pySCENIC Documentation

Release latest

Bram Van de Sande

Jul 17, 2020

Contents

1 Installation **1**
- 1.1 Stable . . . 1
- 1.2 Development . . . 1
- 1.3 Containers . . . 1

2 Auxiliary datasets **3**

3 Command Line Interface **5**

4 Docker and Singularity Images **7**
- 4.1 Docker . . . 7
- 4.2 Singularity . . . 8
- 4.3 Using the Docker or Singularity images with Jupyter notebook . . . 8

5 Nextflow **9**

6 Tutorial **11**
- 6.1 Case studies . . . 11
- 6.2 Zeisel et al. dataset . . . 11
- 6.3 References . . . 14

7 Frequently Asked Questions **15**
- 7.1 I am having problems with Dask . . . 15
- 7.2 How can I prioritize the target genes within a particular regulon? . . . 15
- 7.3 Can I create my own ranking databases? . . . 16
- 7.4 Can I draw the distribution of AUC values for a regulon across cells? . . . 16

8 Release Notes **19**
- 8.1 0.10.2 | 2020-06-05 . . . 19
- 8.2 0.10.1 | 2020-05-17 . . . 19
- 8.3 0.10.0 | 2020-02-27 . . . 19

9 pySCENIC **21**
- 9.1 News and releases . . . 21
- 9.2 Overview . . . 22
- 9.3 Additional resources . . . 22
- 9.4 Acknowledgments . . . 23
- 9.5 References . . . 23

CHAPTER 1

Installation

1.1 Stable

The latest stable release of the **package** itself can be installed via

```
pip install pyscenic
```

Caution: pySCENIC needs a python 3.6 or greater interpreter.

1.2 Development

You can also install the bleeding edge (i.e. less stable) version of the package directly from the source:

```
git clone https://github.com/aertslab/pySCENIC.git
cd pySCENIC/
pip install .
```

1.3 Containers

pySCENIC containers are also available for download and immediate use. In this case, no compiling or installation is required, provided either Docker or Singularity software is installed on the user's system. Images are available from Docker Hub. Usage of the containers is shown below (*Docker and Singularity Images*). To pull the docker images, for example:

```
docker pull aertslab/pyscenic:0.10.0
```

CHAPTER 2

Auxiliary datasets

To successfully use this pipeline you also need **auxilliary datasets**:

1. *Databases ranking the whole genome* of your species of interest based on regulatory features (i.e. transcription factors). Ranking databases are typically stored in the feather format and can be downloaded from cisTargetDBs.
2. *Motif annotation* database providing the missing link between an enriched motif and the transcription factor that binds this motif. This pipeline needs a TSV text file where every line represents a particular annotation.

Annotations	Species
HGNC annotations	Homo sapiens
MGI annotations	Mus musculus
Flybase annotations	Drosophila melanogaster

Caution: These ranking databases are 1.1 Gb each so downloading them might take a while. An annotations file is typically 100Mb in size.

A list of transcription factors is required for the network inference step (GENIE3/GRNBoost2). These lists can be downloaded from resources section on GitHub.

CHAPTER 3

Command Line Interface

A command line version of the tool is included. This tool is available after proper installation of the package via `pip`.

```
{ ~ }  » pyscenic                                                    ~
usage: pyscenic [-h] {grn,ctx,aucell} ...

Single-CEll regulatory Network Inference and Clustering (0.10.0)

positional arguments:
  {grn,ctx,aucell}  sub-command help
    grn             Derive co-expression modules from expression matrix.
    ctx             Find enriched motifs for a gene signature and optionally
                    prune targets from this signature based on cis-regulatory
                    cues.
    aucell          Quantify activity of gene signatures across single cells.

optional arguments:
  -h, --help        show this help message and exit

Arguments can be read from file using a @args.txt construct. For more
information on loom file format see http://loompy.org . For more information
on gmt file format see https://software.broadinstitute.org/cancer/software/gse
a/wiki/index.php/Data_formats .
```

CHAPTER 4

Docker and Singularity Images

pySCENIC is available to use with both Docker and Singularity, and tool usage from a container is similar to that of the command line interface. Note that the feather databases, transcription factors, and motif annotation databases need to be accessible to the container via a mounted volume. In the below examples, a single volume mount is used for simplicity, which will contains the input, output, and databases files.

For additional usage examples, see the documentation associated with the SCENIC protocol Nextflow implementation.

4.1 Docker

Docker images are available from Docker Hub, and can be obtained by running `docker pull aertslab/pyscenic:[version]`, with the version tag as the latest release.

To run pySCENIC using Docker, use the following three steps. A mount point (or more than one) needs to be specified, which contains the input data and necessary resources).

```
docker run -it --rm \
    -v /data:/data \
    aertslab/pyscenic:0.10.0 pyscenic grn \
        --num_workers 6 \
        -o /data/expr_mat.adjacencies.tsv \
        /data/expr_mat.tsv \
        /data/allTFs_hg38.txt

docker run -it --rm \
    -v /data:/data \
    aertslab/pyscenic:0.10.0 pyscenic ctx \
        /data/expr_mat.adjacencies.tsv \
        /data/hg19-tss-centered-5kb-7species.mc9nr.feather \
        /data/hg19-tss-centered-10kb-7species.mc9nr.feather \
        --annotations_fname /data/motifs-v9-nr.hgnc-m0.001-o0.0.tbl \
        --expression_mtx_fname /data/expr_mat.tsv \
        --mode "dask_multiprocessing" \
```

(continues on next page)

(continued from previous page)

```
        --output /data/regulons.csv \
        --num_workers 6

docker run -it --rm \
    -v /data:/data \
    aertslab/pyscenic:0.10.0 pyscenic aucell \
        /data/expr_mat.tsv \
        /data/regulons.csv \
        -o /data/auc_mtx.csv \
        --num_workers 6
```

4.2 Singularity

As of release `0.9.19`, pySCENIC Singularity images are no longer being built on Singularity Hub, however images can easily be built using Docker Hub as a source:

```
singularity build aertslab-pyscenic-0.10.0.sif docker://aertslab/pyscenic:0.10.0
```

To run pySCENIC with Singularity, the usage is very similar to that of Docker. Note that in Singularity 3.0+, the mount points are automatically overlaid, but bind points can be specified similarly to Docker with `--bind`/`-B`. The first step (GRN inference) is shown as an example:

```
singularity run aertslab-pyscenic-0.10.0.sif \
    pyscenic grn \
        --num_workers 6 \
        -o expr_mat.adjacencies.tsv \
        expr_mat.tsv \
        allTFs_hg38.txt
```

4.3 Using the Docker or Singularity images with Jupyter notebook

As of version 0.9.7, the pySCENIC containers have the `ipykernel` package installed, and can also be used interactively in a notebook. This can be achieved using a kernel command similar to the following (for singularity). Note that in this case, a bind needs to be specified.

```
singularity exec -B /data:/data aertslab-pyscenic-0.10.0.sif ipython kernel -f
↪{connection_file}
```

CHAPTER 5

Nextflow

There are two Nextflow implementations available:

- SCENICprotocol: A Nextflow DSL1 implementation.
- VSNPipelines: A Nextflow DSL2 implementation.

CHAPTER 6

Tutorial

There are a number of tutorials available to demonstrate how to use pySCENIC. Some of these are in the form of Jupyter notebooks in the pySCENIC notebooks directory, and contain example analyses.

6.1 Case studies

These case studies are associated with the SCENICprotocol and the resulting loom files can be viewed in the SCope viewer here.

6.1.1 PBMC 10k dataset (10x Genomics)

Full SCENIC analysis, plus filtering, clustering, visualization, and SCope-ready loom file creation:

- Jupyter notebook | HTML render

Extended analysis post-SCENIC:

- Jupyter notebook | HTML render

6.1.2 Cancer data sets

- Jupyter notebook | HTML render

6.2 Zeisel et al. dataset

For this tutorial 3,005 single cell transcriptomes taken from the mouse brain (somatosensory cortex and hippocampal regions) are used as an example[4]. The analysis is done in a Jupyter notebook.

[4] Zeisel, A. et al. Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq. Science 347, 1138–1142 (2015).

This dataset is also included in the case studies of the SCENIC protocol, and the analysis notebook can be found therein (Jupyter notebook | HTML render).

Caution: If you run this from a python script instead of a Jupyter notebook, please enclose the code in a `if __name__ == '__main__':` construct.

First we import the necessary modules and declare some constants:

```
import os
import glob
import pickle
import pandas as pd
import numpy as np

from dask.diagnostics import ProgressBar

from arboreto.utils import load_tf_names
from arboreto.algo import grnboost2

from pyscenic.rnkdb import FeatherRankingDatabase as RankingDatabase
from pyscenic.utils import modules_from_adjacencies, load_motifs
from pyscenic.prune import prune2df, df2regulons
from pyscenic.aucell import aucell

import seaborn as sns

DATA_FOLDER="~/tmp"
RESOURCES_FOLDER="~/resources"
DATABASE_FOLDER = "~/databases/"
SCHEDULER="123.122.8.24:8786"
DATABASES_GLOB = os.path.join(DATABASE_FOLDER, "mm9-*.mc9nr.feather")
MOTIF_ANNOTATIONS_FNAME = os.path.join(RESOURCES_FOLDER, "motifs-v9-nr.mgi-m0.001-o0.
↪0.tbl")
MM_TFS_FNAME = os.path.join(RESOURCES_FOLDER, 'mm_tfs.txt')
SC_EXP_FNAME = os.path.join(RESOURCES_FOLDER, "GSE60361_C1-3005-Expression.txt")
REGULONS_FNAME = os.path.join(DATA_FOLDER, "regulons.p")
MOTIFS_FNAME = os.path.join(DATA_FOLDER, "motifs.csv")
```

The scRNA-Seq data is downloaded from GEO: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE60361 and loaded into memory:

```
ex_matrix = pd.read_csv(SC_EXP_FNAME, sep='\t', header=0, index_col=0).T
ex_matrix.shape
```

```
(3005, 19970)
```

and the list of Transcription Factors (TF) for *Mus musculus* are read from file. The list of known TFs for Mm was prepared from TFCat (cf. notebooks section).

```
tf_names = load_tf_names(MM_TFS_FNAME)
```

Finally the ranking databases are loaded:

```
db_fnames = glob.glob(DATABASES_GLOB)
def name(fname):
```

(continues on next page)

(continued from previous page)

```
    return os.path.splitext(os.path.basename(fname))[0]
dbs = [RankingDatabase(fname=fname, name=name(fname)) for fname in db_fnames]
dbs
```

```
[FeatherRankingDatabase(name="mm9-tss-centered-10kb-10species.mc9nr"),
 FeatherRankingDatabase(name="mm9-500bp-upstream-7species.mc9nr"),
 FeatherRankingDatabase(name="mm9-500bp-upstream-10species.mc9nr"),
 FeatherRankingDatabase(name="mm9-tss-centered-5kb-10species.mc9nr"),
 FeatherRankingDatabase(name="mm9-tss-centered-10kb-7species.mc9nr"),
 FeatherRankingDatabase(name="mm9-tss-centered-5kb-7species.mc9nr")]
```

In the initial phase of the pySCENIC pipeline the single cell expression profiles are used to infer co-expression modules from.

The arboreto package is used for this phase of the pipeline. For this notebook only a sample of 1,000 cells is used for the co-expression module inference is used.

```
adjacencies = grnboost2(ex_matrix, tf_names=tf_names, verbose=True)
```

Regulons are derived from adjacencies based on three methods.

The first method to create the TF-modules is to select the best targets for each transcription factor:

1. Targets with importance > the 50th percentile.
2. Targets with importance > the 75th percentile
3. Targets with importance > the 90th percentile.

