rnaiutilities

Apr 02, 2019

Contents

A collection of python tools for processing image-based RNAi screens.

CHAPTER 1

Introduction

Welcome to `rnaiutilities`.

`rnaiutilities` provide a set of python modules and commandline scripts that can be used to process, convert, query and analyse imaged-based RNAi-screens.

The packages are designed for the following workflow:

- Download raw `mat` files from an openBIS instance or where ever your data lie. The `mat` files are supposed to be created by `CellProfiler`, i.e. platewise data-sets, where every file describes a single features for single-cells.
- Parse the downloaded data using `rnai-parse`: install the package, and process as described in the package folder. This generates a list of raw `tsv` files or a bundled `h5` file. Until now the parser writes featuresets for *cells*, *perinuclei*, *nuclei*, *expandednuclei*, *bacteria* and *invasomes*.
- Query the meta DB using `rnai-query` and create and combine datasets. For that first meta files generated from the step above are written into a database. Then the DB can be queried against to subset single *genes*, *sirnas*, *pathogens*, etc. and write the *normalized* results.
- To come: `rnai-analyse` for analysing large-scale RNAi screens.

The package is still under development, so if you'd like to contribute, fork us on GitHub.

CHAPTER 2

Installation

Make sure to have `python3` installed. `rnaiutilities` does not support previous versions. The best way to do that is to download anaconda and create a virtual environment.

Download the latest release first and install it using:

```
pip install .
```

If you get errors, I probably forgot some dependency.

CHAPTER 3

Usage

3.1 rnai-parse

`rnai-parse` parses Matlab files of image-based single-cell features from RNAi screens. We assume the data has been generated using `CellProfiler` which creates a single file for each feature that can be measured from a flourescence channel.

Usually from the matlab files features for a single cell are hard to access, since they are distributed over the different files. With `rnai-parse` we first iterate over the single feature files and combine the features into a feature matrix that is easier to work with. The result is a single `tsv` file for every plate where the rows are single-cells and the columns single-cell features.

The following sections describe the usage of `rnai-parse`, its subcommands and the required *CONFIG* file. So far the following subcommands are available:

- `rnai-parse checkdownload` for checking if all files are present correctly,
- `rnai-parse parse` for parsing the data,
- `rnai-parse parsereport` for creating a report of parsed files,
- `rnai-parse featuresets` for creating feature set overlap statistics.

The subcommands are needed to be called consecutively. So you need to parse the files before creating reports and featureset statistics.

3.1.1 Introduction

To use `rnai-parse`, you need to create a *CONFIG* file in `yaml` format with the following content:

Listing 1: Contents of `config.yml` file

```
layout_file: "layout.tsv"
plate_folder: "./"
output_path: "./out"
```

(continues on next page)

(continued from previous page)

```
plate_id_file: "experiment_meta_file.tsv"
plate_regex: '.*\/\w+\-\w[P|U]\-[G|K]\d+(-\w+)*\/.*'
multiprocessing: False
```

You can have a look at an example yaml file here.

layout_file describes the placement of siRNAs and genes on the plates, *plate_folder* points to the collection of matlab files, *output_path* is the target directory where files are written to. *plate_id_file* is a list of ids of plates that are going to be parsed in case only a subset of *plate_folder* should get parsed. *plate_regex* is a pattern which plates you want to use in the *plate_id*file*. *multiprocessing* is a boolean determining whether python uses multiple processes or not.

Check out the data folder in the main repository for some example datasets. The folder contains an example data-set for the pathogen *S. Typhimurium*, the respective `yaml` config file, the meta file that contains the plates to be parsed and the layout file for genes, sirnas, etc.

For all subcommands only the config file is needed as an argument. So if you create the file once, you are settled.

3.1.2 Checking for file availability

As a first step it makes sense to check if all plates from your meta plate file (experiment_meta_file.tsv) exist, i.e. have been downloaded:

```
rnai-parse checkdownload CONFIG
```

This just prints a report to `stderr` if the files exist or not.

3.1.3 Parsing the data

If the files are downloaded as intended, parse them to tsv:

```
rnai-parse parse CONFIG
```

The result of the parsing process should be a set of files for every plate. For example every plate should create `*data.tsv` files and a respective `*meta.tsv` for every data file. Every data file contains the features for a specific channel, like DAPI.

