MAGIC Documentation

Release 1.0.0

Scott Gigante and Daniel Dager, Krishnaswamy Lab, Yale Univers

Jun 30, 2018

Contents

MAGIC is a tool that shares information across similar cells, via data diffusion, to denoise the cell count matrix and fill in missing transcripts. To see how MAGIC can be applied to single-cell RNA-seq, elucidating the epithelial-to-mesenchymal transition, read our publication in Cell.

David van Dijk, et al. Recovering Gene Interactions from Single-Cell Data Using Data Diffusion. 2018. Cell.

CHAPTER 1

Installation

1.1 Python installation

1.1.1 Installation with *pip*

The Python version of MAGIC can be installed using:

```
pip install --user git+git://github.com/KrishnaswamyLab/MAGIC.git#subdirectory=python
```

1.1.2 Installation from source

The Python version of MAGIC can be installed from GitHub by running the following from a terminal:

```
git clone --recursive git://github.com/KrishnaswamyLab/MAGIC.git
cd MAGIC/python
python setup.py install --user
```

1.2 MATLAB installation

1. The MATLAB version of MAGIC can be accessed using:

   ```
   git clone git://github.com/KrishnaswamyLab/MAGIC.git
   cd MAGIC/Matlab
   ```

2. Add the MAGIC/Matlab directory to your MATLAB path and run any of our *run* or *test* scripts to get a feel for MAGIC.

1.3 R installation

In order to use MAGIC in R, you must also install the Python package.

1.3.1 Installation with *devtools* and *pip*

Install *Rmagic* from CRAN by running the following code in R:

```
if (!require(devtools)) install.packages("devtools")
library(devtools)
install_github("KrishnaswamyLab/magic/Rmagic")
```

Install *magic* in Python by running the following code from a terminal:

```
pip install --user git+git://github.com/KrishnaswamyLab/MAGIC.git#subdirectory=python
```

If *python* or *pip* are not installed, you will need to install them. We recommend Miniconda3 to install Python and *pip* together, or otherwise you can install *pip* from https://pip.pypa.io/en/stable/installing/.

1.3.2 Installation from source

The latest source version of MAGIC can be accessed by running the following in a terminal:

```
git clone https://github.com/KrishnaswamyLab/MAGIC.git
cd MAGIC/Rmagic
R CMD INSTALL
cd ../python
python setup.py install --user
```

If the *Rmagic* folder is empty, you have may forgotten to use the *–recursive* option for *git clone*. You can rectify this by running the following in a terminal:

```
cd MAGIC
git submodule init
git submodule update
cd Rmagic
R CMD INSTALL
cd ../python
python setup.py install --user
```

CHAPTER 2

Tutorial

To run MAGIC on your dataset, create a MAGIC operator and run *fit_transform*. Here we show an example with an artificial test dataset located in the MAGIC repository:

```
import magic
import matplotlib.pyplot as plt
import pandas as pd
X = pd.read_csv("MAGIC/data/test_data.csv")
magic_operator = magic.MAGIC()
X_magic = magic_operator.fit_transform(X, genes=['VIM', 'CDH1', 'ZEB1'])
plt.scatter(X_magic['VIM'], X_magic['CDH1'], c=X_magic['ZEB1'], s=1, cmap='inferno')
plt.show()
magic.plot.animate_magic(X, gene_x='VIM', gene_y='CDH1', gene_color='ZEB1',␣
↪operator=magic_operator)
```

A demo on MAGIC usage for single cell RNA-seq data can be found in this notebook: http://nbviewer.jupyter.org/github/KrishnaswamyLab/magic/blob/master/python/tutorial_notebooks/emt_tutorial.ipynb

A second tutorial analyzing myeloid and erythroid cells in mouse bone marrow is available here: http://nbviewer.jupyter.org/github/KrishnaswamyLab/magic/blob/develop/python/tutorial_notebooks/bonemarrow_tutorial.ipynb

CHAPTER 3

API

3.1 MAGIC

3.2 File Input/Output

magic.io.**load_10X**(*data_dir*, *sparse=True*, *gene_labels='symbol'*, *allow_duplicates=None*)

Basic IO for 10X data produced from the 10X Cellranger pipeline.

A default run of the *cellranger count* command will generate gene-barcode matrices for secondary analysis. For both "raw" and "filtered" output, directories are created containing three files: 'matrix.mtx', 'barcodes.tsv', 'genes.tsv'. Running *phate.io.load_10X(data_dir)* will return a Pandas DataFrame with genes as columns and cells as rows. The returned DataFrame will be ready to use with PHATE.

Parameters

- **data_dir** (*string*) – path to input data directory expects 'matrix.mtx', 'genes.tsv', 'barcodes.tsv' to be present and will raise an error otherwise
- **sparse** (*boolean*) – If True, a sparse Pandas DataFrame is returned.
- **gene_labels** (*string, {'id', 'symbol', 'both'} optional, default: 'symbol'*) – Whether the columns of the dataframe should contain gene ids or gene symbols. If 'both', returns symbols followed by ids in parentheses.
- **allow_duplicates** (*bool, optional (default: None)*) – Whether or not to allow duplicate gene names. If None, duplicates are allowed for dense input but not for sparse input.

Returns **data** – imported data matrix

Return type pandas.DataFrame shape = (n_cell, n_genes)

magic.io.**load_10X_zip**(*filename*, *sparse=True*, *gene_labels='symbol'*, *allow_duplicates=None*)

Basic IO for zipped 10X data produced from the 10X Cellranger pipeline.

Runs *load_10X* after unzipping the data contained in *filename*

Parameters

- **filename** (*string*) – path to zipped input data directory expects 'matrix.mtx', 'genes.tsv', 'barcodes.tsv' to be present and will raise an error otherwise
- **sparse** (*boolean*) – If True, a sparse Pandas DataFrame is returned.
- **gene_labels** (*string, {'id', 'symbol', 'both'} optional, default: 'symbol'*) – Whether the columns of the dataframe should contain gene ids or gene symbols. If 'both', returns symbols followed by ids in parentheses.
- **allow_duplicates** (*bool, optional (default: None)*) – Whether or not to allow duplicate gene names. If None, duplicates are allowed for dense input but not for sparse input.

Returns **data** – imported data matrix

Return type pandas.DataFrame shape = (n_cell, n_genes)

magic.io.**load_csv**(*filename*, *cell_axis='row'*, *delimiter=','*, *gene_names=True*, *cell_names=True*, *sparse=False*, ***kwargs*)

Load a csv file

Parameters

- **filename** (*str*) – The name of the csv file to be loaded
- **cell_axis** (*{'row', 'column'}, optional (default: 'row')*) – If your data has genes on the rows and cells on the columns, use cell_axis='column'
- **delimiter** (*str, optional (default: ',')*) – Use 't' for tab separated values (tsv)
- **gene_names** (*bool*, *str*, array-like, or *None* (default: True)) – If *True*, we assume gene names are in the first row/column. Otherwise expects a filename or an array containing a list of gene symbols or ids
- **cell_names** (*bool*, *str*, array-like, or *None* (default: True)) – If *True*, we assume cell names are in the first row/column. Otherwise expects a filename or an array containing a list of cell barcodes.
- **sparse** (*bool, optional (default: False)*) – If True, loads the data as a pd.SparseDataFrame. This uses less memory but more CPU.
- ****kwargs** (optional arguments for *pd.read_csv*.) –

Returns **data**

Return type pd.DataFrame

magic.io.**load_mtx**(*mtx_file*, *cell_axis='row'*, *gene_names=None*, *cell_names=None*, *sparse=None*)

Load a mtx file

Parameters

- **filename** (*str*) – The name of the mtx file to be loaded
- **cell_axis** (*{'row', 'column'}, optional (default: 'row')*) – If your data has genes on the rows and cells on the columns, use cell_axis='column'
- **gene_names** (*str*, array-like, or *None* (default: None)) – Expects a filename or an array containing a list of gene symbols or ids
- **cell_names** (*str*, array-like, or *None* (default: None)) – Expects a filename or an array containing a list of cell barcodes.