The second method is to select the top targets for a given TF:

1. Top 50 targets (targets with highest weight)

The alternative way to create the TF-modules is to select the best regulators for each gene (this is actually how GENIE3 internally works). Then, these targets can be assigned back to each TF to form the TF-modules. In this way we will create three more gene-sets:

1. Targets for which the TF is within its top 5 regulators
2. Targets for which the TF is within its top 10 regulators
3. Targets for which the TF is within its top 50 regulators

A distinction is made between modules which contain targets that are being activated and genes that are being repressed. Relationship between TF and its target, i.e. activator or repressor, is derived using the original expression profiles. The Pearson product-moment correlation coefficient is used to derive this information.

In addition, the transcription factor is added to the module and modules that have less than 20 genes are removed.

```
modules = list(modules_from_adjacencies(adjacencies, ex_matrix))
```

```
# Calculate a list of enriched motifs and the corresponding target genes for all␣
↪modules.
with ProgressBar():
    df = prune2df(dbs, modules, MOTIF_ANNOTATIONS_FNAME)

# Create regulons from this table of enriched motifs.
regulons = df2regulons(df)

# Save the enriched motifs and the discovered regulons to disk.
```

(continues on next page)

(continued from previous page)

```
df.to_csv(MOTIFS_FNAME)
with open(REGULONS_FNAME, "wb") as f:
    pickle.dump(regulons, f)
```

Clusters can be leveraged in the following way:

```
# The clusters can be leveraged via the dask framework:
df = prune2df(dbs, modules, MOTIF_ANNOTATIONS_FNAME, client_or_address=SCHEDULER)
```

Caution: The nodes of the clusters need to have access to a shared network drive on which the ranking databases are stored.

Reloading the enriched motifs and regulons from file should be done as follows:

```
df = load_motifs(MOTIFS_FNAME)
with open(REGULONS_FNAME, "rb") as f:
    regulons = pickle.load(f)
```

We characterize the different cells in a single-cell transcriptomics experiment via the enrichment of the previously discovered regulons. Enrichment of a regulon is measured as the Area Under the recovery Curve (AUC) of the genes that define this regulon.

```
auc_mtx = aucell(ex_matrix, regulons, num_workers=4)
sns.clustermap(auc_mtx, figsize=(8,8))
```

6.3 References

CHAPTER 7

Frequently Asked Questions

7.1 I am having problems with Dask

The Arboreto package `v0.1.5`, and some steps of the cisTarget step within pySCENIC, seem to depend on an older version of Dask/Distributed. Using a more recent version of Dask/Distributed can result in some cryptic errors. It is recommended to use the older version of Dask and Distributed for stability here:

```
pip install dask==1.0.0 distributed'>=1.21.6,<2.0.0'
```

But in many cases this still results in issues with the GRN step. An alternative is to use the multiprocessing implementation of Arboreto recently included in pySCENIC (arboreto_with_multiprocessing.py). This script uses the Arboreto and pySCENIC codebase to run GRNBoost2 (or GENIE3) without Dask. The eliminates the possibility of running the GRN step across multiple nodes, but brings provides additional stability. The run time is generally equivalent to the Dask implementation using the same number of workers.

The basic usage is:

```
arboreto_with_multiprocessing.py \
    expr_mat.loom \
    allTFs_hg38.txt \
    --method grnboost2 \
    --output adj.tsv \
    --num_workers 20 \
    --seed 777
```

The algorithm can be selected using the `--method` option (`genie3` or `grnboost2`). Possible input formats for the expression data are the same as for the pySCENIC CLI: loom, and csv.

7.2 How can I prioritize the target genes within a particular regulon?

It can be useful to have a better idea of which particular target genes within a regulon are more important, especially in the cases of regulons with many genes. There are multiple possibilities to address this.

