can add it to the object when you save the .rds file with a command like this:

```
object@misc$markers <- FindAllMarkers(object)
```

`cbImportSeurat` will then use these markers. Otherwise, if `misc$markers` is not present in the object, it will run FindAllMarkers with the default values (Wilcoxon and 0.25 as the cutoff). Alternatively, you can also save the markers to a tab-separated file yourself and provide this file with the `--markerFile` option.

Lastly, go into the `pbmc3kImport` directory and run `cbBuild` to create the cell browser output files:

```
cd pbmc3kImport
cbBuild -o ~/public_html/cb
```

Alternatively, you can use the `--htmlDir` option for `cbImportSeurat` to automatically run `cbBuild` for you:

```
cbImportSeurat -i pbmc3k_small.rds -o pbmc3kImport --htmlDir=~/public_html/cb
```

5.2 Convert a Seurat object from R

The function ExportToCellbrowser() is already part of Seurat 3. You can install pre-release Seurat3 like this:

```
install.packages("devtools")
devtools::install_github("satijalab/seurat", ref = "release/3.0")
```

For Seurat 2, you have to load the function with this command:

```
source("https://raw.githubusercontent.com/maximilianh/cellBrowser/master/src/cbPyLib/
↪cellbrowser/R/ExportToCellbrowser-seurat2.R")
```

You can then write a Seurat object to a directory from which you can run cbBuild:

```
ExportToCellbrowser(pbmc_small, dir="pbmcSmall", dataset.name="pbmcSmall")
```

Or, you can build a cell browser from this dataset into the `htdocs` directory, serve the result on port 8080 via http, and open a web browser from within R:

```
ExportToCellbrowser(pbmc_small, dir="pbmcSmall", cb.dir="htdocs", dataset.name=
↪"pbmcSmall", port=8080)
```

Writing the expression matrix is somewhat slow. If you have already exported into the same output directory before and just updated a part of the cell annotation data (e.g. clustering), you can use the argument *skip.matrix=TRUE* to save some time:

ExportToCellbrowser(pbmc_small, dir="pbmcSmall", dataset.name="pbmcSmall", skip-matrix=TRUE)

5.3 Run a basic Seurat pipeline

If you have never used Seurat before and just want to process an expression matrix as quickly as possible, this section is for you.

If you do not have R installed yet, we recommend that you install it via conda. To install miniconda, follow their installation instructions.

After setting up conda, install R:

```
conda install r
```

Then, install Seurat:

```
conda install -c bioconda r-seurat
```

To process an example dataset now, download the 10X pbmc3k expression matrix:

```
rsync -Lavzp genome-test.gi.ucsc.edu::cells/datasets/pbmc3k/ ./pbmc3k/ --progress
```

Now run the expression matrix `filtered_gene_bc_matrices/hg19/matrix.mtx` through Seurat:

```
cbSeurat -e filtered_gene_bc_matrices/hg19 --name pbmc3kSeurat -o seuratOut
```

This will create a script (`seuratOut/runSeurat.R`), run it through Rscript, and will fill the directory `seuratOut/` with everything needed to create a cell browser. After the `cbSeurat` script completes, you can build your cell browser from the output:

```
cd seuratOut
cbBuild -o ~/public_html/cells
```

5.3.1 Changing the defaults using `seurat.conf`

This set of steps will run a basic Seurat pipeline with the default settings. You can modify the settings for Seurat by creating a `seurat.conf` file:

```
cbSeurat --init
```

You can edit the settings in `seurat.conf` and re-run the `cbSeurat` command to generate a new set of Seurat output using these new settings.

CHAPTER 6

With Scanpy

There area few different ways to create a cell browser using Scanpy:

- **Run our basic Scanpy pipeline** - with just an expression matrix and `cbScanpy`, you can the standard preprocessing, embedding, and clustering through Scanpy.
- **Import a Scanpy h5ad file** - create a cell browser from your `h5ad` file using the command-line program `cbImportScanpy`.
- **Use a few Python 3 function** - you can build a cell browser from a Scanpy `h5ad` file and start a web server, e.g. from Jupyter, with the Python3 function `cellbrowser.scanpyToCellbrowser(ad, outDir, datasetname)`.

