Zerone tutorial

Pol Cuscó and Guillaume Filion

May 9, 2016

1 Building instructions

Zerone is available as a Linux command line application and as an R package.

1.1 Downloading

We recommend that you use git to keep Zerone updated. You can clone the repository from Github with the following command on a standard terminal.

```
git clone git@github.com:nanakiksc/zerone
```

Note that this requires that you already have a Github account and that the computer you are working on has an SSH key registered on GitHub. If this is not the case, follow the instructions from `https://help.github.com/articles/generating-ssh-keys/`.

Alternatively, if you prefer not to use git, you can download the source code from `https://github.com/nanakiksc/zerone/archive/master.zip` with the following commands.

```
wget https://github.com/nanakiksc/zerone/archive/master.zip
unzip zerone-master.zip
mv zerone-master zerone
```

This should create a directory named `zerone`.

1.2 Compiling

To build Zerone, execute the following from the `zerone` directory.

```
cd zerone
make
```

This should succeed on most Linux systems because `make` is available by default. If this is not the case, you can obtain it by typing `sudo apt-get install make` on the Ubuntu terminal. Calling `make` should create an executable called `zerone`.

Installing the Zerone R package

To install the Zerone R package, simply run this command from the `zerone` directory.

```
R CMD INSTALL ZeroneRPackage
```

Note that you need to have R installed on your computer. If this is not the case, run the command `sudo apt-get install r-base` on Ubuntu. If applicable, you may also append `sudo` before the command to install the package system-wide.

2 Zerone basics

2.1 Running Zerone

To run Zerone, you have to specify the files that contain the mapped reads of the ChIP-seq experiment you want to discretize. These can be in BED, SAM/BAM and GEM (.map) formats and can be gzipped. You can include as many mock control files and as many experimental replicates as you need, provided there is at least one mock control and one experimental replicate. Just enter mock controls after the `-0` or `--mock` option, and targets after the `-1` or `--chip` option.

For example, you can type in the following commands from the `zerone` directory.

```
./zerone --mock data/mock.sam --chip data/ctcf1.sam,data/ctcf2.sam
```

Note that path expansion will not work when using comma separated file names, so if you want to use path names starting with ∼, you can simply specify the mock and chip options as many times as needed, as shown below.

```
./zerone -0 data/mock.sam -1 data/ctcf1.sam -1 data/ctcf2.sam
```

Enter the option `-h` or `--help` for usage instructions and `--version` to print the version number.

2.2 Window output

Running any of the examples above should produce an output like the following.

```
# QC score: 1.710
# features: 0.472, 10.000, 0.017, 0.404, 0.858
# advice: accept discretization.
chr1    1       300     1   0   0   0    0.29979
chr1    301     600     1   0   0   0    0.16205
chr1    601     900     1   0   0   0    0.09607
chr1    901     1200    1   0   0   0    0.06440
...
chr1    998701  999000  2   1   11  4    0.97890
chr1    999001  999300  2   2   72  34   1.00000
chr1    999301  999600  2   2   128 69   1.00000
chr1    999601  999900  2   0   10  5    0.99894
```

> The first three lines contain the result of the **quality control**. It consists of a quality score followed by the numeric values of 5 features, and an advice to either **accept or reject** the discretization. The recommendation is to accept if the score is positive and reject if it is negative. If the score is higher than 1 (or lower than -1), the advice is considered extremely reliable.

The rest of the lines contain the discretization proper. The first three columns specify the chromosome, start and end positions of each window.

The fourth column represents the **enrichment**. Zerone classifies each window into one of three possible states. States 0 and 1 represent two types of background signal. State **2 represents an enriched window**.

The fifth column contains the read counts of all the control profiles summed together per window. The following columns show the number of reads per window in the ChIP-seq files, in the order they were provided. The final column is the estimated probability that the window is a target (it is a confidence score for the call).

2.3 List output

With the `-l` or `--list-output` option, Zerone produces an alternative output in which only the targets are shown after merging consecutive windows. For instance, when running the following command

```
./zerone -l -0 data/mock.sam -1 data/ctcf1.sam,data/ctcf2.sam
```

you should obtain the output shown below.

```
# QC score: 1.710
# features: 0.472, 10.000, 0.017, 0.404, 0.858
# advice: accept discretization.
chr1  237601  238200  1.00000
chr1  521401  522000  0.99969
chr1  567301  567900  0.91610
...
chr1  975901  976500  1.00000
chr1  990001  990600  0.99054
chr1  994201  995400  1.00000
chr1  998701  999900  1.00000
```

The first three columns are the same as in window output *i.e.* chromosome or sequence name, start and end. The last column is the confidence score of the called target. It is the *highest* confidence of the windows merged in the same target region.

The Zerone R package

Once you have installed the Zerone R package, you can load it in your R session with the command

```
library(zerone)
```

The package contains a single function called `zerone()`. You can display the help page with the command `?zerone`. The object `ZeroneExampleData` is an example of `data.frame` properly formatted for `zerone()`. The command below runs `zerone()` on this data and returns the Viterbi path (inferred sequence of states).

```
path <- zerone(ZeroneExampleData)
```

If you want to return the parameters fitted by `zerone()`, you can do it as shown below.

```
listinfo <- zerone(ZeroneExampleData, returnall=TRUE)
```

The list object `listinfo` then contains all the associated parameters and extra information.

3 Troubleshooting

In case Zerone crashes, recompile it in debug mode. To do so, run the following commands from the `zerone` directory.

```
make clean
make debug
```

Then repeat the call that triggered the crash and contact guillaume.filion@gmail.com attaching the debug information.

If you are using Linux, you can also run the unit tests to check that the code behaves as expected on your machine. For this, run the following commands in the Zerone directory.

```
make test -C src/test
```

If everything works well, you should see the following output

```
*******************************
*      Zerone unit tests       *
*******************************
hmm/fwdb                       OK
hmm/fwdb (NAs)                 OK
hmm/fwdb (underflow)           OK
hmm/block_fwdb                 OK
hmm/block_fwdb (NAs)           OK
...
zerone/zinm_prob               OK
zerone/bw_zinm                 OK
zerone/update_trans            OK
zerone/eval_bw_f               OK
*******************************
```

The tests will not run on every machine (in particular they will not run on Mac), but this does not mean that something is broken. If `Zerone unit tests` in the header does not appear, it means that the tests cannot be compiled on your machine, but this does not say anything about Zerone. If one of the tests fails, `OK` will be replaced by `FAIL` and the test harness will give some more information about what went wrong. You can contact guillaume.filion@gmail.com attaching the test results so that we can help you troubleshoot.

You can also share your issues directly on the Github repository at `https://github.com/nanakiksc/zerone/issues`. Be sure to give enough information so that we can reproduce the issue.

Finally, we provide a Docker image with a running implementation of Zerone. This can serve as an environment to run Zerone if it is not available on your machine, or as an example to see how to configure your own machine. The Docker image is available at `https://hub.docker.com/r/nanakiksc/zerone/`.