1. **iRegulon analysis.** One possibility is to take the pre-refinement modules for the regulon of interest, and export them for analysis in iRegulon. There are six unrefined modules created from the GRN output for each TF of interest, which are generated using multiple approaches. The use of iRegulon for the pruning/refinement allows for more control over the pruning of these modules and possibly a better idea of which genes are more important. See the last section, *Further exploration of modules directly from the network inference output*, in this notebook for an example of how to get started. A full tutorial on module refinement with iRegulon can be found here.
2. **pySCENIC multi-runs.** Another approach is to run the whole pySCENIC procedure multiple times (~10-100x). Because of the stochastic nature of the GRN step in particular, a slightly varying result is produced for each run, both in terms of the overall regulons found, as well as the target genes for a particular regulon. The target genes (and regulons themselves) can then be scored by the number of times it occurs across all runs, and considered as 'high confidence' if they occur in >80% of runs, for example. This has the potential to be very computationally intensive for a whole dataset, but if only a few regulons are of interest, pySCENIC can be run using only these (by limiting the TFs included in the TFs file that goes into the GRN step). The multi-runs capability is implemented in the SCENIC section of our single cell Nextflow pipeline. The entrypoint scenic_multiruns provides an automated way to run this procedure.

7.3 Can I create my own ranking databases?

Yes you can. The code snippet below shows you how to create your own databases:

```
from pyscenic.rnkdb import DataFrameRankingDatabase as RankingDatabase
import numpy as np
import pandas as pd

# Every model in a database is represented by a whole genome ranking. The rankings of␣
↪the genes must be 0-based.
df = pd.DataFrame(
        data=[[0, 1],
              [1, 0]],
        index=['Model1', 'Model2'],
        columns=['Symbol1', 'Symbol2'],
        dtype=np.int32)
RankingDatabase(df, 'custom').save('custom.db')
```

7.4 Can I draw the distribution of AUC values for a regulon across cells?

```
import pandas as pd
import matplotlib.pyplot as plt


def plot_binarization(auc_mtx: pd.DataFrame, regulon_name: str, threshold: float,␣
↪bins: int=200, ax=None) -> None:
    """
    Plot the "binarization" process for the given regulon.

    :param auc_mtx: The dataframe with the AUC values for all cells and regulons (n_
↪cells x n_regulons).
    :param regulon_name: The name of the regulon.
    :param bins: The number of bins to use in the AUC histogram.
    :param threshold: The threshold to use for binarization.
```

(continues on next page)

(continued from previous page)

```
    """
    if ax is None:
        ax=plt.gca()
    auc_mtx[regulon_name].hist(bins=bins,ax=ax)

    ylim = ax.get_ylim()
    ax.plot([threshold]*2, ylim, 'r:')
    ax.set_ylim(ylim)
    ax.set_xlabel('AUC')
    ax.set_ylabel('#')
    ax.set_title(regulon_name)
```

CHAPTER 8

Release Notes

8.1 0.10.2 | 2020-06-05

- Bugfix for CLI grn step

8.2 0.10.1 | 2020-05-17

- CLI: file compression (optionally) enabled for intermediate files for the major steps: grn (adjacencies matrix), ctx (regulons), and aucell (auc matrix). Compression is used when the file name argument has a .gz ending.

8.3 0.10.0 | 2020-02-27

- Added a helper script arboreto_with_multiprocessing.py that runs the Arboreto GRN algorithms (GRNBoost2, GENIE3) without Dask for compatibility.
- Ability to set a fixed seed in both the AUCell step and in the calculation of regulon thresholds (CLI parameter `--seed`; aucell function parameter `seed`).
- (since 0.9.18) In the modules_from_adjacencies function, the default value of `rho_mask_dropouts` is changed to False. This now matches the behavior of the R version of SCENIC. The cli version has an additional option to turn dropout masking back on (`--mask_dropouts`).

CHAPTER 9

pySCENIC

pySCENIC is a lightning-fast python implementation of the SCENIC pipeline (Single-Cell rEgulatory Network Inference and Clustering) which enables biologists to infer transcription factors, gene regulatory networks and cell types from single-cell RNA-seq data.

The pioneering work was done in R and results were published in Nature Methods[1]. A new and comprehensive description of this Python implementation of the SCENIC pipeline is available in Nature Protocols[5] (see here).

pySCENIC can be run on a single desktop machine but easily scales to multi-core clusters to analyze thousands of cells in no time. The latter is achieved via the dask framework for distributed computing[2].

Full documentation is available on Read the Docs

9.1 News and releases

9.1.1 0.10.2 | 2020-06-05

- Bugfix for CLI grn step

9.1.2 0.10.1 | 2020-05-17

- CLI: file compression (optionally) enabled for intermediate files for the major steps: grn (adjacencies matrix), ctx (regulons), and aucell (auc matrix). Compression is used when the file name argument has a .gz ending.

[1] Aibar, S. et al. SCENIC: single-cell regulatory network inference and clustering. Nat Meth 14, 1083–1086 (2017).

[5] Van de Sande B., Flerin C., et al. A scalable SCENIC workflow for single-cell gene regulatory network analysis. Nat Protoc. June 2020:1-30. doi:10.1038/s41596-020-0336-2

[2] Rocklin, M. Dask: parallel computation with blocked algorithms and task scheduling. conference.scipy.org

9.1.3 0.10.0 | 2020-02-27

- Added a helper script arboreto_with_multiprocessing.py that runs the Arboreto GRN algorithms (GRNBoost2, GENIE3) without Dask for compatibility.
- Ability to set a fixed seed in both the AUCell step and in the calculation of regulon thresholds (CLI parameter `--seed`; aucell function parameter `seed`).
- (since 0.9.18) In the modules_from_adjacencies function, the default value of `rho_mask_dropouts` is changed to False. This now matches the behavior of the R version of SCENIC. The cli version has an additional option to turn dropout masking back on (`--mask_dropouts`).

See also the extended Release Notes.

9.2 Overview

The pipeline has three steps:

1. First transcription factors (TFs) and their target genes, together defining a regulon, are derived using gene inference methods which solely rely on correlations between expression of genes across cells. The arboreto package is used for this step.
2. These regulons are refined by pruning targets that do not have an enrichment for a corresponding motif of the TF effectively separating direct from indirect targets based on the presence of cis-regulatory footprints.
3. Finally, the original cells are differentiated and clustered on the activity of these discovered regulons.

The most impactful speed improvement is introduced by the arboreto package in step 1. This package provides an alternative to GENIE3[3] called GRNBoost2. This package can be controlled from within pySCENIC.

All the functionality of the original R implementation is available and in addition:

1. You can leverage multi-core and multi-node clusters using dask and its distributed scheduler.
2. We implemented a version of the recovery of input genes that takes into account weights associated with these genes.
3. Regulons, i.e. the regulatory network that connects a TF with its target genes, with targets that are repressed are now also derived and used for cell enrichment analysis.

9.3 Additional resources

For more information, please visit LCB, or SCENIC (R version). The CLI to pySCENIC has also been streamlined into a pipeline that can be run with a single command, using the Nextflow workflow manager. There are two Nextflow implementations available:

- SCENICprotocol: A Nextflow DSL1 implementation of pySCENIC alongside a basic "best practices" expression analysis. Includes details on pySCENIC installation, usage, and downstream analysis, along with detailed tutorials.
- VSNPipelines: A Nextflow DSL2 implementation of pySCENIC with a comprehensive and customizable pipeline for expression analysis. Includes additional pySCENIC features (multi-runs, integrated motif- and track-based regulon pruning, loom file generation).

[3] Huynh-Thu, V. A. et al. Inferring regulatory networks from expression data using tree-based methods. PLoS ONE 5, (2010).

9.4 Acknowledgments

We are grateful to all providers of TF-annotated position weight matrices, in particular Martha Bulyk (UNIPROBE), Wyeth Wasserman and Albin Sandelin (JASPAR), BioBase (TRANSFAC), Scot Wolfe and Michael Brodsky (FlyFactorSurvey) and Timothy Hughes (cisBP).

9.5 References