- **sparse** (*bool, optional (default: None)*) – If True, loads the data as a pd.SparseDataFrame. This uses less memory but more CPU.

Returns **data**

Return type pd.DataFrame

magic.io.**load_tsv**(*filename*, *cell_axis='row'*, *delimiter='\t'*, *gene_names=True*, *cell_names=True*, *sparse=False*, ***kwargs*)

Load a csv file

Parameters

- **filename** (*str*) – The name of the csv file to be loaded
- **cell_axis** (*{'row', 'column'}, optional (default: 'row')*) – If your data has genes on the rows and cells on the columns, use cell_axis='column'
- **delimiter** (*str, optional (default: 't')*) – Use ',' for comma separated values (csv)
- **gene_names** (*bool*, *str*, array-like, or *None* (default: True)) – If *True*, we assume gene names are in the first row/column. Otherwise expects a filename or an array containing a list of gene symbols or ids
- **cell_names** (*bool*, *str*, array-like, or *None* (default: True)) – If *True*, we assume cell names are in the first row/column. Otherwise expects a filename or an array containing a list of cell barcodes.
- **sparse** (*bool, optional (default: False)*) – If True, loads the data as a pd.SparseDataFrame. This uses less memory but more CPU.
- ****kwargs** (optional arguments for *pd.read_csv*.) –

Returns **data**

Return type pd.DataFrame

3.3 Data Preprocessing

magic.preprocessing.**library_size_normalize**(*data*, *verbose=False*)

Performs L1 normalization on input data Performs L1 normalization on input data such that the sum of expression values for each cell sums to 1 then returns normalized matrix to the metric space using median UMI count per cell effectively scaling all cells as if they were sampled evenly.

Parameters **data** (*ndarray [n,p]*) – 2 dimensional input data array with n cells and p dimensions

Returns **data_norm** – 2 dimensional array with normalized gene expression values

Return type ndarray [n, p]

CHAPTER 4

Quick Start

To run MAGIC on your dataset, create a MAGIC operator and run *fit_transform*. Here we show an example with a small, artificial dataset located in the MAGIC repository:

```
import magic
import pandas as pd
import matplotlib.pyplot as plt
X = pd.read_csv("MAGIC/data/test_data.csv")
magic_operator = magic.MAGIC()
X_magic = magic_operator.fit_transform(X, genes=['VIM', 'CDH1', 'ZEB1'])
plt.scatter(X_magic['VIM'], X_magic['CDH1'], c=X_magic['ZEB1'], s=1, cmap='inferno')
plt.show()
magic.plot.animate_magic(X, gene_x='VIM', gene_y='CDH1', gene_color='ZEB1',␣
↪operator=magic_operator)
```

class magic.**MAGIC**(*k=10*, *a=15*, *t='auto'*, *n_pca=100*, *knn_dist='euclidean'*, *n_jobs=1*, *random_state=None*, *verbose=1*)

MAGIC operator which performs dimensionality reduction.

Markov Affinity-based Graph Imputation of Cells (MAGIC) is an algorithm for denoising and transcript recover of single cells applied to single-cell RNA sequencing data, as described in van Dijk et al, 2018[1].

Parameters

- **k** (*int, optional, default: 10*) – number of nearest neighbors on which to build kernel
- **a** (*int, optional, default: 15*) – sets decay rate of kernel tails. If None, alpha decaying kernel is not used
- **t** (*int, optional, default: 'auto'*) – power to which the diffusion operator is powered. This sets the level of diffusion. If 'auto', t is selected according to the R squared of the diffused data

[1] Van Dijk D *et al.* (2018), *Recovering Gene Interactions from Single-Cell Data Using Data Diffusion*, Cell.