3.1.4 Generating a report

If parsing is complete, you can create a report if all files have been parsed or if some are missing:

```
rnai-parse parsereport CONFIG
```

The report is similar to `rnai-parse checkdownload`, only that this time we check if every file has been parsed correctly and meta files have been created. Output is written to `stderr`.

3.1.5 Computing overlapping feature sets

Finally after having done the parsing and file checking you can create feature overlap statistics between the different screens like this:

```
rnai-parse featuresets CONFIG
```

The script writes the results to `stdout`.

3.2 rnai-query

`rnai-query` builds on the previously **parsed** Matlab files (see *rnai-parse*) and uses them for quickly subsetting the complete dataset of single-cells using an *SQLite*.

For that, you first need to create a database that indexes the plates' meta information. Having the database set up, you can query for different plates and compile different data-sets.

The following sections will explain how `rnai-query` and its subcommands are used. So far the following subcommands are available:

- `rnai-query insert` for inserting meta information to a database,
- `rnai-query query` for querying the database and writing plates to a file,
- `rnai-query compose` for creating of data sets,
- `rnai-query select` for selecting single variables from the database.

The steps have to be taken in succession (or at least insert has to be the first command to be executed), so make sure to read it all.

3.2.1 Inserting meta information

Before being able query the database, we need to insert the parsed meta files. We can to that by calling:

```
rnai-query insert /i/am/a/file/called/tix.db
                  /i/am/a/path/to/parsed/data
```

where `/i/am/a/path/to/parsed/data` points to the folder where the `*meta.tsv` and `*data.tsv` files lie (the result from *rnai-parse*). This creates an *SQLite* database called `tix.db` which we will use for querying the data and creating datasets.

3.2.2 Creating data-sets

Having the database set up, we can query it and create custom single-cell data-sets by filtering on meta information. As a motivating example consider these two scripts:

```
rnai-query compose --sample 10 /i/am/a/file/called/tix.db OUTFILE
rnai-query compose --plate dz05-1e --gene pik3ca
                   /i/am/a/file/called/tix.db
                   OUTFILE
```

The first query would return 10 single cells randomly sampled from each well from all plates and write it to the file *OUTFILE*. The second query would only look at plate *dz05-1e* and gene *pik3ca* and write the single cells that fit the criteria to *OUTFILE*.

The next sections walk you through using `rnai-query compose`.

Command line arguments

Say we would want to filter the database on some critera and only write the single-cell features that fit these conditions. Using `rnai-query compose` you can choose which plates/gene/sirnas/etc. to choose from, by setting the respective command line arguments:

--normalize	The normalization methods to use, e.g. like 'zscore' or a comma-separated string of normalisations such as 'bscore,loess,zscore'. **Defaults to 'zscore'**. If you do **not** want to normalize you need to explicitely set to 'none'.
--from-file	You can provide an optional *tsv* file that has been created using *rnai-query query* such that only on these files will be searched. The filters you provide, like *–study* or *–pathogen*, still need to be given.
--study	The study to query for, e.g. like 'infectx', or a comma-separated string of libraries, such as 'infectx,infectx_published'.
--pathogen	The pathogen to query for, e.g. like 'adeno', or a comma-separated string of pathogens, such as 'adeno, bartonella'.
--library	The library to query for, e.g. like 'd', or a comma-separated string of libraries, such as 'd,q'.
--plate	The plate to query for, e.g. like 'dz03-1k', or a comma-separated string of plates, such as 'dz03-1k,dz04-1k'.
--design	The design to query for, e.g. like 'p'.
--replicate	The replicate to query for, e.g. like '1', or a comma-separated string of replicates, such as '1,4'.
--gene	The gene to query for, e.g. like 'pik3ca', or a comma-separated string of genes, such as 'pik3ca,pik4ca'.
--sirna	The sirna to query for, e.g. like 's12312', or a comma-separated string of sirnas, such as 's12312,s123112'.
--well	The well to query for, e.g. like 'a01', or a comma-separated string of wells, such as 'a01,l05'.
--featureclass	The featureclass to query for, e.g. like 'cells' or a or a comma-separated string of cells, such as 'cells,perinuclei,nuclei'.
--sample	The amount of single cells that are sampled per well,like '100'. If unset defaults to all cells.
--debug	Dont write the files, but only print debug information.
--help	Print a help message.