6.1 A standard Scanpy pipeline

Requirements: Python3 with Scanpy installed, see their installation instructions for information about setting up Scanpy. As part of the Scanpy installion process, ensure that the igraph library is also installed. It's needed for the most basic scanpy features even though it's not an official requirement. The command `pip install scanpy[louvain]` will make sure that igraph is installed.

We provide a wrapper around Scanpy, named `cbScanpy`, which runs filtering, PCA, nearest-neighbors, clustering, t-SNE, and UMAP. The individual steps are explained in more detail in the Scanpy PBMC3k tutorial.

The output of `cbScanpy` is formatted to be directly usable to build a cell browser with `cbBuild`.

You can test `cbScanpy` yourself using the following set of steps. To process an example dataset, download the 10x pbmc3k expression matrix from our servers:

```
mkdir ~/cellData
cd ~/cellData
rsync -Lavzp genome-test.gi.ucsc.edu::cells/datasets/pbmc3k/ ./pbmc3k/ --progress
cd pbmc3k
```

Next, run the expression matrix `filtered_gene_bc_matrices/hg19/matrix.mtx` through Scanpy:

```
cbScanpy -e filtered_gene_bc_matrices/hg19/matrix.mtx -o scanpyOut -n pbmc3k
```

This will run Scanpy and will fill the directory `scanpyOut/` with everything needed to create a cell browser. After the `cbScanpy` script completes, you can build your cell browser from the output:

```
cd scanpyout
cbBuild -o ~/public_html/cb -p 8888
```

6.1.1 Changing the defaults using `scanpy.conf`

This set of steps will run a basic Scanpy pipeline with the default settings. You can modify the settings for Scanpy by creating a `scanpy.conf`:

```
cbScanpy --init
```

You can edit the settings in `scanpy.conf` and re-run the `cbScanpy` command to generate a new set of Scanpy output using these new settings.

6.2 Convert a Scanpy `h5ad`

If you have run Scanpy and have an output `h5ad` file, you can import it into a cell browser using the command `cbImportScanpy`.

The steps in this section walk you through the process of importing data from a Scanpy file and then building a cell browser from the output. The steps use an example `h5ad` file available for a small pbmc dataset from our Github repo: anndata.h5ad.

First, use `cbImportScanpy` to extract the data from the `h5ad`:

```
cbImportScanpy -i anndata.h5ad -o pbmc3kImportScanpy
```

The `-i` option specifies the input `h5ad` file and the `-o` option specifies a name for the output directory. You can use the `-n` option to change the dataset name in the cell browser; if it is not specified, it will default to the output directory name.

The output of `cbImportScanpy` will be formatted so that you can immediately build a cell browser from it. Go into the pbmc3kImportScanpy directory and run `cbBuild` to create the cell browser output files:

```
cd pbmc3kImportScanpy
cbBuild -o ~/public_html/cb
```

Alternatively, you can use the `--htmlDir` option for `cbImportScanpy` to automatically run cbBuild for you:

```
cbImportScanpy -i anndata.h5ad -o pbmc3kImportScanpy --htmlDir=~/public_html/cb
```

6.3 Convert a Scanpy object

From Jupyter or Python3, you can create a data directory with the necessary tsv files and a basic `cellbrowser.conf`:

```
import cellbrowser.cellbrowser as cb
cb.scanpyToCellbrowser(adata, "scanpyOut", "myScanpyDataset")
```

Here `adata` is your Scanpy object, `scanpyOut` is your output directory, and `myScanpyDataset` is your dataset name.

Then, build the cell browser from this output directory into a html directory:

```
cb.build("scanpyOut", "~/public_html/cells")
```

If you don't have a web server running already, use this function start one to serve up this directory:

```
cb.serve("~/public_html/cells", 8888)
```

You can stop the web server with the function:

```
cb.stop()
```

Or from a Unix shell, you can build and start a web server using `cbBuild`:

```
cd scanpyOut
cbBuild -o ~/public_html/cells/ -p 8888
```

CHAPTER 7

With Cell Ranger

Find the `cellranger` **OUT** directory, it should contain an `analysis` directory and a subdirectory `filtered_gene_bc_matrices`. The **OUT** directory will be the input for our tool `cbImportCellranger`. The tool converts the `cellranger` files to ones formatted for `cbBuild`.

As we are reading Cell Ranger `mtx` files, we need the scipy package (add `--user` if you are not the admin on your machine):

```
pip install scipy
```

The example below use the pbmc3k `cellranger` output files from the 10x website. First, download the files with the command:

```
rsync -Lavzp genome-test.gi.ucsc.edu::cells/datasets/pbmc3kCellranger/ ./
↪pbmc3kCellranger/ --progress
```

Next, run `cbImportCellranger` to convert the `cellranger` files into something that can be used to build a cell browser:

```
cbImportCellranger -i pbmc3kCellranger -o cellrangerOut --name pbmc3k_cellranger
```

The `-i` option specifies the input `cellranger` directory and the `-o` option specifies a name for the output directory. You can use the `-n` option to change the dataset name in the cell browser; if it is not specified, it will default to the output directory name.

Lastly, go into the `cellrangerOut` directory and run `cbBuild` to create a cell browser:

```
cd cellrangerOut
cbBuild -o ~/public_html/cells -p 9999
```

CHAPTER 8

Advanced Topics

8.1 Cell browser updates and web server configuration

New features and bug fixes are being added to the UCSC Cell Browser software all the time. You can update the javascript files and re-create the index.html using the command line tool `cbUpgrade`.

Your web server should support byte-range requests. This isn't important for smaller, but for datasets with files larger than 30MB, a warning message will be shown once. For Apache, byte-range requests are enabled by default but may need to be activated in nginx.

8.2 Default output directory for building cell browsers

The html directory can be defined in all tools with the option `-o`. If that becomes cumbersome, you can also permanently set it through the environment variable CBOUT (e.g. in your `~/.bashrc`) or by adding a line like this to `~/.cellbrowser.conf`:

```
htmlDir = "/data/www/cb/"
```

8.3 Google Analytics

To add Google Analytics tracking to your cell browser, create a file `.cellbrowser.conf` in your home directory and add a line like this:

```
gaTag = "UA-11231232-1"
```

Then run `cbBuild` or `cbUpgrade` to rebuild your index.html, after which it should contain your Google Analytics tracking code.

8.4 Various `cellbrowser.conf` configurations

For a reference of all possible `cellbrowser.conf` statements, see the example conf

If you have meta fields with very long names, you can reduce the font size. Configure them like this:

metaOpt = {'Cluster_field' : {'fontSize': '10px'}}

In your meta.tsv, you can have URLs to images. These will be shown on mouse over in the left annotation bar.

If you set the default coloring field to 'None' (without the quotes), then there is no coloring at all when the cell browser starts.

To change the coloring/label field automatically when the user activates some coordinates (layout), use the option "colorOnMeta" to specify the field:

```
coords=[
    {"file":"tsne.coords.tsv", "shortLabel":"t-SNE on WGCNA"},
    # you can force coloring of some other meta data field when a layout is changed␣
↪to another one
    {"file":"subset.coords.tsv", "shortLabel":"neural cells", colorOnMeta=
↪"neuralCluster"},
]
```

In very rare cases, it can be necessary to tell `cbBuild` that the numbers in the matrix are floating point numbers. The setting looks like this:

```
matrixType = "float"
```

CHAPTER 9

Annotate Genes

When you load a basic set of marker genes into your cell browser, they will be imported as gene symbols along with an associated score, with no extra annotations. To make the list of markers genes more useful, you can add extra annotations using the `cbMarkerAnnotate` tool. This tool will add information about in other resources that describe gene expression profiles (Allen Brain Atlas), diseases they have been linked to (OMIM, HPO, Sfari), and protein class (HPRD).