- **n_pca** (*int, optional, default: 100*) – Number of principal components to use for calculating neighborhoods. For extremely large datasets, using n_pca < 20 allows neighborhoods to be calculated in roughly log(n_samples) time.
- **knn_dist** (*string, optional, default: 'euclidean'*) – recommended values: 'euclidean', 'cosine', 'precomputed' Any metric from *scipy.spatial.distance* can be used distance metric for building kNN graph. If 'precomputed', *data* should be an n_samples x n_samples distance or affinity matrix
- **n_jobs** (*integer, optional, default: 1*) – The number of jobs to use for the computation. If -1 all CPUs are used. If 1 is given, no parallel computing code is used at all, which is useful for debugging. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used. Thus for n_jobs = -2, all CPUs but one are used
- **random_state** (*integer or numpy.RandomState, optional, default: None*) – The generator used to initialize random PCA If an integer is given, it fixes the seed Defaults to the global *numpy* random number generator
- **verbose** (*int* or *boolean*, optional (default: 1)) – If *True* or > *0*, print status messages

X
array-like, shape=[n_samples, n_dimensions] – Input data

X_magic
array-like, shape=[n_samples, n_dimensions] – Output data

graph
graphtools.BaseGraph – The graph built on the input data

Examples

```
>>> import magic
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> X = pd.read_csv("../../data/test_data.csv")
>>> X.shape
(500, 197)
>>> magic_operator = magic.MAGIC()
>>> X_magic = magic_operator.fit_transform(X, genes=['VIM', 'CDH1', 'ZEB1'])
>>> X_magic.shape
(500, 3)
>>> magic_operator.set_params(t=7)
MAGIC(a=15, k=5, knn_dist='euclidean', n_jobs=1, n_pca=100, random_state=None,
   t=7, verbose=1)
>>> X_magic = magic_operator.transform(genes=['VIM', 'CDH1', 'ZEB1'])
>>> X_magic.shape
(500, 3)
>>> X_magic = magic_operator.transform(genes="all_genes")
>>> X_magic.shape
(500, 197)
>>> plt.scatter(X_magic['VIM'], X_magic['CDH1'], c=X_magic['ZEB1'], s=1, cmap=
↪'inferno')
>>> plt.show()
>>> magic.plot.animate_magic(X, gene_x='VIM', gene_y='CDH1', gene_color='ZEB1',␣
↪operator=magic_operator)
```

References

calculate_error(*data*, *data_prev=None*, *weights=None*, *subsample_genes=None*)
Calculates difference before and after diffusion

Parameters

- **data** (*array-like*) – current data matrix
- **data_prev** (*array-like, optional (default: None)*) – previous data matrix. If None, *data* is simply prepared for comparison and no error is returned
- **weights** (*list-like, optional (default: None)*) – weightings for dimensions of data. If None, dimensions are equally weighted
- **subsample_genes** (*like-like, optional (default: None)*) – genes to select in subsampling. If None, no subsampling is performed

Returns

- **error** (*float*) – Procrustes disparity value
- **data_curr** (*array-like*) – transformed data to use for the next comparison

diff_op
The diffusion operator calculated from the data

fit(*X*)
Computes the diffusion operator

Parameters **X** (*array, shape=[n_samples, n_features]*) – input data with *n_samples* samples and *n_dimensions* dimensions. Accepted data types: *numpy.ndarray*, *scipy.sparse.spmatrix*, *pd.DataFrame*, *anndata.AnnData*. If *knn_dist* is 'precomputed', *data* should be a n_samples x n_samples distance or affinity matrix

Returns **magic_operator** – The estimator object

Return type MAGIC

fit_transform(*X*, ***kwargs*)
Computes the diffusion operator and the position of the cells in the embedding space

Parameters

- **X** (*array, shape=[n_samples, n_features]*) – input data with *n_samples* samples and *n_dimensions* dimensions. Accepted data types: *numpy.ndarray*, *scipy.sparse.spmatrix*, *pd.DataFrame*, *anndata.AnnData* If *knn_dist* is 'precomputed', *data* should be a n_samples x n_samples distance or affinity matrix
- **kwargs** (further arguments for *PHATE.transform()*) – Keyword arguments as specified in `transform()`

Returns **X_magic** – The gene expression values after diffusion

Return type array, shape=[n_samples, n_genes]

impute(*data*, *t_max=20*, *plot=False*, *ax=None*, *max_genes_compute_t=500*, *threshold=0.001*)
Peform MAGIC imputation

Parameters

- **data** (*graphtools.Graph, graphtools.Data or array-like*) – Input data

- **t_max** (*int, optional (default: 20)*) – Maximum value of t to consider for optimal t selection
- **plot** (*bool, optional (default: False)*) – Plot the optimal t selection graph
- **ax** (*matplotlib.Axes, optional (default: None)*) – Axis on which to plot. If None, a new axis is created
- **max_genes_compute_t** (*int, optional (default: 500)*) – Above this number, genes will be subsampled for optimal t selection
- **threshold** (*float, optional (default: 0.001)*) – Threshold after which Procrustes disparity is considered to have converged for optimal t selection

Returns **X_magic** – Imputed data

Return type array-like, shape=[n_samples, n_pca]

set_params(***params*)

Set the parameters on this estimator.

Any parameters not given as named arguments will be left at their current value.

Parameters

- **k** (*int, optional, default: 10*) – number of nearest neighbors on which to build kernel
- **a** (*int, optional, default: 15*) – sets decay rate of kernel tails. If None, alpha decaying kernel is not used
- **t** (*int, optional, default: 'auto'*) – power to which the diffusion operator is powered. This sets the level of diffusion. If 'auto', t is selected according to the R squared of the diffused data
- **n_pca** (*int, optional, default: 100*) – Number of principal components to use for calculating neighborhoods. For extremely large datasets, using n_pca < 20 allows neighborhoods to be calculated in roughly log(n_samples) time.
- **knn_dist** (*string, optional, default: 'euclidean'*) – recommended values: 'euclidean', 'cosine', 'precomputed' Any metric from *scipy.spatial.distance* can be used distance metric for building kNN graph. If 'precomputed', *data* should be an n_samples x n_samples distance or affinity matrix
- **n_jobs** (*integer, optional, default: 1*) – The number of jobs to use for the computation. If -1 all CPUs are used. If 1 is given, no parallel computing code is used at all, which is useful for debugging. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used. Thus for n_jobs = -2, all CPUs but one are used
- **random_state** (*integer or numpy.RandomState, optional, default: None*) – The generator used to initialize random PCA If an integer is given, it fixes the seed Defaults to the global *numpy* random number generator
- **verbose** (*int* or *boolean*, optional (default: 1)) – If *True* or > *0*, print status messages

Returns

Return type self

transform(*X=None*, *genes=None*, *t_max=20*, *plot_optimal_t=False*, *ax=None*)

Computes the values of genes after diffusion

Parameters

- **X** (*array, optional, shape=[n_samples, n_features]*) – input data with *n_samples* samples and *n_dimensions* dimensions. Not required, since MAGIC does not embed cells not given in the input matrix to *MAGIC.fit()*. Accepted data types: *numpy.ndarray*, *scipy.sparse.spmatrix*, *pd.DataFrame*, *anndata.AnnData*.
- **genes** (*list or {"all_genes", "pca_only"}, optional (default: None)*) – List of genes, either as integer indices or column names if input data is a pandas DataFrame. If "all_genes", the entire smoothed matrix is returned. If "pca_only", PCA on the smoothed data is returned. If None, the entire matrix is also returned, but a warning may be raised if the resultant matrix is very large.
- **t_max** (*int, optional, default: 20*) – maximum t to test if *t* is set to 'auto'
- **plot_optimal_t** (*boolean, optional, default: False*) – If true and *t* is set to 'auto', plot the disparity used to select t
- **ax** (*matplotlib.axes.Axes, optional*) – If given and *plot_optimal_t* is true, plot will be drawn on the given axis.

Returns **X_magic** – The gene expression values after diffusion

Return type array, shape=[n_samples, n_genes]

Python Module Index

m

Index