If any argument is not specified it is internally set to None, the whole database will be searched and no filters applied.

Examples

Here, we show some examples how you can query. In these examples we use a *SQLite* database called `database.db`.

Sample 100 cells from every well for every plate and write **standardized** data to *OUTFILE*.

```
rnai-query compose --sample 100
                   database.db OUTFILE
```

Filter by pathogens *shigella* and *bartonella* and write **standardized** data to *OUTFILE*.

```
rnai-query compose --pathogen shigella,bartonella
                   database.db OUTFILE
```

Filter by pathogens *Shigella* and *Bartonella* and gene *pik3ca* and write standardized data to *OUTFILE*.

```
rnai-query compose --pathogen shigella,bartonella
                   --gene pik3ca
                   --normalize zscore
                   database.db OUTFILE
```

Filter by pathogens *Shigella* and *Bartonella* and gene *pik3ca* and only write debug info.

```
rnai-query compose --pathogen shigella,bartonella
                   --gene pik3ca
                   --debug
                   database.db OUTFILE
```

Filter by gene *nfkb1*, pathogen *Shigella*, study *infectx*, *pooled* designs, sample 1000 cells per well and write *unnormalized* data to *OUTFILE*.

```
rnai-query compose  --gene nfkb1
                    --pathogen shigella
                    --study infectx
                    --design p
                    --sample 1000
                    --normalize none
                    database.db OUTFILE
```

Filter by gene *pik3ca* and *mock*, feature classes *cells* and *perinuclei*, pathogens *Shigella* and *Bartonella*, library *Dharmacon* with a *pooled* siRNA design, sample 100 cells from each well and write **standardized** data to *OUTFILE*.

```
rnai-query compose --featureclass cells,perinuclei
                   --gene pik3ca,mock
                   --library d
                   --design p
                   --pathogen shigella,bartonella
                   --sample 100
                   database.db OUTFILE
```

Filter from a pre-made list of plates and the same filters as before.

```
rnai-query compose --from-file file.tsv
                   --gene pik3ca,mock
                   --library d
                   --design p
                   --pathogen shigella,bartonella
                   --sample 100
                   database.db OUTFILE
```

3.2.3 Querying for plates

If you are only interested in getting the plates the fullfil some criteria and writing them to a file *rnai-query query* does the job.

```
rnai-query query  --gene pik3ca,mock
                  --library d
                  --design p
                  --pathogen shigella,bartonella
```

(continues on next page)

(continued from previous page)

```
                  --sample 100
                  database.db OUTFILE
```

The file you are getting can then be used **input** for *–from-file* for *rnai-query compose*. Sometiems this is required because the queries we want to submit to the data base are so big that it crashes. The arguments are quite the same as above.

3.2.4 Selecting single variables from the database

Sometimes we might want to select single features from the database without writing them to a file, for instance

- if we want to see which genes are available for a pathogen,
- to see which libraries are available for a pathogen,
- to see which plates carry which genes,
- ...

We can use `rnai-query select` for this kind of question. For example, if we are interested in finding which genes are available on plate *dz05-1e*, we would call

```
rnai-query select --plate dz05-1e database.db gene
```

`rnai-query select` takes the same filters as `rnai-query compose`, except *sample*, *normalize* and *debug*, so check section *Command line arguments*.

Examples

Here, we show some examples how you can select variables. In these examples we use a *SQLite* database called `database.db`.

Select which genes are available for pathogens *shigella* and *bartonella*.

```
rnai-query select --pathogen shigella,bartonella
                  database.db gene
```

Select which libraries are available for pathogens *shigella* and *bartonella* and gene *pik3ca*.

```
rnai-query select --pathogen shigella,bartonella
                  --gene pik3ca
                  database.db library
```

Select pathogens for which *pik3ca* and *mock*, feature classes *cells* and *perinuclei*, library *Dharmacon* with a *pooled* siRNA design are available.

```
rnai-query select --featureclass cells,perinuclei
                  --gene pik3ca,mock
                  --library d
                  --design p
                  database.db pathogen
```