Running this script on your marker genes file is very simple:

```
cbMarkerAnnotate inFname outFname
```

The format for `inFname` is the same as for standard cell browser marker gene files, a tsv or csv table with at least three columns, in this order:

1. cluster - needs to match `labelField` in `cellbrowser.conf`.
2. gene - can be a gene symbol or Ensembl gene ID, with or without the version.
3. score - scores are typically "avg_diff" or "p-Value" or similar. Gene

`cbMarkerAnnotate` will map Ensembl gene IDs to symbols and then lookup various gene-related databases to add more columns to `inFname` and write the result to `outFname`. You can then change the marker genes file described by the `markers` parameter in your `cellbrowser.conf` to point to `outFname`.

You can also add your own annotations (e.g. number of associated PubMed articles) to your marker genes files. These will be displayed alongside any other annotations that you may have added using `cbMarkerAnnotate`.

CHAPTER 10

`cbTool`: combine and convert your data

The script `cbTool` included in the Cell Browser package includes a number of utilties for combining or converting your data. These different functions and how to use them are described below.

10.1 Combining results

10.1.1 Metadata

You can use the `cbTool metaCat` utility to merge the metadata files from different sources or pipelines (e.g. `cbScanpy` or `cbSeurat`) into a single one. A command to do combine a metadata set of metadata files from Scanpy and Seurat with a separate one might look like this:

```
cbTool metaCat myMeta.tsv seuratOut/meta.tsv scanpyOut/meta.tsv ./newMeta.tsv --fixDot
```

The resulting file will include the columns from all three of the original files combined into a new metadata file. Note that `cbTool metaCat` assumes that the first column of each file contains the same cell identifier that it can use to join them.

10.1.2 Matrices

Similar to `metaCat` for combining metadata files, `cbTool matCat` can be used to combine expression matrices from different sources. A command combine the two different matrixes would look like this:

```
cbTool matCat mat1.tsv.gz mat2.tsv.gz exprMatrix.tsv.gz
```

10.2 Converting mtx to tsv

Using `cbTool mtx2tsv`, you can convert your expression matrix in matrix market to tsv format (with one gene per line and one cell per column):

```
cbTool mtx2tsv matrix.mtx genes.tsv barcodes.tsv exprMatrix.tsv.gz
```

10.3 Fixing R Seurat output

The option `--fixDot` for `cbTool` will work around R's strange habit of replacing special characters in the cell identifiers with ".". Directories created with the `ExportToCellbrowser()` function from R should not have this problem, but others may.

——— REQUIRED SETTINGS ———

example config file with all possible settings

internal short name, only visible in the URL # same as the output directory name # no special chars, no whitespace, please name = "sample"

priority determines the order of the datasets # smallest comes first priority = 10

tags are shown in the dataset browser # current tags: # smartseq2,10x tags = ["smartseq2"]

human-readable name of this dataset shortLabel="CellBrowser 100-genes demo"

name of the expression matrix file, genes are rows exprMatrix="exprMatrix.tsv.gz"

"gencode-human", "gencode-mouse" or "symbol" # For "symbol" you can specify which database to use to check # symbols or, for cbHub, how to map them to the genome. # 'auto' will automatically detect Ensembl human/mouse IDs # and translate to symbols geneIdType="auto"

name of the meta data table ("samplesheet). One sample per row. First row is name of sample. meta="meta.tsv"

we try to auto-detect the field type of fields in the meta data. # Sometimes, this doesn't work, e.g. when your cluster ID is a numer # or your C1 chip ID is a number, but you don't want them binned, you want # to treat as if they were categories enumFields = ["c1_cell_id"]

tsv files with coordinates of every sample in format <sampleId, x, y> # first the name of the file, then a human readable description coords=[

{ "file":"tsne.coords.tsv", "flipY" : False, # R/Matplotlib files need to be flipped on the Y-axis "shortLabel":"t-SNE on WGCNA"

}, {

"file":"subset.coords.tsv", "shortLabel":"neural cells", # you can overlay lines onto the cells, table has to have columns named x1, x2, y1, y2 "lineFile" : "lines.tsv", # you can flip the y-axis of just the lines, relative to the points # This was necessary for a user when using the files produced by the URD pseudotime package #"lineFlipY" : True, # you can automatically switch on coloring on a meta data field whenever a layout is activated "colorOnMeta":"neuralCluster"

},

]

default field in the meta data table with the name of the cluster clusterField="WGCNAcluster"

default field in the meta data table used for the label of the clusters shown by default labelField="WGCNAcluster"

——— OPTIONAL SETTINGS ———

genes that are highlighted in your paper can be pre-loaded and are shown as a clickable table on the left # this is optional but we highly recommend that you define at least 2-3 quick genes, it makes the browser a lot # more intuitive for users quickGenesFile = "quickGenes.csv"

if you want to enforce some order of the values of your enums, e.g. your cluster annotation should be sorted # in a given order in the display, supply a text file with the values in the right order, one per line. # You can supply one text file per meta data field # enumOrder = { "WGCNAcluster" : "clusterorder.txt" }

tsv files with marker gene lists for the clusters # format is (clusterName, geneSymbol, pValue, enrichment) + any additional fields or URLs you want to show markers=[

{"file":"markers.tsv", "shortLabel":"Cluster-specific markers"}

]

do not show this dataset on the dataset list. This can be used for pre-publication data. # hideDataset=True

optional: UCSC track hub with the BAM file reads and expression values # Alternatively, you can also provide a full link to a UCSC Genome Browser session here hubUrl="http://cells.ucsc.edu/cortex-dev/hub/hub.txt"

optional: table with <name><color> for any meta data values # color is a six-digit hexcode # name is a any value in the meta data table, e.g. cluster name. Canb be a .tsv or .csv file. colors="colors.tsv"

should the cluster labels be shown by default (default: true) showLabels=True

the radius of the circles. If not specified, reasonable defaults will be used #radius = 5 # the alpha/transparency of the circles. If not specified, reasonable defaults will be used. #alpha = 0.3

you need short names for your clusters, as there is little space on the plot # but cell types have complicated and long names # So you can provide a table with two columns: 1) short cluster name 2) long version # e.g. EC, endothelial cells # can be a .tsv or .csv file acronymFile = "acronyms.tsv"

the unit of the values in the expression matrix # any string, shown on genome browser and violin y-Axis # typical values are: "read count/UMI", "log of read count/UMI", "TPM", "log of TPM", "CPM", "FPKM", "RPKM" unit = "TPM"

format of the numbers in the matrix. # 'auto' works in 99% of the cases. Otherwise you can use 'int' for integers and 'float' for floating point numbers. # Use 'forceInt' if your matrix contains only integers but in a format like 3.123e10 # or the matrix has only integers expressed like 100.000, 200.000, 300.00, ... matrixType='auto'

— The following options are only used by cbHub — hubName = "100 Genes Sample Hub" # name of hub (optional, default is value of 'shortLabel') ucscDb = "hg38" # UCSC genome ID of the BAM files, required bamDir = "bam" # directory with .bam files, optional. If not present, don't do bam merging #clusterOrder = "clusterOrder.txt" # file with cluster names to order the tracks (default is alphabetical)

CHAPTER 11

Describing datasets

The file `desc.conf` is a key-value file, similar to `cellbrowser.conf`, but it describes the dataset.

A sample file can be created with the command `cbBuild --init`. The tags in the file refer to either HTML files or directly contain the relevant text, URL, accession or in rare cases key/value information.

The most important tags are `title`, `abstract`, `methods`, `unitDesc`, `image`, `pmid`, `paper_url`, `lab` and `submitter`. All tags are described below and they are sorted by content.

These tags contain longer text and can include HTML markup:

- `title`: title of the dataset, often the paper title
- `abstract`: a big picture summary of the dataset, as a string
- `methods`: the methods for the dataset, as a string
- `unitDesc`: a description of the values / the unit in the expression matrix (e.g. 'TPM' or 'log'ed counts')

Instead of long strings with HTML content for `abstract` and `methods`, you can also create the files `abstract.html` and `methods.html`, they will be used instead. Or use the statements `abstractFile` and `methodsFile` to specify other file names. In the HTML, you can use text like `<section>some subtitle</section>` to split the text into sections.

These tags contain a file name:

- `image`: usually a 400px-wide thumbnail of the dimensionality reduction. You can use a command like `convert graphical_abstract.png -sampling-factor 4:2:0 -strip -resize 400 -quality 85 -interlace JPEG -colorspace sRGB thumb.jpg` to create `thumb.jpg`, a version of the image only 400 pixels wide that loads faster.
- `rawMatrixFile`: the file name of the raw unprocessed matrix. Usually a .zip or .gz file. Also see `rawMatrixNote`.

The following tags can contain URLs and optionally, separated with a space, a label for the link. If you do not specify the label, a default label will be used (e.g. 'Biorxiv Preprint'):

- `biorxiv_url`: URL of the pre-print
- `paper_url`: URL to any website with the fulltext

- `other_url`: URL to a website that describes the dataset

The following tags contain accession IDs and will be translated to links:

- `pmid`: Pubmed ID of the publication (CIRM TagsV5)
- `geo_series`: NCBI GEO series ID (CIRM TagsV5)
- `sra`: NCBI SRA accession
- `arrayexpress`: EBI Arrayexpress accession
- `ena_project`: EBI ENA project accession, ENAPxxxx
- `sra_study`: NCBI SRA SRPxxxx accession
- `doi`: DOI of paper fulltext
- `dbgap`: NCBI dbGaP accession, starts with phs
- `bioproject`: NCBI Bioproject accession, PRJNAxxxx. Can be included with or without the PRJNA prefix.

The following tags contain just text:

- `submitter`: name and/or email of submitter
- `lab`: lab and University of submitter
- `submission_date`: ideally in format year-month-day
- `rawMatrixNote`: text to describe the raw matrix, see `rawMatrixFile`
- `version`: version of dataset, a simple number (1,2,3,...) that should be increased each time a major change (usually meta data) was received from the lab

The following tags contain key-value information:

```
- ``custom``: anything about the dataset that does not have an existing tag, e.g. {
↪'taxon_id':'9606'}
- ``urls``: any url you want to show, e.g. { 'Raw data on synapse':'https://www.
↪synapse.org/#!Synapse:syn21560407' }.
- ``algParams``: algorithm parameters, e.g. { 'louvainRes':'0.7' }. This tag is␣
↪generated by cbScanpy.
```

The following tags contain a list of key-value information:: - `supplFiles`: additional files that should be copied and be shown in the 'Download' section, like protocols, Seurat

or Scanpy files, e.g. `supplFiles=[{'file':'seurat3.rds', 'label':'Seurat3 RDS'}]`

The folllowing tags contain only `True` or `False`:

```
- ``hideDownload=True``: do not show the download instructions.
```

CHAPTER 12

Dataset Hierarchies

The Cell Browser allows you to group related datasets into a hierarchy, where datasets are grouped into collections, like files are grouped into directories. When you open a collection, it will show you all of the datasets within it.

This requires your datasets to be arranged in directories on disk. Let's say you have two directories with data files, one in directory `data1` and one in directory `data2`, each with their own cellbrowser.conf files, then these two directories must be both subdirectories of a parent directoryn named e.g. `dataParent`. The names of all the datasets are the names of their directories, not the names specified via the `name` statement in their `cellbrowser.conf` files anymore. Specified names from cellbrowser.conf are ignored when dataset hierarchies are used and are replaced with their directory names.

To enable dataset hierarchies, you only have to add a single line pointing to the top-level parent directory where all of your single-cell data lives. Add a statement like the following to your `~/.cellbrowser.conf`:

```
dataRoot='/celldata/'
```

Then, create a "stub" cellbrowser.conf into this directory, it should only contain a single line like `shortLabel="some description"`. You can describe your collection as discussed under the **Describing datasets** section. Put the `desc.conf` file into the same directory as the `cellbrowser.conf` you just created. Define at least the statements `title` and `description`. They will be shown at the top of your dataset list. This directory can be called the top-level collection.

Arrange your dataset directories under this directory. You can add empty directories, which will become collections, by creating dataset directories in them and put a `cellbrowser.conf` and `desc.conf` into it, e.g. like this:

```
mkdir -p /celldata/organoids
cd /celldata/organoids
echo 'name="organoids"' > cellbrowser.conf
echo 'shortLabel="Brain Organoids"' >> cellbrowser.conf
echo 'tags=["10x"]' >> cellbrowser.conf
```

Now you can run `cbBuild` in each subdirectory of a collection. in the collection. Or you can rebuild in all subdirectories using `cbBuild -r`.

If you view the cell browser now using a web browser, you should see this new collection present. When viewing a dataset in a collection, you can move quickly to any other dataset in the same collection using the "Collection"

dropdown menu in the toolbar.

CHAPTER 13

Loading a dataset

Once you have found a dataset of interest on https://cells.ucsc.edu, it is very easy to load it into your favorite analysis environment. (Let us know if something if we are missing one below.)

First, download the expression matrix and the meta data, usually in a Unix terminal:

```
mkdir adultPancreas
cd adultPancreas
wget https://cells.ucsc.edu/adultPancreas/exprMatrix.tsv.gz
wget https://cells.ucsc.edu/adultPancreas/meta.tsv
```

Replace "quakePancreas" above with the dataset name of interest, it is shown in the URL when you open a dataset after "ds=" or in the download instructions or on the dataset page as the "CellBrowser dataset identifier".

Then open your favorite tool (e.g. RStudio or Jupyter) and follow the instructions below.

13.1 Seurat

Run these commands if you have downloaded the file as above:

```
require(Seurat)
require(data.table)
setwd("adultPancreas")
mat <- fread("zcat < exprMatrix.tsv.gz")
meta <- read.table("meta.tsv", header=T, sep="\t", as.is=T, row.names=1)
genes = mat[,1][[1]]
genes = gsub(".+[|]", "", genes)
mat = data.frame(mat[,-1], row.names=genes)
so <- CreateSeuratObject(counts = mat, project = "adultPancreas", meta.data=meta)
```

Or you can download directly into R, without wget, by replacing the fread commands with these:

```
mat <- fread("curl https://cells.ucsc.edu/adultPancreas/exprMatrix.tsv.gz | zcat")
meta <- data.frame(fread("https://cells.ucsc.edu/adultPancreas/meta.tsv"), row.
↪names=1)
```

(continues on next page)

(continued from previous page)

13.2 Scanpy

To create an anndata object in Scanpy:

```
import scanpy as sc
import pandas as pd
ad = sc.read_text("exprMatrix.tsv.gz")
meta = pd.read_csv("meta.tsv", sep="\t")
ad.var = meta
```

CHAPTER 14

Optional Python modules

There are currently no required Pythom modules for the core Cell Browser script: `cbBuild`. However, to take advantage of some of the advanced features or specialized scripts such as `cbScanpy` or `cbSeurat`, you will need to install some extra packages or tools.

14.1 Custom Colors

In your cellbrowser.conf you can specify a color file, e.g. colors.tsv, with custom colors for your metadata values. The file can be in tsv or csv format and it has two columns, first metadataValue and then colorCode. If this file contains HTML color names instead of color codes, you have to install the module webcolors:

```
pip install webcolors
```

14.2 Image sizes

To get the image sizes, cbBuild uses either the "file" command or the "identify" command (for JPEGs). You may have to install the ImageMagick package to get the identify command.

14.3 Matrices in mtx format

To read expression matrices in .mtx format, you have to install scipy:

```
pip install scipy
```

14.4 `cbScanpy` and `cbSeurat`

`cbScanpy` requires that Scanpy is installed. See the Scanpy documentation for installation instructions.

`cbSeurat` requires that both R and Seurat are installed. See the Seurat website for installation instructions. Note Conda can also be used to install Seurat and R. You can confirm that seurat is installed for R by typing *Rscript* and looking for Seurat in the output.