astir

Jinyu Hou, Sunyun Lee, Michael Geuenich, Kieran Campbell

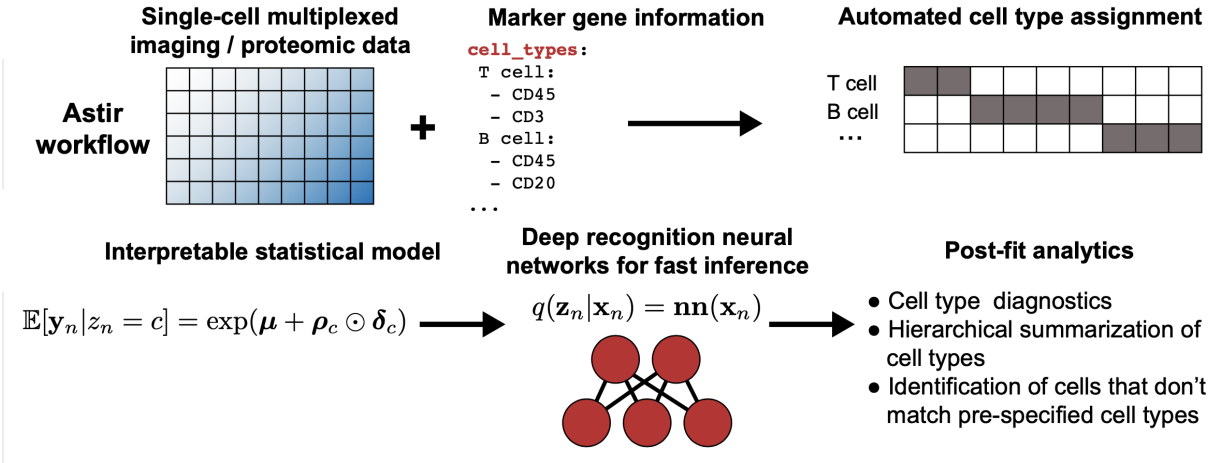
Jun 24, 2021

CONTENTS

`astir` is a modelling framework for the assignment of cell type across a range of single-cell technologies such as Imaging Mass Cytometry (IMC). `astir` is built using pytorch and uses recognition networks for fast minibatch stochastic variational inference.

Key applications:

- Automated assignment of cell type and state from highly multiplexed imaging and proteomic data
- Diagnostic measures to check quality of resulting type and state inferences
- Ability to map new data to cell types and states trained on existing data using recognition neural networks
- A range of plotting and data loading utilities



CHAPTER

ONE

GETTING STARTED

Launch the interactive tutorial:

See the full documentation and check out the tutorials.

CHAPTER

TWO

AUTHORS

Jinyu Hou, Sunyun Lee, Michael Geuenich, Kieran Campbell
Lunenfeld-Tanenbaum Research Institute & University of Toronto

2.1 Installation

2.1.1 Prerequisites

Install python 3.7 Astir uses python 3.*

2.1.2 Astir installation

PyPI

```
pip3 install astir
```

Dev

Clone this repo and run

```
git clone https://github.com/camlab-bioml/astir.git
cd astir
pip install -e .
```

2.2 Tutorials

2.2.1 Getting started with astir

Table of Contents:

0. Load necessary libraries

[39]:
```
# !pip install -e ../../..
import os
import sys
module_path = os.path.abspath(os.path.join('../../..'))
if module_path not in sys.path:
    sys.path.append(module_path)
```

[5]:
```
from astir.data import from_csv_yaml

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%load_ext autoreload
%autoreload 2
%matplotlib inline
```

1. Load data

We start by reading expression data in the form of a csv file and marker gene information in the form of a yaml file:

[1]:
```
expression_mat_path = "../../../tests/test-data/sce.csv"
# expression_mat_path = "data/sample_data.csv"
yaml_marker_path = "../../../tests/test-data/jackson-2020-markers.yml"
```

Note: Expression data should already be in a cleaned and normalized form. For IMC data, we perform this by an arcsinh transformation of the data with a cofactor so `transformed_expression = archsinh(raw_expression / cofactor)`, where typically `cofactor=5`.

We can view both the expression data and marker data:

[2]:
```
!head -n 20 ../../../tests/test-data/jackson-2020-markers.yml
```

```
cell_states:
  RTK_signalling:
```

(continues on next page)

(continued from previous page)

```
    - Her2
    - EGFR
  proliferation:
    - Ki-67
    - phospho Histone
  mTOR_signalling:
    - phospho mTOR
    - phospho S6
  apoptosis:
    - cleaved PARP
    - Cleaved Caspase3

cell_types:
  stromal:
    - Vimentin
    - Fibronectin
  B cells:
```

[6]:
```
pd.read_csv(expression_mat_path, index_col=0)[['EGFR','E-Cadherin', 'CD45', 'Cytokeratin␣
↪5']].head()
```

[6]:

	EGFR	E-Cadherin	CD45	Cytokeratin 5
BaselTMA_SP41_186_X5Y4_3679	0.346787	0.938354	0.227730	0.095283
BaselTMA_SP41_153_X7Y5_246	0.833752	1.364884	0.068526	0.124031
BaselTMA_SP41_20_X12Y5_197	0.110006	0.177361	0.301222	0.052750
BaselTMA_SP41_14_X1Y8_84	0.282666	1.122174	0.606941	0.093352
BaselTMA_SP41_166_X15Y4_266	0.209066	0.402554	0.588273	0.064545

Then we can create an astir object using the `from_csv_yaml` function. For more data loading options, see the data loading tutorial.

[7]:
```
ast = from_csv_yaml(expression_mat_path, marker_yaml=yaml_marker_path)
print(ast)
```

```
Astir object, 6 cell types, 4 cell states, 100 cells
```

2. Fitting cell types

To fit cell types, simply call

[8]:
```
ast.fit_type(max_epochs=10, n_init=3, n_init_epochs=2)
```

```
training restart 1/3: 100%|| 2/2 [ 4.51epochs/s, current loss: 745.5]
training restart 2/3: 100%|| 2/2 [108.41epochs/s, current loss: 776.6]
training restart 3/3: 100%|| 2/2 [40.70epochs/s, current loss: 774.7]
training restart (final): 100%|| 10/10 [82.58epochs/s, current loss: 709.0]
/Users/jinelles.h/Documents/Camlab/astir-top-level/astir/astir/astir.py:178: UserWarning:
↪ Maximum epochs reached. More iteration may be needed to complete the training.
  warnings.warn(msg)
```

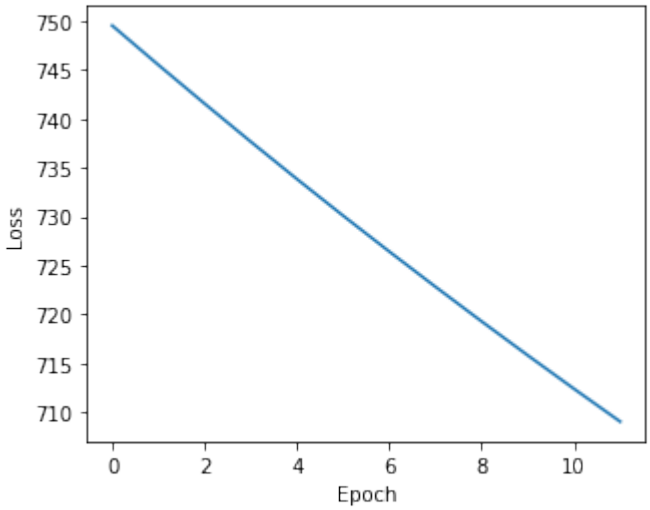
Note: **Controlling inference** There are many different options for controlling inference in the `fit_type` function, including `max_epochs` (maximum number of epochs to train), `learning_rate` (ADAM optimizer learning rate),

`batch_size` (minibatch size), `delta_loss` (stops iteration once the change in loss falls below this value), `n_inits` (number of restarts using random initializations). For full details, see the function documentation.

We should always plot the losses to assess convergence:

[9]:
```
plt.figure(figsize=(5,4))
plt.plot(np.arange(len(ast.get_type_losses())), ast.get_type_losses())
plt.ylabel("Loss")
plt.xlabel("Epoch")
```

[9]:
```
Text(0.5, 0, 'Epoch')
```



We can then get cell type assignment probabilities by calling

[10]:
```
assignments = ast.get_celltype_probabilities()
assignments
```

[10]:

	stromal	B cells	T cells	macrophage	\
BaselTMA_SP41_186_X5Y4_3679	0.088444	0.084654	0.198811	0.105010	
BaselTMA_SP41_153_X7Y5_246	0.110921	0.156938	0.193806	0.122539	
BaselTMA_SP41_20_X12Y5_197	0.099029	0.107530	0.210399	0.135788	
BaselTMA_SP41_14_X1Y8_84	0.117755	0.119934	0.221191	0.103741	
BaselTMA_SP41_166_X15Y4_266	0.102338	0.106135	0.221755	0.135476	
...	...	...	...	...	
BaselTMA_SP41_114_X13Y4_1057	0.100268	0.128181	0.204113	0.140633	
BaselTMA_SP41_141_X11Y2_2596	0.112603	0.143363	0.201669	0.148209	
BaselTMA_SP41_100_X15Y5_170	0.111955	0.130958	0.203715	0.144000	
BaselTMA_SP41_14_X1Y8_2604	0.076987	0.084889	0.218953	0.114034	
BaselTMA_SP41_186_X5Y4_81	0.131032	0.128943	0.206584	0.123474	

	epithelial(basal)	epithelial(luminal)	Other
BaselTMA_SP41_186_X5Y4_3679	0.199379	0.119610	0.204091
BaselTMA_SP41_153_X7Y5_246	0.200939	0.114491	0.100367
BaselTMA_SP41_20_X12Y5_197	0.174190	0.126138	0.146926
BaselTMA_SP41_14_X1Y8_84	0.150916	0.140125	0.146337
BaselTMA_SP41_166_X15Y4_266	0.157753	0.125728	0.150815
...	...	...	...

(continues on next page)

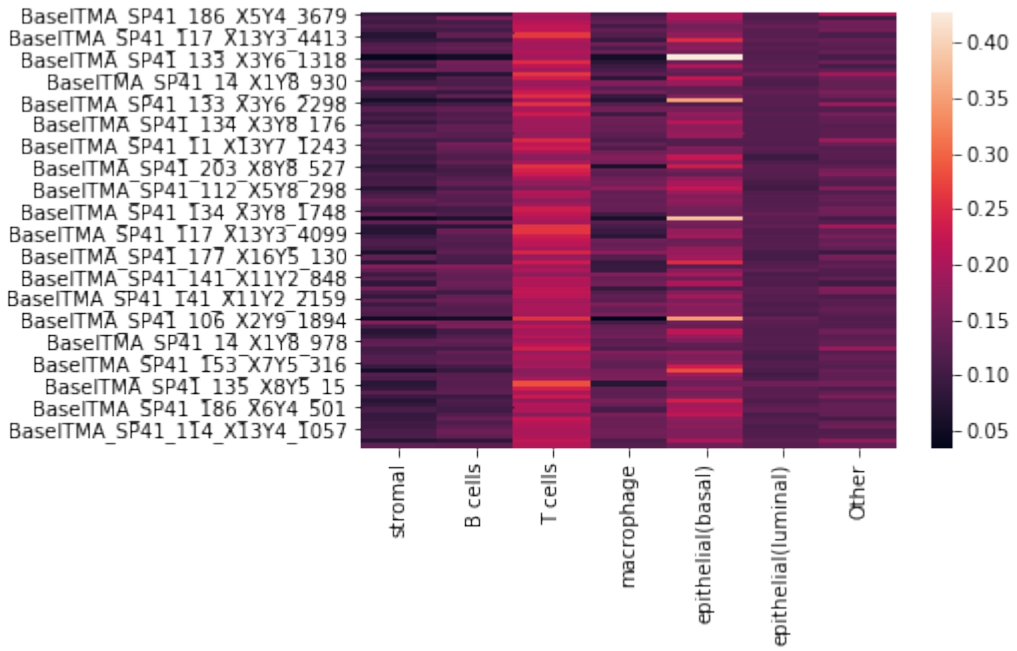
(continued from previous page)

```
BaselTMA_SP41_114_X13Y4_1057          0.170518          0.111839  0.144448
BaselTMA_SP41_141_X11Y2_2596          0.161044          0.116404  0.116708
BaselTMA_SP41_100_X15Y5_170           0.163306          0.119814  0.126253
BaselTMA_SP41_14_X1Y8_2604            0.200917          0.130515  0.173706
BaselTMA_SP41_186_X5Y4_81             0.145218          0.125953  0.138796

[100 rows x 7 columns]
```

We can also visualize the assignment probabilities using a heatmap:

[11]:
```
sns.heatmap(assignments)
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8041845410>
```



where each row corresponds to a cell, and each column to a cell type, with the entry being the probability of that cell belonging to a particular cell type.

To fetch an array corresponding to the most likely cell type assignments, call

[12]:
```
ast.get_celltypes()
```

[12]:
```
                              cell_type
BaselTMA_SP41_186_X5Y4_3679     Unknown
BaselTMA_SP41_153_X7Y5_246      Unknown
BaselTMA_SP41_20_X12Y5_197      Unknown
BaselTMA_SP41_14_X1Y8_84        Unknown
BaselTMA_SP41_166_X15Y4_266     Unknown
...                                 ...
BaselTMA_SP41_114_X13Y4_1057    Unknown
BaselTMA_SP41_141_X11Y2_2596    Unknown
BaselTMA_SP41_100_X15Y5_170     Unknown
BaselTMA_SP41_14_X1Y8_2604      Unknown
```

(continues on next page)

(continued from previous page)

```
BaselTMA_SP41_186_X5Y4_81      Unknown

[100 rows x 1 columns]
```

Cell type diagnostics

It is important to run diagnostics to ensure that cell types express their markers at higher levels than other cell types. To do this, run the `diagnostics_celltype()` function, which will alert to any issues if a cell type doesn't express its marker signficantly higher than an alternative cell type (for which that protein isn't a marker):

[12]:
```
ast.diagnostics_celltype().head(n=10)
```

[12]:
```
Empty DataFrame
Columns: [feature, should be expressed higher in, than, mean cell type 1, mean cell type␣
↪2, p-value, note]
Index: []
```

Note: In this tutorial, we end up with many "Only 1 cell in a type: comparison not possible" notes - this is simply because the small dataset size results in only a single cell assigned to many types, making statistical testing infeasible.

Calling `ast.diagnostics_celltype()` returns a `pd.DataFrame`, where each column corresponds to a particular protein and two cell types, with a warning if the protein is not expressed at higher levels in the cell type for which it is a marker than the cell type for which it is not.

The diagnostics:

1. Iterates through every cell type and every marker for that cell type
2. Given a cell type c and marker g, find the set of cell types D that don't have g as a marker
3. For each cell type d in D, perform a t-test between the expression of marker g in c vs d
4. If g is not expressed significantly higher (at significance *alpha*), output a diagnostic explaining this for further investigation.

If multiple issues are found, the markers and cell types may need refined.

3. Fitting cell state

> **Caution:** Cell state fitting in Astir is currently experimental and not included in the initial paper.

Similarly as before, to fit cell state, call

[45]:
```
ast.fit_state(batch_size = 1024, learning_rate=1e-3, max_epochs=10)
```

```
/Users/jinelles.h/Documents/Camlab/astir-top-level/astir/astir/astir.py:222: UserWarning:
↪ Delta loss batch size is greater than the number of epochs
  warnings.warn("Delta loss batch size is greater than the number of epochs")
training restart 1/5: 100%|| 5/5 [129.59epochs/s, current loss: 196.5]
training restart 2/5: 100%|| 5/5 [117.63epochs/s, current loss: 231.9]
training restart 3/5: 100%|| 5/5 [124.46epochs/s, current loss: 217.7]
```

(continues on next page)

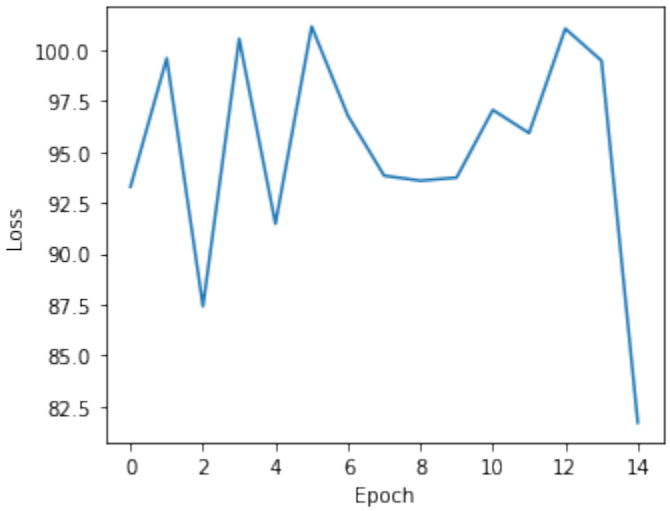
(continued from previous page)

```
training restart 4/5: 100%|| 5/5 [88.48epochs/s, current loss: 275.4]
training restart 5/5: 100%|| 5/5 [100.32epochs/s, current loss: 202.0]
training restart (final): 100%|| 10/10 [92.36epochs/s, current loss: 170.3]
/Users/jinelles.h/Documents/Camlab/astir-top-level/astir/astir/astir.py:280: UserWarning:
↪ Maximum epochs reached. More iteration may be needed to complete the training.
  warnings.warn(msg)
```

and similary plot the losses via

[14]:
```
plt.figure(figsize=(5,4))
plt.plot(np.arange(len(ast.get_state_losses())), ast.get_state_losses())
plt.ylabel("Loss")
plt.xlabel("Epoch")
```

[14]:
```
Text(0.5, 0, 'Epoch')
```



and cell state assignments can be inferred via

[15]:
```
states = ast.get_cellstates()
states
```

[15]:

	RTK_signalling	proliferation	mTOR_signalling	apoptosis
BaselTMA_SP41_186_X5Y4_3679	0.568386	0.900861	0.656078	0.548110
BaselTMA_SP41_153_X7Y5_246	0.421357	0.000000	0.524859	
BaselTMA_SP41_20_X12Y5_197	0.944829	0.561159	0.979277	
BaselTMA_SP41_14_X1Y8_84	0.858426	0.705787	0.938068	
BaselTMA_SP41_166_X15Y4_266	0.933672	0.574031	0.980568	
...	...	...	...	
BaselTMA_SP41_114_X13Y4_1057	0.881551	0.447002	0.899008	
BaselTMA_SP41_141_X11Y2_2596	0.767853	0.684847	0.856773	
BaselTMA_SP41_100_X15Y5_170	0.952977	0.548220	0.977899	
BaselTMA_SP41_14_X1Y8_2604	0.836241	0.692617	0.908256	
BaselTMA_SP41_186_X5Y4_81	0.691698	0.719179	0.705012	

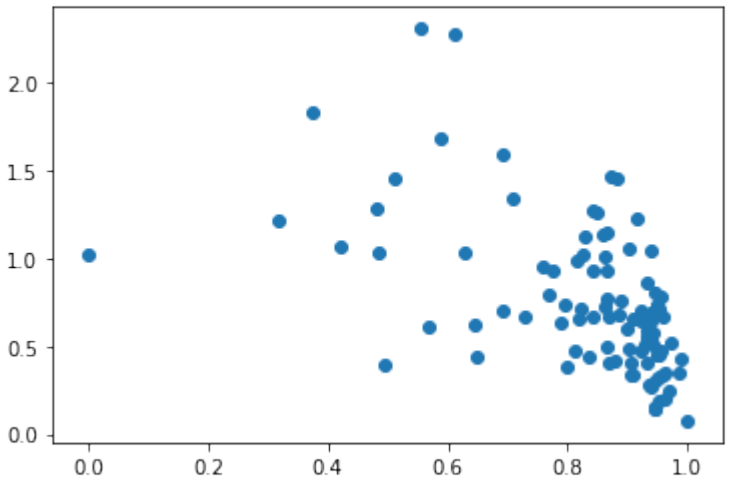
(continues on next page)

(continued from previous page)

```
BaselTMA_SP41_153_X7Y5_246     0.820946
BaselTMA_SP41_20_X12Y5_197     0.778787
BaselTMA_SP41_14_X1Y8_84       0.697319
BaselTMA_SP41_166_X15Y4_266    0.764967
...                                 ...
BaselTMA_SP41_114_X13Y4_1057   1.000000
BaselTMA_SP41_141_X11Y2_2596   0.722046
BaselTMA_SP41_100_X15Y5_170    0.786169
BaselTMA_SP41_14_X1Y8_2604     0.700867
BaselTMA_SP41_186_X5Y4_81      0.660071

[100 rows x 4 columns]
```

[16]:
```
plt.scatter(
    states['RTK_signalling'],
    ast.get_state_dataset().get_exprs_df()['Her2']
)
```

[16]:
```
<matplotlib.collections.PathCollection at 0x7f8041709d90>
```



Cell state diagnostics

It is important to run diagnostics on cell states model for the same reasons stated for the cell type model. `Astir.diagnostics_cellstate()` spots any non marker protein and pathway pairs whose expressions are higher than those of the marker proteins of the pathway.

[17]:
```
ast.diagnostics_cellstate().head(n=10)
```

[17]:
```
           pathway     protein A  correlation of protein A         protein B  \
0   RTK_signalling          Her2                 -0.542371            Ki-67
1   RTK_signalling          Her2                 -0.542371        phospho S6
2    proliferation         Ki-67                  0.241679              Her2
3    proliferation         Ki-67                  0.241679        phospho S6
4    proliferation         Ki-67                  0.241679      phospho mTOR
5  mTOR_signalling  phospho mTOR                 -0.776636  Cleaved Caspase3
```

(continues on next page)

(continued from previous page)

```
6  mTOR_signalling  phospho mTOR                -0.776636              EGFR
7  mTOR_signalling  phospho mTOR                -0.776636              Her2
8  mTOR_signalling  phospho mTOR                -0.776636             Ki-67
9  mTOR_signalling  phospho mTOR                -0.776636      cleaved PARP

   correlation of protein B                                            note
0                 -0.062491  Her2 is marker for RTK_signalling but Ki-67 isn't
1                 -0.519752  Her2 is marker for RTK_signalling but phospho ...
2                  0.344042   Ki-67 is marker for proliferation but Her2 isn't
3                  0.411222  Ki-67 is marker for proliferation but phospho ...
4                  0.316836  Ki-67 is marker for proliferation but phospho ...
5                 -0.557334  phospho mTOR is marker for mTOR_signalling but...
6                 -0.412919  phospho mTOR is marker for mTOR_signalling but...
7                 -0.405998  phospho mTOR is marker for mTOR_signalling but...
8                 -0.012072  phospho mTOR is marker for mTOR_signalling but...
9                 -0.557334  phospho mTOR is marker for mTOR_signalling but...
```

Calling `ast.diagnostics_cellstate()` returns a `pd.DataFrame`, where each column corresponds to a particular protein and two cell types, with a warning if the protein is not expressed at higher levels in the cell state for which it is a marker than the cell state for which it is not.

The diagnostics:

1. Get correlations between all cell states and proteins
2. For each cell state c, get the smallest correlation with marker g
3. For each cell state c and its non marker g, find any correlation that is bigger than those smallest correlation for c.
4. Any c and g pairs found in step 3 will be included in the output of `Astir.diagnostics_cellstate()`, including an explanation.

If multiple issues are found, the markers and cell states may need refined.

4. Saving results

Both cell type and cell state information can easily be saved to disk via

[18]:
```
ast.type_to_csv("data/cell-types.csv")
ast.state_to_csv("data/cell-states.csv")
```

[19]:
```
!head -n 3 data/cell-types.csv
```

```
,cell_type
BaselTMA_SP41_186_X5Y4_3679,Unknown
BaselTMA_SP41_153_X7Y5_246,Unknown
```

[20]:
```
!head -n 3 data/cell-states.csv
```

```
,RTK_signalling,proliferation,mTOR_signalling,apoptosis
BaselTMA_SP41_186_X5Y4_3679,0.5683861877028941,0.9008611646823064,0.656078413193121,0.
↪5481102160359178
BaselTMA_SP41_153_X7Y5_246,0.42135746208636826,0.0,0.5248592191573492,0.8209456743672588
```

where the first (unnamed) column always corresponds to the cell name/ID.

5. Accessing internal functions and data

Data stored in `astir` objects is in the form of an `SCDataSet`. These can be retrieved via

[21]:
```
celltype_data = ast.get_type_dataset()
celltype_data
```

[21]:
```
<astir.data.scdataset.SCDataset at 0x7f8041983290>
```

and similarly for cell state via `ast.get_state_dataset()`.

These have several helper functions to retrieve relevant information to the dataset:

[22]:
```
celltype_data.get_cell_names()[0:4] # cell names
```

[22]:
```
['BaselTMA_SP41_186_X5Y4_3679',
 'BaselTMA_SP41_153_X7Y5_246',
 'BaselTMA_SP41_20_X12Y5_197',
 'BaselTMA_SP41_14_X1Y8_84']
```

[23]:
```
celltype_data.get_classes() # cell type names
```

[23]:
```
['stromal',
 'B cells',
 'T cells',
 'macrophage',
 'epithelial(basal)',
 'epithelial(luminal)']
```

[24]:
```
print(celltype_data.get_n_classes()) # number of cell types
print(celltype_data.get_n_features()) # number of features / proteins
```

```
6
14
```

[25]:
```
celltype_data.get_exprs() # Return a torch tensor corresponding to the expression data␣
↪used
```

[25]:
```
tensor([[0.1026, 0.1004, 0.2277,  ..., 0.6097, 2.2151, 0.7714],
        [0.1081, 0.0176, 0.0685,  ..., 1.0622, 0.5026, 3.9632],
        [0.0498, 0.0943, 0.3012,  ..., 0.1601, 0.8102, 0.0481],
        ...,
        [0.0695, 0.0119, 0.0869,  ..., 0.4487, 0.7593, 1.4923],
        [0.0929, 0.1266, 0.2395,  ..., 0.4405, 2.2464, 0.4174],
        [0.0618, 0.1439, 0.2476,  ..., 0.7055, 3.1238, 0.2552]],
       dtype=torch.float64)
```

[26]:
```
celltype_data.get_exprs_df() # Return a pandas DataFrame corresponding to the expression␣
↪data used
```

[26]:

	CD20	CD3	CD45	CD68	\
BaselTMA_SP41_186_X5Y4_3679	0.102576	0.100401	0.227730	2.227252	
BaselTMA_SP41_153_X7Y5_246	0.108137	0.017637	0.068526	0.208297	
BaselTMA_SP41_20_X12Y5_197	0.049809	0.094316	0.301222	0.581624	
BaselTMA_SP41_14_X1Y8_84	0.024256	0.140441	0.606941	0.490982	
BaselTMA_SP41_166_X15Y4_266	0.138571	0.111722	0.588273	1.039967	

(continues on next page)

(continued from previous page)

```
...                                 ...       ...       ...       ...
BaselTMA_SP41_114_X13Y4_1057  0.185995  0.049244  0.151869  0.296231
BaselTMA_SP41_141_X11Y2_2596  0.100324  0.024006  0.025612  0.091424
BaselTMA_SP41_100_X15Y5_170   0.069503  0.011859  0.086852  0.538247
BaselTMA_SP41_14_X1Y8_2604    0.092944  0.126645  0.239459  1.967150
BaselTMA_SP41_186_X5Y4_81     0.061784  0.143930  0.247627  0.316248

                           Cytokeratin 14  Cytokeratin 19  Cytokeratin 5  \
BaselTMA_SP41_186_X5Y4_3679         0.195163        0.190923       0.095283
BaselTMA_SP41_153_X7Y5_246          0.234853        0.685858       0.124031
BaselTMA_SP41_20_X12Y5_197          0.072666        0.115979       0.052750
BaselTMA_SP41_14_X1Y8_84            0.165863        0.652143       0.093352
BaselTMA_SP41_166_X15Y4_266         0.162696        0.086235       0.064545
...                                      ...             ...            ...
BaselTMA_SP41_114_X13Y4_1057        0.151970        0.175114       0.108374
BaselTMA_SP41_141_X11Y2_2596        0.049996        0.324138       0.083866
BaselTMA_SP41_100_X15Y5_170         0.037131        0.321179       0.096496
BaselTMA_SP41_14_X1Y8_2604          0.171216        0.132563       0.055748
BaselTMA_SP41_186_X5Y4_81           0.232111        0.145036       0.080324

                           Cytokeratin 7  Cytokeratin 8/18  E-Cadherin  \
BaselTMA_SP41_186_X5Y4_3679        0.057050          0.461040    0.938354
BaselTMA_SP41_153_X7Y5_246         0.485330          0.382767    1.364884
BaselTMA_SP41_20_X12Y5_197         0.035875          0.020290    0.177361
BaselTMA_SP41_14_X1Y8_84           0.351700          0.904383    1.122174
BaselTMA_SP41_166_X15Y4_266        0.009627          0.046967    0.402554
...                                     ...               ...         ...
BaselTMA_SP41_114_X13Y4_1057       0.021539          0.207346    2.120132
BaselTMA_SP41_141_X11Y2_2596       0.069627          0.614213    1.637392
BaselTMA_SP41_100_X15Y5_170        0.000000          0.696445    0.861641
BaselTMA_SP41_14_X1Y8_2604         0.082334          0.128377    0.532135
BaselTMA_SP41_186_X5Y4_81          0.000000          0.208533    0.899182

                           Fibronectin      Her2  Vimentin  pan Cytokeratin
BaselTMA_SP41_186_X5Y4_3679    1.829905  0.609694  2.215089         0.771352
BaselTMA_SP41_153_X7Y5_246     1.226994  1.062229  0.502627         3.963248
BaselTMA_SP41_20_X12Y5_197     2.222520  0.160135  0.810243         0.048100
BaselTMA_SP41_14_X1Y8_84       1.402750  1.133448  1.742495         1.917118
BaselTMA_SP41_166_X15Y4_266    2.669947  0.558439  1.659587         0.687005
...                                 ...       ...       ...              ...
BaselTMA_SP41_114_X13Y4_1057   0.743909  1.460545  0.069651         1.666665
BaselTMA_SP41_141_X11Y2_2596   0.860960  0.789372  0.000000         2.584532
BaselTMA_SP41_100_X15Y5_170    1.724828  0.448688  0.759268         1.492342
BaselTMA_SP41_14_X1Y8_2604     1.899111  0.440527  2.246434         0.417445
BaselTMA_SP41_186_X5Y4_81      2.475906  0.705456  3.123784         0.255158

[100 rows x 14 columns]
```

[27]:
```
ast.normalize()
```

[28]:
```
ast.get_type_dataset().get_exprs_df()
```

[28]:

	CD20	CD3	CD45	CD68	\
BaselTMA_SP41_186_X5Y4_3679	0.020514	0.020079	0.045530	0.384408	
BaselTMA_SP41_153_X7Y5_246	0.021626	0.003527	0.013705	0.041647	
BaselTMA_SP41_20_X12Y5_197	0.009962	0.018862	0.060208	0.116064	
BaselTMA_SP41_14_X1Y8_84	0.004851	0.028085	0.121092	0.098039	
BaselTMA_SP41_166_X15Y4_266	0.027711	0.022343	0.117385	0.206522	
...	...	...	...	...	
BaselTMA_SP41_114_X13Y4_1057	0.037190	0.009849	0.030369	0.059212	
BaselTMA_SP41_141_X11Y2_2596	0.020063	0.004801	0.005122	0.018284	
BaselTMA_SP41_100_X15Y5_170	0.013900	0.002372	0.017370	0.107443	
BaselTMA_SP41_14_X1Y8_2604	0.018588	0.025326	0.047873	0.383928	
BaselTMA_SP41_186_X5Y4_81	0.012356	0.028782	0.049505	0.063207	

	Cytokeratin 14	Cytokeratin 19	Cytokeratin 5	\
BaselTMA_SP41_186_X5Y4_3679	0.039023	0.038175	0.019055	
BaselTMA_SP41_153_X7Y5_246	0.046953	0.136745	0.024804	
BaselTMA_SP41_20_X12Y5_197	0.014533	0.023194	0.010550	
BaselTMA_SP41_14_X1Y8_84	0.033167	0.130062	0.018669	
BaselTMA_SP41_166_X15Y4_266	0.032533	0.017246	0.012909	
...	...	...	...	
BaselTMA_SP41_114_X13Y4_1057	0.030389	0.035016	0.021673	
BaselTMA_SP41_141_X11Y2_2596	0.009999	0.064782	0.016772	
BaselTMA_SP41_100_X15Y5_170	0.007426	0.064192	0.019298	
BaselTMA_SP41_14_X1Y8_2604	0.034236	0.026509	0.011149	
BaselTMA_SP41_186_X5Y4_81	0.046406	0.029003	0.016064	

	Cytokeratin 7	Cytokeratin 8/18	E-Cadherin	\
BaselTMA_SP41_186_X5Y4_3679	0.011410	0.092078	0.186586	
BaselTMA_SP41_153_X7Y5_246	0.096914	0.076479	0.269695	
BaselTMA_SP41_20_X12Y5_197	0.007175	0.004058	0.035465	
BaselTMA_SP41_14_X1Y8_84	0.070282	0.179905	0.222592	
BaselTMA_SP41_166_X15Y4_266	0.001925	0.009393	0.080424	
...	...	...	...	
BaselTMA_SP41_114_X13Y4_1057	0.004308	0.041457	0.412250	
BaselTMA_SP41_141_X11Y2_2596	0.013925	0.122536	0.321891	
BaselTMA_SP41_100_X15Y5_170	0.000000	0.138843	0.171486	
BaselTMA_SP41_14_X1Y8_2604	0.016466	0.025673	0.106227	
BaselTMA_SP41_186_X5Y4_81	0.000000	0.041695	0.178881	

	Fibronectin	Her2	Vimentin	pan Cytokeratin
BaselTMA_SP41_186_X5Y4_3679	0.358267	0.121639	0.429674	0.153665
BaselTMA_SP41_153_X7Y5_246	0.243000	0.210879	0.100357	0.726918
BaselTMA_SP41_20_X12Y5_197	0.431033	0.032021	0.161348	0.009620
BaselTMA_SP41_14_X1Y8_84	0.276994	0.224792	0.341804	0.374601
BaselTMA_SP41_166_X15Y4_266	0.511404	0.111457	0.326107	0.136972
...	...	...	...	...
BaselTMA_SP41_114_X13Y4_1057	0.148238	0.288107	0.013930	0.327450
BaselTMA_SP41_141_X11Y2_2596	0.171352	0.157226	0.000000	0.496282
BaselTMA_SP41_100_X15Y5_170	0.338466	0.089618	0.151276	0.294206
BaselTMA_SP41_14_X1Y8_2604	0.371236	0.087992	0.435399	0.083392
BaselTMA_SP41_186_X5Y4_81	0.476898	0.140627	0.589937	0.051009

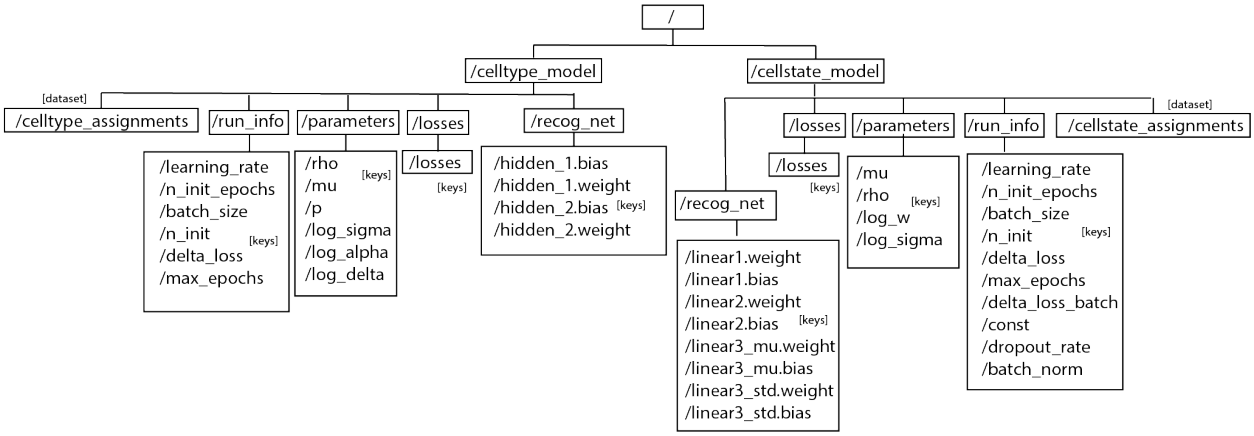
[100 rows x 14 columns]

6. Saving models

After fixing the models, we can save the cell type/state assignment, the losses, the parameters (e.g. `mu`, `rho`, `log_sigma`, etc) and the run informations (e.g. `batch_size`, `learning_rate`, `delta_loss`, etc) to an hdf5 file.

[29]:
```
ast.save_models("data/astir_summary.hdf5")
```

The hierarchy of the hdf5 file would be:

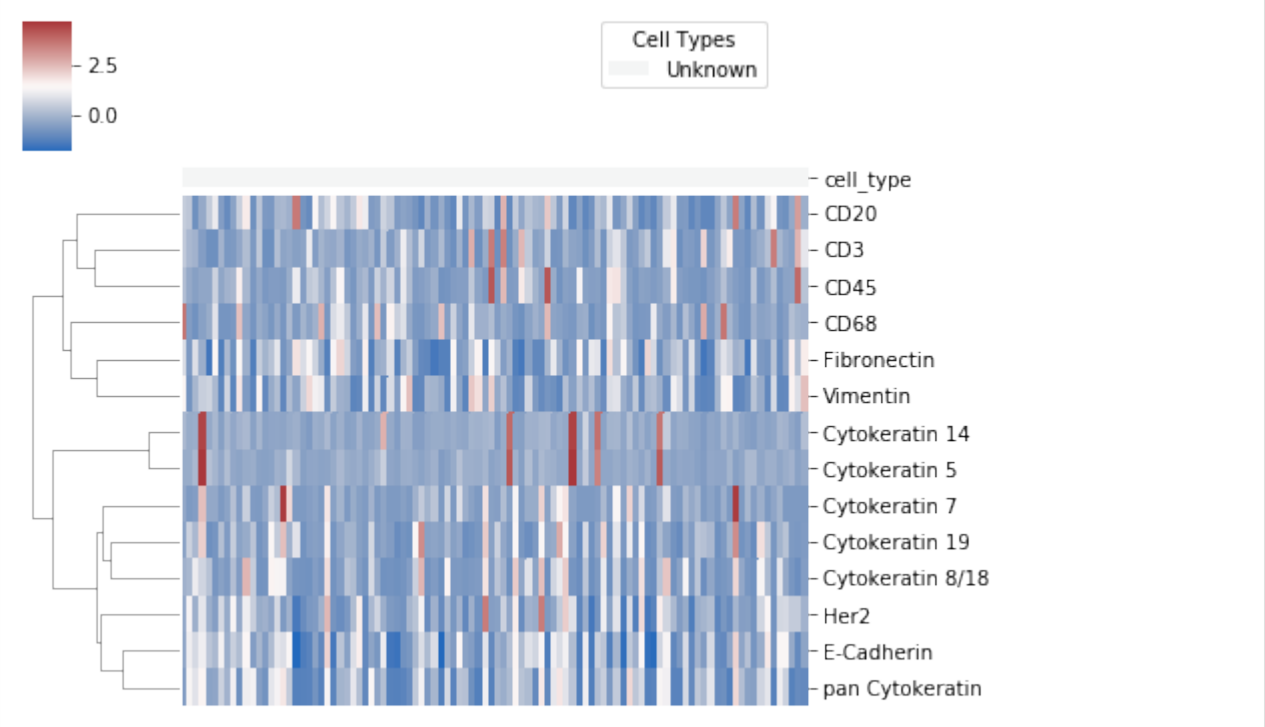


Only the model that is trained will be saved (`CellTypeModel` or `CellStateModel` or both). If the functioned is called before any model is trained, exception will be raised. Data saved in the file is either `int` or `np.array`.

7. Plot clustermap of expression data

After fixing the cell type model, we can also plot a heatmap of protein expression of cells clustered by type. The heatmap will be saved at the location `plot_name`, which is default to `"./celltype_protein_cluster.png"`

[30]:
```
ast.type_clustermap(plot_name="./img/celltype_protein_cluster.png", threshold = 0.7,␣
↪figsize=(7, 5))
```



Note: `threshold` is the probability threshold above which a cell is assigned to a cell type, default to 0.7.

8. Hierarchical model specification

In the marker yaml file, the user can also add a section called `hierarchy`, which specifies the hierarchical structure of cell types. Here's an example:

```
hierarchy:
    epithelial_cells:
        - epithelial(luminal)
        - epithelial(basal)
    immune cells:
        non-lymphocytes:
            - macrophage
        lymphocytes:
            - T cells
            - B cells
```

Some notes: 1. The section would be accessed by key `hierarchy`. 2. In the section, the higher-levelled cell type names should be the keys. 3. The values in the section should also exist as the cell type names in the `cell_types` section. (e.g. if we have `"B cells"` in `marker["hierarchy"]["immune"]`, we should also be able to get `marker["cell_types"]["B cells"]`) 4. In terms of depth in the example: - depth=1: assign to epithelial_cells or immune_cells - depth=2: assign to epithelial(luminal), epithelial(basal), non-lymphocyte and lymphocyte - depth=3: assign to epithelial(luminal), epithelial(basal), macrophage, T cells and B cells

This section could be used to summarize the cell types assignment at a higher hierarchical level. (e.g. a cell is predicted as "immune" instead of "B cells" or "T cells")

[31]:
```
hierarchy_probs = ast.assign_celltype_hierarchy(depth = 1)
hierarchy_probs.head()
```

[31]:
```
                            epithelial_cells  immune cells
BaselTMA_SP41_186_X5Y4_3679          0.318990      0.388475
BaselTMA_SP41_153_X7Y5_246           0.315430      0.473283
BaselTMA_SP41_20_X12Y5_197           0.300328      0.453717
BaselTMA_SP41_14_X1Y8_84             0.291041      0.444866
BaselTMA_SP41_166_X15Y4_266          0.283482      0.463366
```

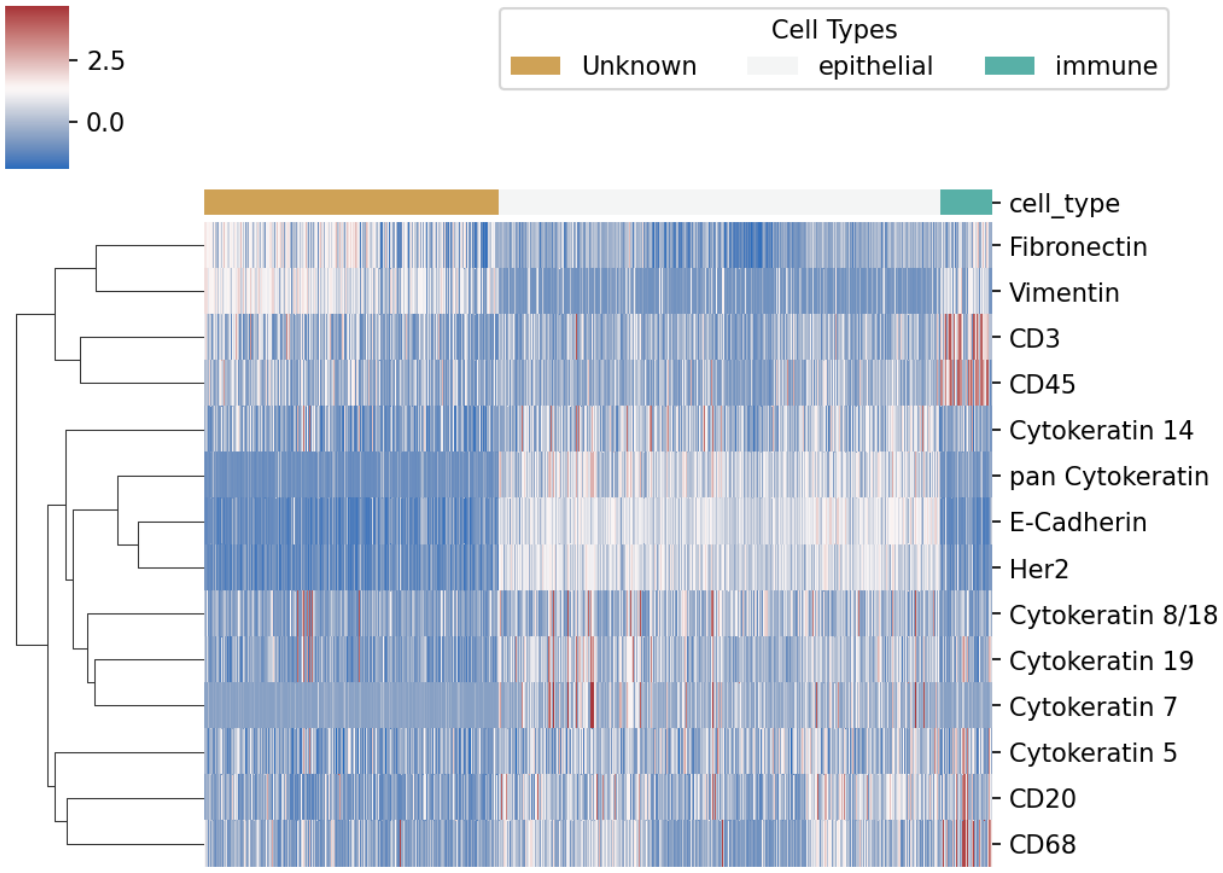
[32]:
```
hierarchy_probs = ast.assign_celltype_hierarchy(depth = 2)
hierarchy_probs.head()
```

[32]:
```
                            epithelial(luminal)  epithelial(basal)  \
BaselTMA_SP41_186_X5Y4_3679             0.119610           0.199379
BaselTMA_SP41_153_X7Y5_246              0.114491           0.200939
BaselTMA_SP41_20_X12Y5_197              0.126138           0.174190
BaselTMA_SP41_14_X1Y8_84                0.140125           0.150916
BaselTMA_SP41_166_X15Y4_266             0.125728           0.157753

                            non-lymphocytes  lymphocytes
BaselTMA_SP41_186_X5Y4_3679         0.105010     0.283465
BaselTMA_SP41_153_X7Y5_246          0.122539     0.350744
BaselTMA_SP41_20_X12Y5_197          0.135788     0.317929
BaselTMA_SP41_14_X1Y8_84            0.103741     0.341125
BaselTMA_SP41_166_X15Y4_266         0.135476     0.327890
```

[33]:
```
hierarchy_probs = ast.assign_celltype_hierarchy(depth = 3)
hierarchy_probs.head()
```

[33]:
```
                            epithelial(luminal)  epithelial(basal)  \
BaselTMA_SP41_186_X5Y4_3679             0.119610           0.199379
BaselTMA_SP41_153_X7Y5_246              0.114491           0.200939
BaselTMA_SP41_20_X12Y5_197              0.126138           0.174190
BaselTMA_SP41_14_X1Y8_84                0.140125           0.150916
BaselTMA_SP41_166_X15Y4_266             0.125728           0.157753

                            macrophage   T cells   B cells
BaselTMA_SP41_186_X5Y4_3679    0.105010  0.198811  0.084654
BaselTMA_SP41_153_X7Y5_246     0.122539  0.193806  0.156938
BaselTMA_SP41_20_X12Y5_197     0.135788  0.210399  0.107530
BaselTMA_SP41_14_X1Y8_84       0.103741  0.221191  0.119934
BaselTMA_SP41_166_X15Y4_266    0.135476  0.221755  0.106135
```

To make it more clear, here's a heatmap for the cell assignment in a higher hierarchy:



The way it is calculated is simply summing up the probabilities of the cell type assignments under the same hierarchy.

9. Using astir as command line tool

`astir` could also be used as a command line tool with `csv` input. Here are some example.

To fit cell types on the sample `csv` file and marker with a design matrix, setting `n_init` to 3 and `batch_size` to 128:

[44]:
```
!astir type ../../../tests/test-data/test_data.csv ../../../tests/test-data/jackson-2020-
↪markers.yml data/test_data_type_assignments.csv --design ../../../tests/test-data/
↪design.csv --n_init 3 --batch_size 128
```

```
training restart 1/3: 100%|| 5/5 [164.98epochs/s, current loss: -25.6]
training restart 2/3: 100%|| 5/5 [181.48epochs/s, current loss: -33.4]
training restart 3/3: 100%|| 5/5 [177.48epochs/s, current loss: -12.5]
training restart (final): 100%|| 50/50 [198.48epochs/s, current loss: -591.9]
/Users/jinelles.h/Documents/Camlab/astir-top-level/astir/astir/astir.py:178: UserWarning:
↪ Maximum epochs reached. More iteration may be needed to complete the training.
  warnings.warn(msg)
```

To fit cell states on the sample csv file and marker, setting `learning rate` to `5e-4`, `dropout_rate` to `0.2` and `batch_norm` to `True`:

[43]:
```
!astir state ../../../tests/test-data/test_data.csv ../../../tests/test-data/jackson-
↪2020-markers.yml data/test_data_type_assignments.csv --design ../../../tests/test-data/
↪design.csv --learning_rate 5e-4 --dropout_rate 0.2 --batch_norm True
```

(continues on next page)

(continued from previous page)

```
training restart 1/3: 100%|| 5/5 [142.02epochs/s, current loss: 32237.3]
training restart 2/3: 100%|| 5/5 [226.10epochs/s, current loss: 963.2]
training restart 3/3: 100%|| 5/5 [230.45epochs/s, current loss: 147046.4]
training restart (final):   0%|          | 0/50 [?epochs/s, current loss: 883.6]
```

Moreover, astir could also be used as a converter which converts `rds` file with `SingleCellExperiment` to `csv` file. see more details

Run `astir -h`, `astir type -h`, `astir state -h` and `astir convert -h` in the command line for more details.

[33]:

2.2.2 Loading data into astir

Table of Contents:

0. Loading necessary libraries and define paths

[1]:
```
# !pip install -e ../../..
import os
import sys

module_path = os.path.abspath(os.path.join('../../..'))
if module_path not in sys.path:
    sys.path.append(module_path)
module_path = os.path.abspath(os.path.join('../../../astir'))
if module_path not in sys.path:
    sys.path.append(module_path)
print(sys.path)
```

(continues on next page)

(continued from previous page)

```
%load_ext autoreload
%autoreload 2
```

```
['/Users/jinelles.h/Documents/Camlab/astir-top-level/astir/docs/tutorials/notebooks', '/
↪Users/jinelles.h/opt/anaconda3/lib/python37.zip', '/Users/jinelles.h/opt/anaconda3/lib/
↪python3.7', '/Users/jinelles.h/opt/anaconda3/lib/python3.7/lib-dynload', '', '/Users/
↪jinelles.h/.local/lib/python3.7/site-packages', '/Users/jinelles.h/opt/anaconda3/lib/
↪python3.7/site-packages', '/Users/jinelles.h/opt/anaconda3/lib/python3.7/site-packages/
↪aeosa', '/Users/jinelles.h/Documents/Camlab/astir-top-level/astir', '/Users/jinelles.h/
↪Documents/Camlab/astir-top-level/margo', '/Users/jinelles.h/opt/anaconda3/lib/python3.
↪7/site-packages/IPython/extensions', '/Users/jinelles.h/.ipython', '/Users/jinelles.h/
↪Documents/Camlab/astir-top-level/astir/astir']
```

[2]:
```
import yaml
import pandas as pd
import astir as ast
import numpy as np
import torch
```

[3]:
```
yaml_marker_path = "../../../tests/test-data/jackson-2020-markers.yml"
design_mat_path = "../../../tests/test-data/design.csv"
expression_mat_path = "../../../tests/test-data/test_data.csv"
expression_dir_path = "../../../tests/test-data/test-dir-read"
expression_loom_path = "../../../tests/test-data/basel_100.loom"
expression_anndata_path="../../../tests/test-data/adata_small.h5ad"
```

1. Starting Astir within python

The input dataset should represent protein expression in single cells. The rows should represent each cell (one row per cell) and the columns should represent each protein (one column per protein). A marker which maps the features (proteins) to cell type/state may is required. A design matrix is optional. If provided, it should be either `np.array` or `pd.DataFrame`.

The initialization of `Astir` requires input dataset `input_expr` as one of `pd.DataFrame`, `Tuple[np.array, List[str], List[str]]` and `Tuple[SCDataset, SCDataset]`.

Note: `dtype` and `random_seed` are always customizable. `dtype` is default to `torch.float64` and `random_seed` is default to `1234`.

1.0 Loading marker dictionary and design matrix

Marker Dictionary

[4]:
```
with open(yaml_marker_path, "r") as stream:
    marker_dict = yaml.safe_load(stream)
print(marker_dict)
```

```
{'cell_states': {'RTK_signalling': ['Her2', 'EGFR'], 'proliferation': ['Ki-67', 'phospho␣
↪Histone'], 'mTOR_signalling': ['phospho mTOR', 'phospho S6'], 'apoptosis': ['cleaved␣
↪PARP', 'Cleaved Caspase3']}, 'cell_types': {'stromal': ['Vimentin', 'Fibronectin'], 'B␣
↪cells': ['CD45', 'CD20'], 'T cells': ['CD45', 'CD3'], 'macrophage': ['CD45', 'CD68'],
↪'epithelial(basal)': ['E-Cadherin', 'pan Cytokeratin', 'Cytokeratin 5', 'Cytokeratin 14
↪', 'Her2'], 'epithelial(luminal)': ['E-Cadherin', 'pan Cytokeratin', 'Cytokeratin 7',
↪'Cytokeratin 8/18', 'Cytokeratin 19', 'Cytokeratin 5', 'Her2']}, 'hierarchy': {
↪'epithelial_cells': ['epithelial(luminal)', 'epithelial(basal)'], 'immune cells': {
↪'non-lymphocytes': ['macrophage'], 'lymphocytes': ['T cells', 'B cells']}}}
```

(continued from previous page)

(continued from previous page)

Some notes:

1. The marker `marker_dict` is not required when `input_expr` is `Tuple[SCDataset, SCDataset]`. Otherwise, it is required to be `Dict[str, Dict[str, str]]`.
2. The outer dictionary may have at most three keys: `cell_type`, `cell_state` and `hierarchy`. `cell_type` and `cell_state` maps to the corresponding dictionary which maps the name of cell type/state to protein features. `hierarchy` maps to the dictionary which indicates the cell type hierarchy.
3. If the user is only intended to classify one of cell type and cell state, only the intended marker dictionary should be provided. So that marker_dict is one of `{"cell_state": {...}}`, `{"cell_type": {...}}` and `{"cell_type": {...}, "cell_state": {...}}`.
4. The `hierarchy` dictionary should be included when the client tends to call `Astir.assign_celltype_hierarchy()`

Design matrix:

Note that the design matrix must have the same number of rows as there are number of cells.

[5]:
```
design_df = pd.read_csv(design_mat_path, index_col=0)
print(design_df.shape)
```

```
(49, 40)
```

Note: `design` is not necessary when `input_expr` is `Tuple[SCDataset, SCDataset]`. Otherwise it is optional.

1.1 Loading data as `pd.DataFrame`

When the input is `pd.DataFrame`, its row and column should respectively represent the cells and the features (proteins).

[6]:
```
df_expr = pd.read_csv(expression_mat_path, index_col=0)
df_expr.head()
```

[6]:

	EGFR	Ruthenium_1	Ruthenium_2	Ruthenium_3	Ruthenium_4	Ruthenium_5	Ruthenium_6	Ruthenium_7	E-Cadherin	DNA1	...	CD45	CD68
BaselTMA_SP41_126_X14Y7_1	0.281753	1.319588	0.597380	1.782863	1.757824	1.991857	2.580564	2.287167	1.814309	2.261638	...	0.044733	0.184805
BaselTMA_SP41_126_X14Y7_2	0.303016	1.319588	0.597380	1.782863	1.757824	1.991857	2.580564	2.287167	1.517685	1.613060	...	0.046802	0.080406
BaselTMA_SP41_126_X14Y7_3	0.252374	1.319588	0.597380	1.782863	1.757824	1.991857	2.580564	2.287167	1.246433	2.138744	...	0.028499	0.203248
BaselTMA_SP41_126_X14Y7_4	0.397732	1.306852	0.534496	1.678217	1.757824	1.961430	2.528551	2.183814	1.246433	2.138744	...	0.028499	0.203248
BaselTMA_SP41_126_X14Y7_5	0.426352	1.173439	0.597380	1.589303	1.389839	1.789887	2.343743	2.123334	1.246433	2.138744	...	0.028499	0.203248

(continues on next page)

(continued from previous page)

```
BaselTMA_SP41_126_X14Y7_4    1.839785  1.816015  ...  0.069053  0.305200
BaselTMA_SP41_126_X14Y7_5    1.618347  1.355214  ...  0.233777  0.135084

                                CD3  Carbonic Anhydrase IX  Cytokeratin 8/18  \
BaselTMA_SP41_126_X14Y7_1  0.000000               0.928929          0.025526
BaselTMA_SP41_126_X14Y7_2  0.110806               0.752101          0.000000
BaselTMA_SP41_126_X14Y7_3  0.020617               0.740759          0.083311
BaselTMA_SP41_126_X14Y7_4  0.060264               1.095968          0.184603
BaselTMA_SP41_126_X14Y7_5  0.057195               1.427983          0.035371

                           Cytokeratin 7     Twist  phospho Histone  \
BaselTMA_SP41_126_X14Y7_1       0.043423  0.209742         0.137454
BaselTMA_SP41_126_X14Y7_2       0.032056  0.108013         0.048428
BaselTMA_SP41_126_X14Y7_3       0.081503  0.119058         0.063097
BaselTMA_SP41_126_X14Y7_4       0.131531  0.160778         0.090666
BaselTMA_SP41_126_X14Y7_5       0.038448  0.014434         0.127032

                           phospho mTOR  phospho S6
BaselTMA_SP41_126_X14Y7_1      0.572811    0.215508
BaselTMA_SP41_126_X14Y7_2      0.539647    0.655731
BaselTMA_SP41_126_X14Y7_3      0.409735    0.437845
BaselTMA_SP41_126_X14Y7_4      0.305718    0.132236
BaselTMA_SP41_126_X14Y7_5      0.261205    0.157786

[5 rows x 45 columns]
```

[7]:
```
a_df = ast.Astir(input_expr=df_expr, marker_dict=marker_dict, design=design_df)
print(a_df)
```

```
Astir object, 6 cell types, 4 cell states, 49 cells
```

1.2 Loading data as `np.array` or `torch.tensor`

When the input is `Tuple[Union[np.array, torch.tensor]], List[str], List[str]]`, the first element `np.array` or `torch.tensor` is the input dataset, the second element `List[str]` is the title of the columns (the names of proteins) and the third element `List[str]` is the title of the rows (the nameof the cells). The length of the second and third list should be equal to the number of columns and rows of the first array.

[8]:
```
# Load as np.array
np_expr = df_expr.values
features = list(df_expr.columns)
cores = list(df_expr.index)
a_np = ast.Astir(input_expr=(np_expr, features, cores), marker_dict=marker_dict,␣
↪design=design_df)
print(a_np)
```

```
Astir object, 6 cell types, 4 cell states, 49 cells
```

[9]:
```
# Load as torch.tensor
t_expr = torch.from_numpy(np_expr)
a_t = ast.Astir(input_expr=(t_expr, features, cores), marker_dict=marker_dict,␣
↪design=design_df)
```

(continues on next page)

(continued from previous page)

```
print(a_t)
```

```
Astir object, 6 cell types, 4 cell states, 49 cells
```

1.3 Loading data as SCDataset

When the input is Tuple[SCDataset, SCDataset], the first SCDataset should be the cell type dataset and the second SCDataset should be the cell state dataset.

[10]:
```
type_scd = ast.SCDataset(expr_input=df_expr, marker_dict=marker_dict["cell_types"],
                         include_other_column=True, design=design_df)
state_scd = ast.SCDataset(expr_input=df_expr, marker_dict=marker_dict["cell_states"],
                          include_other_column=False, design=design_df)
a_scd = ast.Astir(input_expr=(type_scd, state_scd))
print(a_scd)
```

```
Astir object, 6 cell types, 4 cell states, 49 cells
```

2. Loading from csv and yaml files

A data reader from_csv_yaml for loading csv and yaml file is provided.

The row of the csv file should represent the information of each single cells and the column of the csv file should represent the expression of each protein in different cells.

[11]:
```
a_csv = ast.from_csv_yaml(csv_input=expression_mat_path, marker_yaml=yaml_marker_path,␣
↪design_csv=design_mat_path)
print(a_csv)
```

```
Astir object, 6 cell types, 4 cell states, 49 cells
```

Some notes:

1. The yaml file at yaml_marker_path contains the marker which maps protein features to cell type/state classes. The format should match the description of **marker dictionary*.
2. from_csv_yaml returns an Astir object.
3. dtype and random_seed are also customizable. dtype is default to torch.float64 and random_seed is default to 1234.

[12]:
```
type(a_csv.get_type_dataset().get_exprs())
```

[12]:
```
torch.Tensor
```

[13]:
```
a_csv.get_type_dataset().get_exprs_df().head()
```

[13]:
```
                              CD20       CD3      CD45      CD68  \
BaselTMA_SP41_126_X14Y7_1  0.207884  0.000000  0.044733  0.184805
BaselTMA_SP41_126_X14Y7_2  0.021506  0.110806  0.046802  0.080406
BaselTMA_SP41_126_X14Y7_3  0.008878  0.020617  0.028499  0.203248
BaselTMA_SP41_126_X14Y7_4  0.053027  0.060264  0.069053  0.305200
BaselTMA_SP41_126_X14Y7_5  0.019127  0.057195  0.233777  0.135084
```

(continues on next page)

(continued from previous page)

```
                          Cytokeratin 14  Cytokeratin 19  Cytokeratin 5  \
BaselTMA_SP41_126_X14Y7_1        0.134128        0.079956       0.178350
BaselTMA_SP41_126_X14Y7_2        0.026951        0.066922       0.081147
BaselTMA_SP41_126_X14Y7_3        0.023515        0.186294       0.076112
BaselTMA_SP41_126_X14Y7_4        0.114420        0.346273       0.164059
BaselTMA_SP41_126_X14Y7_5        0.055368        0.124407       0.095323

                          Cytokeratin 7  Cytokeratin 8/18  E-Cadherin  \
BaselTMA_SP41_126_X14Y7_1       0.043423          0.025526    1.814309
BaselTMA_SP41_126_X14Y7_2       0.032056          0.000000    1.517685
BaselTMA_SP41_126_X14Y7_3       0.081503          0.083311    1.246433
BaselTMA_SP41_126_X14Y7_4       0.131531          0.184603    1.839785
BaselTMA_SP41_126_X14Y7_5       0.038448          0.035371    1.618347

                          Fibronectin      Her2  Vimentin  pan Cytokeratin
BaselTMA_SP41_126_X14Y7_1    1.039734  0.483007  0.444140         1.187512
BaselTMA_SP41_126_X14Y7_2    1.147644  0.513386  0.270070         0.749379
BaselTMA_SP41_126_X14Y7_3    0.988906  0.633226  0.233909         1.216521
BaselTMA_SP41_126_X14Y7_4    0.842710  0.709272  0.542362         1.354303
BaselTMA_SP41_126_X14Y7_5    1.073357  0.482230  0.759944         0.629398
```

3. Loading from a directory of csvs and yaml

The user can also load the data from a directory of `csv` files with `from_csv_dir_yaml`.

In this case, every `csv` file should represent the expression data from different samples. A design matrix will be generated automatically.

[14]:
```
a_dir = ast.from_csv_dir_yaml(input_dir=expression_dir_path, marker_yaml=yaml_marker_
↪path)
print(a_dir)
```

```
Astir object, 6 cell types, 4 cell states, 40 cells
```

Some notes:

1. The yaml file at `yaml_marker_path` contains the marker which maps protein features to cell type/state classes. The format should match the description of **marker dictionary*.
2. `from_csv_dir_yaml` returns an Astir object.
3. `dtype` and `random_seed` are also customizable. `dtype` is default to `torch.float64` and `random_seed` is default to `1234`.

[15]:
```
type(a_dir.get_type_dataset().get_exprs())
```

[15]:
```
torch.Tensor
```

[16]:
```
a_dir.get_type_dataset().get_exprs_df().head()
```

[16]:
```
                             CD20       CD3      CD45      CD68  \
BaselTMA_SP41_126_X14Y7_1  0.207884  0.000000  0.044733  0.184805
BaselTMA_SP41_126_X14Y7_2  0.021506  0.110806  0.046802  0.080406
BaselTMA_SP41_126_X14Y7_3  0.008878  0.020617  0.028499  0.203248
```

(continues on next page)

(continued from previous page)

```
BaselTMA_SP41_126_X14Y7_4  0.053027  0.060264  0.069053  0.305200
BaselTMA_SP41_126_X14Y7_5  0.019127  0.057195  0.233777  0.135084

                           Cytokeratin 14  Cytokeratin 19  Cytokeratin 5  \
BaselTMA_SP41_126_X14Y7_1         0.134128        0.079956       0.178350
BaselTMA_SP41_126_X14Y7_2         0.026951        0.066922       0.081147
BaselTMA_SP41_126_X14Y7_3         0.023515        0.186294       0.076112
BaselTMA_SP41_126_X14Y7_4         0.114420        0.346273       0.164059
BaselTMA_SP41_126_X14Y7_5         0.055368        0.124407       0.095323

                           Cytokeratin 7  Cytokeratin 8/18  E-Cadherin  \
BaselTMA_SP41_126_X14Y7_1       0.043423          0.025526    1.814309
BaselTMA_SP41_126_X14Y7_2       0.032056          0.000000    1.517685
BaselTMA_SP41_126_X14Y7_3       0.081503          0.083311    1.246433
BaselTMA_SP41_126_X14Y7_4       0.131531          0.184603    1.839785
BaselTMA_SP41_126_X14Y7_5       0.038448          0.035371    1.618347

                           Fibronectin      Her2  Vimentin  pan Cytokeratin
BaselTMA_SP41_126_X14Y7_1     1.039734  0.483007  0.444140         1.187512
BaselTMA_SP41_126_X14Y7_2     1.147644  0.513386  0.270070         0.749379
BaselTMA_SP41_126_X14Y7_3     0.988906  0.633226  0.233909         1.216521
BaselTMA_SP41_126_X14Y7_4     0.842710  0.709272  0.542362         1.354303
BaselTMA_SP41_126_X14Y7_5     1.073357  0.482230  0.759944         0.629398
```

4. Loading from loom

It is also possible to load the data from a `loom` file with `from_loompy_yaml`.

[17]:
```
a_loom = ast.from_loompy_yaml(loom_file=expression_loom_path, marker_yaml=yaml_marker_
↪path,
                              protein_name_attr="protein", cell_name_attr="cell_name", batch_
↪name_attr="batch")
print(a_loom)
```

```
Astir object, 6 cell types, 4 cell states, 100 cells
```

Some notes:

1. The protein and cell names are taken from `ds.ra[protein_name_attr]` and `ds.ca[cell_name_attr]` respectively if specified, and `ds.ra["protein"]` and `ds.cs["cell_name"]` otherwise.
2. If `batch_name` is sepecified, the corresponding column of `ds.ca[batch_name_attr]` will be assumed as the batch variable and turned into a design matrix. Otherwise it is taken as `ds.ca["batch"]`
3. The yaml file at `yaml_marker_path` contains the marker which maps protein features to cell type/state classes. The format should match the description of **marker dictionary*.
4. `from_loom_yaml` returns an Astir object.
5. `dtype` and `random_seed` are also customizable. `dtype` is default to `torch.float64` and `random_seed` is default to `1234`.

[18]:
```
type(a_loom.get_type_dataset().get_exprs())
```

[18]:
```
torch.Tensor
```

[19]:
```
a_loom.get_type_dataset().get_exprs_df()
```

[19]:

	CD20	CD3	CD45	CD68	\
BaselTMA_SP41_44_X2Y7_726	0.080173	0.010469	0.013425	0.149373	
BaselTMA_SP41_Liver_X2Y1_1909	0.051691	0.139216	0.061985	0.140193	
BaselTMA_SP41_231_X6Y6_10_798	0.000000	0.078386	0.144959	0.570016	
BaselTMA_SP41_141_X11Y2_4968	0.039043	0.028426	0.089894	0.089386	
BaselTMA_SP41_141_X11Y2_746	0.079079	0.184354	1.174959	0.297893	
...	...	...	...	...	
BaselTMA_SP42_25_X3Y2_1178	0.110930	0.022230	0.031842	0.076643	
BaselTMA_SP43_272_X11Y3_460	0.057730	0.124963	0.000000	0.000000	
BaselTMA_SP42_192_X8Y5_2214	0.053447	0.080335	0.031003	0.050570	
BaselTMA_SP41_203_X8Y8_2433	0.028729	0.030617	0.322813	1.180702	
BaselTMA_SP41_249_X3Y9_996	0.228180	0.117249	0.509954	1.180713	

	Cytokeratin 14	Cytokeratin 19	Cytokeratin 5	\
BaselTMA_SP41_44_X2Y7_726	0.128329	1.577395	0.210581	
BaselTMA_SP41_Liver_X2Y1_1909	0.208142	0.129807	0.117969	
BaselTMA_SP41_231_X6Y6_10_798	0.158847	0.325296	0.129998	
BaselTMA_SP41_141_X11Y2_4968	0.075023	0.294802	0.130921	
BaselTMA_SP41_141_X11Y2_746	0.039844	0.177649	0.129131	
...	...	...	...	
BaselTMA_SP42_25_X3Y2_1178	0.128341	0.845475	0.036395	
BaselTMA_SP43_272_X11Y3_460	0.000000	0.137121	0.201029	
BaselTMA_SP42_192_X8Y5_2214	0.095160	0.605501	0.156250	
BaselTMA_SP41_203_X8Y8_2433	0.081800	0.182332	0.083335	
BaselTMA_SP41_249_X3Y9_996	0.167626	0.107251	0.116069	

	Cytokeratin 7	Cytokeratin 8/18	E-Cadherin	\
BaselTMA_SP41_44_X2Y7_726	0.661001	1.777394	2.028177	
BaselTMA_SP41_Liver_X2Y1_1909	0.000000	1.211757	0.642365	
BaselTMA_SP41_231_X6Y6_10_798	0.000000	0.166604	0.699211	
BaselTMA_SP41_141_X11Y2_4968	0.105543	0.721960	1.462543	
BaselTMA_SP41_141_X11Y2_746	0.056974	0.017406	0.391993	
...	...	...	...	
BaselTMA_SP42_25_X3Y2_1178	0.143434	1.005111	1.917632	
BaselTMA_SP43_272_X11Y3_460	0.000000	0.075655	0.804087	
BaselTMA_SP42_192_X8Y5_2214	0.896955	1.221076	1.452352	
BaselTMA_SP41_203_X8Y8_2433	0.037738	0.212589	1.547722	
BaselTMA_SP41_249_X3Y9_996	0.026734	0.265320	0.502906	

	Fibronectin	Her2	Vimentin	\
BaselTMA_SP41_44_X2Y7_726	1.606446	0.803987	0.279106	
BaselTMA_SP41_Liver_X2Y1_1909	0.943650	1.488154	0.000000	
BaselTMA_SP41_231_X6Y6_10_798	2.967311	0.891388	1.845164	
BaselTMA_SP41_141_X11Y2_4968	0.607401	0.847732	0.032395	
BaselTMA_SP41_141_X11Y2_746	1.529043	1.196052	0.816324	
...	...	...	...	
BaselTMA_SP42_25_X3Y2_1178	0.455979	0.994131	0.314468	
BaselTMA_SP43_272_X11Y3_460	2.755348	0.286016	1.194052	
BaselTMA_SP42_192_X8Y5_2214	0.302654	2.217377	0.687913	

(continues on next page)

(continued from previous page)

```
BaselTMA_SP41_203_X8Y8_2433       2.593907  0.265961  0.074016
BaselTMA_SP41_249_X3Y9_996        1.608899  0.579117  1.673233

                                pan Cytokeratin
BaselTMA_SP41_44_X2Y7_726                4.026491
BaselTMA_SP41_Liver_X2Y1_1909            0.743843
BaselTMA_SP41_231_X6Y6_10_798            0.137033
BaselTMA_SP41_141_X11Y2_4968             2.527473
BaselTMA_SP41_141_X11Y2_746              0.145410
...                                           ...
BaselTMA_SP42_25_X3Y2_1178               2.820787
BaselTMA_SP43_272_X11Y3_460              0.000000
BaselTMA_SP42_192_X8Y5_2214              2.132569
BaselTMA_SP41_203_X8Y8_2433              1.381932
BaselTMA_SP41_249_X3Y9_996               1.438507

[100 rows x 14 columns]
```

5. Loading from anndata

We can read in data from the AnnData format, along with a `yaml` file containing marker information using the `from_anndata_yaml` function. We have temporarily disabled this example while the anndata format is standardized:

[1]:
```
if False:
    a_ann = ast.from_anndata_yaml(anndata_file=expression_anndata_path, marker_yaml=yaml_
↪marker_path,
                                  protein_name="protein",cell_name="cell_name", batch_name=
↪"batch")
    print(a_ann)
```

Some notes:

1. The protein and cell names are taken from `adata.var[protein_name]` and `adata.obs[cell_name]` respectively if specified, and `adata.var_names` and `adata.obs_names` otherwise.
2. If `batch_name` is sepecified, the corresponding column of `adata.var` will be assumed as the batch variable and turned into a design matrix.
3. The yaml file at `yaml_marker_path` contains the marker which maps protein features to cell type/state classes. The format should match the description of **marker dictionary*.
4. `from_anndata_yaml` returns an Astir object.
5. `dtype` and `random_seed` are also customizable. `dtype` is default to `torch.float64` and `random_seed` is default to `1234`.

[21]:
```
if False:
    type(a_ann.get_type_dataset().get_exprs())
```

[21]:
```
torch.Tensor
```

[22]:
```
if False:
    a_ann.get_type_dataset().get_exprs_df()
```

[22]:

	CD20	CD3	CD45	CD68	\
ZTMA208_slide_11_By5x8_1	0.168521	0.090277	0.271871	0.412439	
ZTMA208_slide_11_By5x8_2	0.366301	0.352614	0.284034	0.312862	
ZTMA208_slide_11_By5x8_3	0.177006	0.103808	0.150791	0.122472	
ZTMA208_slide_11_By5x8_4	0.304068	0.222802	0.219736	0.277622	
ZTMA208_slide_11_By5x8_5	0.137789	0.130010	0.105604	1.035280	
ZTMA208_slide_11_By5x8_6	0.182926	0.169596	0.270698	0.257178	
ZTMA208_slide_11_By5x8_7	0.239257	0.149007	0.351788	0.138080	
ZTMA208_slide_11_By5x8_8	0.175299	0.153332	0.215698	0.104709	
ZTMA208_slide_11_By5x8_9	0.210541	0.118273	0.146135	0.148164	
ZTMA208_slide_11_By5x8_10	0.308899	0.326121	0.224866	0.276182	

	Cytokeratin 14	Cytokeratin 19	Cytokeratin 5	\
ZTMA208_slide_11_By5x8_1	0.087354	0.155710	0.100308	
ZTMA208_slide_11_By5x8_2	0.152354	0.508728	0.028651	
ZTMA208_slide_11_By5x8_3	0.292241	0.634366	0.090457	
ZTMA208_slide_11_By5x8_4	0.373870	2.212514	0.304824	
ZTMA208_slide_11_By5x8_5	0.212105	0.144144	0.074692	
ZTMA208_slide_11_By5x8_6	0.224863	1.143546	0.189600	
ZTMA208_slide_11_By5x8_7	0.142505	1.415104	0.124484	
ZTMA208_slide_11_By5x8_8	0.237387	2.190369	0.264600	
ZTMA208_slide_11_By5x8_9	0.362226	1.267224	0.173477	
ZTMA208_slide_11_By5x8_10	0.140240	2.032473	0.334358	

	Cytokeratin 7	Cytokeratin 8/18	E-Cadherin	\
ZTMA208_slide_11_By5x8_1	0.000000	0.096674	0.974271	
ZTMA208_slide_11_By5x8_2	0.029904	0.749755	2.787740	
ZTMA208_slide_11_By5x8_3	0.056627	0.446911	1.927940	
ZTMA208_slide_11_By5x8_4	0.000000	1.904837	3.175959	
ZTMA208_slide_11_By5x8_5	0.000000	0.000000	1.900182	
ZTMA208_slide_11_By5x8_6	0.001542	0.650384	2.580153	
ZTMA208_slide_11_By5x8_7	0.001245	1.091975	2.696699	
ZTMA208_slide_11_By5x8_8	0.000000	1.457901	2.788996	
ZTMA208_slide_11_By5x8_9	0.000000	0.842407	2.950440	
ZTMA208_slide_11_By5x8_10	0.000000	1.503531	2.938590	

	Fibronectin	Her2	Vimentin	pan Cytokeratin
ZTMA208_slide_11_By5x8_1	2.867470	0.552905	2.335253	1.361075
ZTMA208_slide_11_By5x8_2	2.174494	1.046198	0.285699	2.454543
ZTMA208_slide_11_By5x8_3	2.997043	1.020517	2.887193	2.590460
ZTMA208_slide_11_By5x8_4	1.598163	2.269974	0.877098	4.250308
ZTMA208_slide_11_By5x8_5	2.326346	0.610897	2.882146	0.275225
ZTMA208_slide_11_By5x8_6	1.891692	1.724237	1.931947	2.994441
ZTMA208_slide_11_By5x8_7	1.994174	1.796137	0.127125	3.523499
ZTMA208_slide_11_By5x8_8	1.859896	1.726696	0.106661	4.245234
ZTMA208_slide_11_By5x8_9	1.852758	2.183716	0.957369	3.098247
ZTMA208_slide_11_By5x8_10	2.192502	2.312838	1.337983	4.199266

6. Loading model from hdf5

The whole model could be saved with `Astir.save_model(<hdf5_name>)`, and the saved `hdf5` file could be used to load a new model. This makes it possible to access the previously trained parameters and assignments without having to train the model again.

[23]:
```
# Save a trained model
hdf5_summary = "./data/a_summary.hdf5"
a_orig = ast.Astir(input_expr=df_expr, marker_dict=marker_dict, design=design_df)
a_orig.fit_type(max_epochs=5, n_init=1, n_init_epochs=1)
a_orig.fit_type(max_epochs=5, n_init=1, n_init_epochs=1)
a_orig.save_models(hdf5_summary)
```

```
training restart 1/1: 100%|| 1/1 [68.35epochs/s, current loss: 286.9]
training restart (final): 100%|| 5/5 [90.87epochs/s, current loss: 229.9]
/Users/jinelles.h/Documents/Camlab/astir-top-level/astir/astir/astir.py:178: UserWarning:
↪ Maximum epochs reached. More iteration may be needed to complete the training.
  warnings.warn(msg)
training restart 1/1: 100%|| 1/1 [123.53epochs/s, current loss: 286.9]
training restart (final): 100%|| 5/5 [91.99epochs/s, current loss: 229.9]
```

[24]:
```
# Load a trained model
a_load = ast.Astir()
a_load.load_model(hdf5_summary)
print(a_load)
```

```
Astir object
```

Some notes: 1. In the example, we didn't load any dataset, which means we can't get access to the original dataset on which the model was trained. 2. The new model could still be used to predict cell type/status, etc. 3. If the user want to do further operation which requires access to the original dataset, simply load it again when initializing: `a_load = ast.Astir(input_expr=df_expr, marker_dict=marker_dict, design=design_df)`.

2.3 Development

2.3.1 How to render the docs

Install Sphinx

```
$ pip install sphinx
```

From the main project directory *cd* into docs directory

```
$ cd docs
```

Build the existing reStructuredText files

```
$ make html
```

If the above command causes "Could not import extension <extension-name>" pip install them until the build succeeds.

Open `astir/docs/html/index.html` in your favourite browser either by copying the absolute path in your browser URL bar. If you are using *PyCharm* editor, you can

right click on `index.html` in the file browser -> *Open in Browser* -> select your favourite browser

2.3.2 How to run nosetests and add a test

Running nosetests

Method 1

Run one test module at a time

```
$ nosetests astir/tests/test_astir.py
$ nosetests astir/tests/models/test_cellstate.py
```

Method 2

Run all test modules at once

..code:

```
$ nosetests
```

in any project module directory. You might need install the nose package.

Adding a unittest

2.3.3 Best git practices

The best git practice is to start your own local branch, and commit to your local branch's remote branch once in awhile. Once your branch is ready to merge into the origin master repo, you want to git merge, pull, and push.

Git clone and start a new branch

This is the first step you want to take and won't have to repeat unless you want to clone on another machine or create a new branch.

```
$ git clone https://github.com/camlab-bioml/astir.git
$ git checkout -b <new-branch-name>
```

Update your copy in the repo (git add, commit, push)

You might want to do git commits once in awhile to save your work or create new checkpoint.

```
$ git add <filename1> <filename2> ... <filename n>
$ git commit -m "<your-commmit-message>"
```

Additionally push your work in local branch to its remote branch

```
$ git push origin <my-working-branch-name>
```

or

```
$ git push
```

If you are using the second command make sure that your local branch, called *branch-name*, is pushing to its remote branch, called *origin/branch-name*

Update origin/master (git merge, pull)

To update Master remote branch

First, commit and push all your current work to your remote branch

Second, checkout master

```
$ git checkout master
```

This changes your working branch to *local master*.

You can view your current working branch with the following command

```
$ git branch
```

```
$ git merge <branch-to-merge-current-with>
```

Resolve any merge conflicts you get. Once the merge is complete and all conflicts are resolved

Update the local master branch by

```
$ git pull origin master
```

or depending on your setup you may even be able to run

```
$ git pull
```

To merge a branch into the current one Again resolve any conflicts

Update remote master by following the steps outlined in *Update your copy in the repo*

2.4 astir package

2.4.1 Module contents

class astir.**Astir**(*input_expr=None*, *marker_dict=None*, *design=None*, *random_seed=1234*, *dtype=torch.float64*)

Bases: object

Create an Astir object

Parameters

- **input_expr** (*Union[pd.DataFrame, Tuple[np.array, List[str], List[str]], Tuple[SCDataset, SCDataset]]*) – the single cell protein expression dataset
- **marker_dict** (*Dict[str, Dict[str, str]], optional*) – the marker dictionary which maps cell type/state to protein features, defaults to None
- **design** (*Union[pd.DataFrame, np.array], optional*) – the design matrix labeling the grouping of cell, defaults to None
- **random_seed** (*int, optional*) – random seed for parameter initialization, defaults to *1234*
- **dtype** (*torch.dtype, optional*) – dtype of data, defaults to *torch.float64*

Raises **NotClassifiableError** – raised if the model is not trainable

Methods:

assign_celltype_hierarchy([depth])	Get cell type assignment at a specified higher hierarchy according to the hierarchy provided
diagnostics_cellstate()	Run diagnostics on cell state assignments
diagnostics_celltype([threshold, alpha])	Run diagnostics on cell type assignments
fit_state([max_epochs, learning_rate, ...])	Run Variational Bayes to infer cell states
fit_type([max_epochs, learning_rate, ...])	Run Variational Bayes to infer cell types
get_cellstates()	Get cell state activations.
get_celltype_probabilities()	Get the cell assignment probability.
get_celltypes([threshold, assignment_type])	Get the most likely cell type
get_hierarchy_dict()	Get the dictionary for cell type hierarchical structure.
get_state_dataset()	Get the *SCDataset* for cell state training.
get_state_losses()	Getter for losses
get_state_model()	Get the trained *CellStateModel*.
get_state_run_info()	Get the run information (i.e. *max_epochs*, *learning_rate*, *batch_size*,
get_type_dataset()	Get the *SCDataset* for cell type training.
get_type_losses()	Get the final losses of the type model.
get_type_model()	Get the trained *CellTypeModel*.
get_type_run_info()	Get the run information (i.e. *max_epochs*, *learning_rate*,
load_model(hdf5_name)	Load model from hdf5 file
normalize([percentile_lower, percentile_upper])	Normalize the expression data
predict_cellstates([dset])	Get the prediction cell state activations on a dataset on an existing model

continues on next page

Table 1 – continued from previous page

predict_celltypes([dset])	Predict the probabilities of different cell type assignments.
save_models([hdf5_name])	Save the summary of this model to an *hdf5* file.
state_to_csv(output_csv)	Writes state assignment output from training state model in csv file
type_clustermap([plot_name, threshold, ...])	Save the heatmap of protein content in cells with cell types labeled.
type_to_csv(output_csv[, threshold, ...])	Save the cell type assignemnt to a *csv* file.

assign_celltype_hierarchy(*depth=1*)

Get cell type assignment at a specified higher hierarchy according to the hierarchy provided in the dictionary.

Parameters **depth** (*int, optional*) – the depth of hierarchy to assign probability to, defaults to 1

Raises **Exception** – raised when the dictionary for hierarchical structure is not provided or the model hasn't been trained.

Returns probability assignment of cell type at a superstructure

Return type pd.DataFrame

diagnostics_cellstate()

Run diagnostics on cell state assignments

This performs a basic test by comparing the correlation values between all marker genes and all non marker genes. It detects where the non marker gene has higher correlation values than the smallest correlation values of marker genes.

1. Get correlations between all cell states and proteins
2. For each cell state *c*, get the smallest correlation with marker *g*
3. **For each cell state *c* and its non marker *g*, find any correlation that is** bigger than those smallest correlation for *c*.
4. **Any *c* and *g* pairs found in step 3 will be included in the output of** *Astir.diagnostics_cellstate()*, including an explanation.

Return type DataFrame

Returns diagnostics

diagnostics_celltype(*threshold=0.5*, *alpha=0.01*)

Run diagnostics on cell type assignments

This performs a basic test that cell types express their markers at higher levels than in other cell types. This function performs the following steps:

1. Iterates through every cell type and every marker for that cell type
2. **Given a cell type *c* and marker *g*, find the set of cell** types *D* that don't have *g* as a marker
3. **For each cell type *d* in *D*, perform a t-test between the** expression of marker *g* in *c* vs *d*
4. **If *g* is not expressed significantly higher (at significance** *alpha*), output a diagnostic explaining this for further investigation.

Parameters

- **threshold** (float) – The threshold at which cell types are assigned (see *get_celltypes*)
- **alpha** (float) – The significance threshold for t-tests for determining over-expression

Return type DataFrame

Returns Either a pd.DataFrame listing cell types whose markers aren't expressed signficantly higher.

fit_state(*max_epochs=50*, *learning_rate=0.001*, *batch_size=128*, *delta_loss=0.001*, *n_init=5*, *n_init_epochs=5*, *delta_loss_batch=10*, *const=2*, *dropout_rate=0*, *batch_norm=False*)

Run Variational Bayes to infer cell states

Parameters

- **max_epochs** (int) – number of epochs, defaults to *100*
- **learning_rate** (float) – the learning rate, defaults to *1e-2*
- **n_init** (int) – the number of initial parameters to compare, defaults to *5*
- **delta_loss** (float) – stops iteration once the loss rate reaches delta_loss, defaults to *0.001*
- **delta_loss_batch** (int) – the batch size to consider delta loss, defaults to *10*

Return type None

fit_type(*max_epochs=50*, *learning_rate=0.001*, *batch_size=128*, *delta_loss=0.001*, *n_init=5*, *n_init_epochs=5*)

Run Variational Bayes to infer cell types

Parameters

- **max_epochs** (int) – maximum number of epochs to train
- **learning_rate** (float) – ADAM optimizer learning rate
- **batch_size** (int) – minibatch size
- **delta_loss** (float) – stops iteration once the loss rate reaches delta_loss, defaults to *0.001*
- **n_inits** – number of random initializations

Return type None

get_cellstates()

Get cell state activations. It returns the rescaled activations, values between 0 and 1

Returns state assignments

Return type pd.DataFrame

get_celltype_probabilities()

Get the cell assignment probability.

Returns *self.assignments*

Return type pd.DataFrame

get_celltypes(*threshold=0.7*, *assignment_type='threshold'*)

Get the most likely cell type

A cell is assigned to a cell type if the probability is greater than threshold. If no cell types have a probability higher than threshold, then "Unknown" is returned

Parameters

- **threshold** (float) – the probability threshold above which a cell is assigned to a cell type
- **assignment_type** (str) – See astir.CellTypeModel.get_celltypes() for full documentation

Return type DataFrame

Returns a data frame with most likely cell types for each

get_hierarchy_dict()
Get the dictionary for cell type hierarchical structure.

Returns *self._hierarchy_dict*

Return type Dict[str, List[str]]

get_state_dataset()
Get the *SCDataset* for cell state training.

Return type *SCDataset*

Returns *self._state_dset*

get_state_losses()
Getter for losses

Returns a numpy array of losses for each training iteration the model runs

Return type np.array

get_state_model()
Get the trained *CellStateModel*.

Raises **Exception** – raised when this function is celled before the model is trained.

Return type *CellStateModel*

Returns *self._state_ast*

get_state_run_info()

Get the run information (i.e. *max_epochs*, *learning_rate*, *batch_size*, *delta_loss*, *n_init*, *n_init_epochs*, *delta_loss_batch*) of the cell state training.

Raises **Exception** – raised when this function is celled before the model is trained.

Return type Dict[str, Union[int, float]]

Returns *self._state_run_info*

get_type_dataset()
Get the *SCDataset* for cell type training.

Return type *SCDataset*

Returns *self._type_dset*

get_type_losses()
Get the final losses of the type model.

Returns *self.losses*

Return type np.array

`get_type_model`()
Get the trained *CellTypeModel*.

Raises **Exception** – raised when this function is called before the model is trained.

Return type *CellTypeModel*

Returns *self._type_ast*

`get_type_run_info`()

Get the run information (i.e. *max_epochs*, *learning_rate*, *batch_size*, *delta_loss*, *n_init*, *n_init_epochs*) of the cell type training.

Raises **Exception** – raised when this function is celled before the model is trained.

Return type Dict[str, Union[int, float]]

Returns *self._type_run_info*

`load_model`(*hdf5_name*)
Load model from hdf5 file

Parameters **hdf5_name** (str) – the full path to file

Return type None

`normalize`(*percentile_lower=1*, *percentile_upper=99*)
Normalize the expression data

This performs a two-step normalization: 1. A *log(1+x)* transformation to the data 2. Winsorizes to (*percentile_lower*, *percentile_upper*)

Parameters

- **percentile_lower** (int) – Lower percentile for winsorization
- **percentile_upper** (int) – Upper percentile for winsorization

Return type None

`predict_cellstates`(*dset=None*)
Get the prediction cell state activations on a dataset on an existing model

Parameters **new_dset** – the dataset to predict cell state activations, default to None

Return type DataFrame

Returns the prediction of cell state activations

`predict_celltypes`(*dset=None*)
Predict the probabilities of different cell type assignments.

Parameters **dset** (*pd.DataFrame, optional*) – the single cell protein expression dataset to predict, defaults to None

Raises **Exception** – when the type model is not trained when this function is called

Returns the probabilities of different cell type assignments

Return type pd.DataFrame

save_models(*hdf5_name='astir_summary.hdf5'*)
Save the summary of this model to an *hdf5* file.

Parameters **hdf5_name** (str) – name of the output *hdf5* file, default to "astir_summary.hdf5"

Raises **Exception** – raised when this function is called before the model is trained.

Return type None

state_to_csv(*output_csv*)
Writes state assignment output from training state model in csv file

Parameters **output_csv** (*str, required*) – path to output csv

Return type None

type_clustermap(*plot_name='celltype_protein_cluster.png'*, *threshold=0.7*, *figsize=(7, 5)*, *prob_assign=None*)
Save the heatmap of protein content in cells with cell types labeled.

Parameters

- **plot_name** (*str, optional*) – name of the plot, extension(e.g. .png or .jpg) is needed, defaults to "celltype_protein_cluster.png"
- **threshold** (*float, optional*) – the probability threshold above which a cell is assigned to a cell type, defaults to 0.7

Return type None

type_to_csv(*output_csv*, *threshold=0.7*, *assignment_type='threshold'*)
Save the cell type assignemnt to a *csv* file.

Parameters

- **output_csv** (*str*) – name for the output .csv file
- **assignment_type** (str) – See astir.CellTypeModel.get_celltypes() for full documentation

Return type None

2.5 astir.data package

2.5.1 Module contents

Classes:

SCDataset(expr_input, marker_dict, ... [, ...])	Container for single-cell proteomic data in the form of a pytorch dataset

Functions:

from_anndata_yaml(anndata_file, marker_yaml)	Create an Astir object from an anndata.Anndata file and a
from_csv_dir_yaml(input_dir, marker_yaml[, ...])	Create an Astir object a directory containing multiple csv files

continues on next page

Table 3 – continued from previous page

from_csv_yaml(csv_input, marker_yaml[, ...])	Create an Astir object from an expression CSV and marker YAML
from_loompy_yaml(loom_file, marker_yaml[, ...])	Create an Astir object from a loom file and a marker yaml

class astir.data.**SCDataset**(*expr_input*, *marker_dict*, *include_other_column*, *design=None*, *dtype=torch.float64*, *device=device(type='cpu')*)

Bases: Generic[torch.utils.data.dataset.T_co]

Container for single-cell proteomic data in the form of a pytorch dataset

Parameters

- **expr_input** (Union[DataFrame, Tuple[Union[array, Tensor], List[str], List[str]]]) – Input expression data. See details :*expr_input* is either a *pd.DataFrame* or a three-element *tuple*. When it is *pd.DataFrame*, its index and column should indicate the cell name and feature name of the dataset; when it is a three-element *tuple*, it should be in the form of *Tuple[Union[np.array, torch.Tensor], List[str], List[str]]* and its first element should be the actual dataset as either *np.array* or *torch.tensor*, the second element should be a list containing the name of the columns or the names of features, the third element should be a list containing the name of the indices or the names of the cells.:
- **marker_dict** (Dict[str, List[str]]) – Marker dictionary containing cell type and information. See details :The dictionary maps the name of cell type/state to protein features. :
- **design** (Union[DataFrame, array, None]) – A design matrix
- **include_other_column** (bool) – Should an additional 'other' column be included?
- **dtype** (dtype) – torch datatype of the model

Methods:

get_cell_names()	Get the cell names.
get_classes()	Get the cell types/states.
get_design()	Get the design matrix.
get_dtype()	Get the dtype of the *SCDataset*.
get_exprs()	Return the expression data as a torch.Tensor.
get_exprs_df()	Return the expression data as a pandas.DataFrame.
get_features()	Get the features (proteins).
get_marker_mat()	Return the marker matrix as a torch.Tensor.
get_mu()	Get the mean expression of each protein as a torch.Tensor.
get_mu_init([n_putative_cells])	Intelligent initialization for mu parameters
get_n_cells()	Get the number of cells: either the number of cell types or cell states.
get_n_classes()	Get the number of 'classes': either the number of cell types or cell states.
get_n_features()	Get the number of features (proteins).
get_sigma()	Get the standard deviation of each protein
normalize([percentile_lower, ...])	Normalize the expression data
rescale()	Normalize the expression data.

get_cell_names()

Get the cell names.

Returns return self._cell_names

Return type List[str]

`get_classes()`
Get the cell types/states.

Returns return self._classes

Return type List[str]

`get_design()`
Get the design matrix.

Returns return self._design

Return type torch.Tensor

`get_dtype()`
Get the dtype of the *SCDataset*.

Returns *self._dtype*

Return type torch.dtype

`get_exprs()`
Return the expression data as a `torch.Tensor`.

Return type `Tensor`

`get_exprs_df()`
Return the expression data as a `pandas.DataFrame`.

Return type `DataFrame`

`get_features()`
Get the features (proteins).

Returns return self._m_features

Return type List[str]

`get_marker_mat()`
Return the marker matrix as a `torch.Tensor`.

Return type `Tensor`

`get_mu()`
Get the mean expression of each protein as a `torch.Tensor`.

Return type `Tensor`

`get_mu_init(`*n_putative_cells=10*`)`
Intelligent initialization for mu parameters

See manuscript for details

Parameters **n_putative_cells** (`int`) – Number of cells to guess as given cell type

Return type `ndarray`

`get_n_cells()`
Get the number of cells: either the number of cell types or cell states.

Return type `int`

`get_n_classes()`
Get the number of 'classes': either the number of cell types or cell states.

Return type `int`

get_n_features()
Get the number of features (proteins).

Return type int

get_sigma()
Get the standard deviation of each protein

Return type Tensor

Returns standard deviation of each protein

normalize(*percentile_lower=0*, *percentile_upper=99.9*, *cofactor=5.0*)
Normalize the expression data

This performs a two-step normalization: 1. A *log(1+x)* transformation to the data 2. Winsorizes to (*percentile_lower*, *percentile_upper*)

Parameters

- **percentile_lower** (float) – the lower bound percentile for winsorization, defaults to 0
- **percentil_upper** – the upper bound percentile for winsorization, defaults to 99.9
- **cofactor** (float) – a cofactor constant, defaults to 5.0

Return type None

rescale()
Normalize the expression data.

Return type None

astir.data.**from_anndata_yaml**(*anndata_file*, *marker_yaml*, *protein_name=None*, *cell_name=None*, *batch_name='batch'*, *create_design_mat=True*, *random_seed=1234*, *dtype=torch.float64*)

Create an Astir object from an anndata.Anndata file and a marker yaml

Parameters

- **anndata_file** (str) – Path to an anndata.Anndata *h5py* file
- **marker_yaml** (str) – Path to input YAML file containing marker gene information. Should include cell_type and cell_state entries. See documention.
- **protein_name** (Optional[str]) – The column of *adata.var* containing protein names. If this is none, defaults to *adata.var_names*
- **cell_name** (Optional[str]) – The column of *adata.obs* containing cell names. If this is none, defaults to *adata.obs_names*
- **batch_name** (str) – The column of *adata.obs* containing batch names. A design matrix will be built using this (if present) using a one-hot encoding to control for batch, defaults to 'batch'
- **create_design_mat** (bool) – Determines whether a design matrix is created. Defaults to True.
- **random_seed** (int) – The random seed to be used to initialize variables, defaults to 1234
- **dtype** (dtype) – datatype of the model parameters, defaults to torch.float64

Return type Any

Returns An object of class *astir_bash.py.Astir* using data imported from the loom files

astir.data.**from_csv_dir_yaml**(*input_dir*, *marker_yaml*, *create_design_mat=True*, *random_seed=1234*, *dtype=torch.float64*)

Create an Astir object a directory containing multiple csv files

Parameters

- **input_dir** (str) – Path to a directory containing multiple CSV files, each in the format expected by *from_csv_yaml*
- **marker_yaml** (str) – Path to input YAML file containing marker gene information. Should include cell_type and cell_state entries. See documention.
- **design_csv** – Path to design matrix as a CSV. Rows should be cells, and columns covariates. First column is cell identifier, and additional column names are covariate identifiers
- **create_design_mat** (bool) – Determines whether a design matrix is created. Defaults to True.
- **random_seed** (int) – The random seed to be used to initialize variables, defaults to 1234
- **dtype** (dtype) – datatype of the model parameters, defaults to torch.float64

Return type Any

astir.data.**from_csv_yaml**(*csv_input*, *marker_yaml*, *design_csv=None*, *create_design_mat=True*, *random_seed=1234*, *dtype=torch.float64*)

Create an Astir object from an expression CSV and marker YAML

Parameters

- **csv_input** (str) – Path to input csv containing expression for cells (rows) by proteins (columns). First column is cell identifier, and additional column names are gene identifiers.
- **marker_yaml** (str) – Path to input YAML file containing marker gene information. