## Contents

1 Features 3

2 Installation 5

3 Tutorial 7
   3.1 Preliminary work 8
   3.2 Phase I: Inference of co-expression modules 8
   3.3 Phase II: Prune modules for targets with cis regulatory footprints (aka RcisTarget) 9
   3.4 Phase III: Cellular regulon enrichment matrix (aka AUCell) 10

4 Command Line Interface 11

5 Docker and Singularity Images 13
   5.1 Docker 13
   5.2 Singularity 14
   5.3 Using the Docker or Singularity images with Jupyter notebook 14
   5.4 Running pySCENIC with Nextflow 14

6 Frequently Asked Questions 15
   6.1 Can I create my own ranking databases? 15
   6.2 Can I draw the distribution of AUC values for a regulon across cells? 15

7 Website 17

8 License 19

9 Acknowledgments 21

10 References 23
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\(^1\).

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\(^2\).

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.

Note: 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.

---

**Quick Start**

- Installation
- Tutorial
- Command Line Interface
- Docker and Singularity Images
- Frequently Asked Questions
- See notebooks
- Report an issue
- Releases at PyPI

---


\(^2\) Rocklin, M. Dask: parallel computation with blocked algorithms and task scheduling. conference.scipy.org

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.
CHAPTER 2

Installation

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.

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 .
```

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 both Docker Hub and Singularity Hub. Usage of the containers is shown below (*Docker and Singularity Images*).

To successfully use this pipeline you also need **auxiliary 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.

<table>
<thead>
<tr>
<th>Annotations</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>HGNC annotations</td>
<td>Homo sapiens</td>
</tr>
<tr>
<td>MGI annotations</td>
<td>Mus musculus</td>
</tr>
<tr>
<td>Flybase annotations</td>
<td>Drosophila melanogaster</td>
</tr>
</tbody>
</table>

**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 https://github.com/aertslab/pySCENIC/tree/master/resources.
For this tutorial 3,005 single cell transcriptomes taken from the mouse brain (somatosensory cortex and hippocampal regions) are used as an example. The analysis is done in a Jupyter notebook.

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:

```python
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")

---

pySCENIC Documentation, Release 0.6.4

3.1 Preliminary work

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).

```
with open(MM_TFS_FNAME) as f:
    tf_names = load_tf_names(MM_TFS_FNAME)
```

Finally the ranking databases are loaded:

```
def name(fname):
    return os.path.splitext(os.path.basename(fname))[0]

dbs = [RankingDatabase(fname=fname, name=name(fname)) for fname in db_fnames]
```

3.2 Phase I: Inference of co-expression modules

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

3.2.1 Run GENIE3 or GRNBoost from arboreto to infer co-expression modules

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)
```
3.2.2 Derive potential regulons from these co-expression modules

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.

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

3.3 Phase II: Prune modules for targets with cis regulatory footprints (aka RcisTarget)

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

Clusters can be leveraged in the following way:

```python
# 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:

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

### 3.4 Phase III: Cellular regulon enrichment matrix (aka AUCell)

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.

```python
auc_mtx = aucell(ex_matrix, regulons, num_workers=4)
sns.clustermap(auc_mtx, figsize=(8,8))
```
A command line version of the tool is included. This tool is available after proper installation of the package via `pip`.

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

Single-CELL regulatory Network Inference and Clustering

positional arguments:
(  grnboost,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  Find enrichment of regulons across single cells.

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

Arguments can be read from file using a @args.txt construct.
```
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.

### 5.1 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).

```bash
docker run
   -v /path/to/data:/scenicdata \
   aertslab/pyscenic:[version] pyscenic grn \ 
      --num_workers 6 \ 
      -o /scenicdata/expr_mat.adjacencies.tsv \ 
      /scenicdata/expr_mat.tsv \ 
      /scenicdata/allTFs_hg38.txt

docker run
   -v /path/to/data:/scenicdata \
   aertslab/pyscenic:[version] pyscenic ctx \ 
      /scenicdata/expr_mat.adjacencies.tsv \ 
      /scenicdata/hg19-500bp-upstream-7species.mc9nr.feather \ 
      /scenicdata/hg19-tss-centered-5kb-7species.mc9nr.feather \ 
      /scenicdata/hg19-tss-centered-10kb-7species.mc9nr.feather \ 
      --annotations_fname /scenicdata/motifs-v9-nr.hgnc-m0.001-o0.0.tbl \ 
      --expression_mtx_fname /scenicdata/expr_mat.tsv \ 
      --mode "dask_multiprocessing" \ 
      --output /scenicdata/regulons.csv
```

(continues on next page)
5.2 Singularity

Singularity images are available from Singularity Hub and can be obtained by running `singularity pull shub://aertslab/pySCENIC:0.9.7` with the proper version tag.

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 exec pySCENIC_0.9.7.sif
    pyscenic grn
        --num_workers 6
        -o expr_mat.adjacencies.tsv
        expr_mat.tsv
        allTFs_hg38.txt
```

5.3 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 pySCENIC_0.9.7.sif ipython kernel -f {connection_file}
```

5.4 Running pySCENIC with Nextflow

The CLI to pySCENIC has also been streamlined into a pipeline that can be run with a single command, using the Nextflow workflow manager. For details on this usage, see the `scenic-nf` repository.
6.1 Can I create my own ranking databases?

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

```python
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')
```

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

```python
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.
    """
```
def plot_auc_histogram(auc_mtx, regulon_name, bins=10, threshold=0.5, ax=None):
    """Plot AUC histogram for a 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.
    """
    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)
    ax.set_yticklabels([])
Chapter 7

Website

For more information, please visit LCB and SCENIC.
License

GNU General Public License v3
CHAPTER 9

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).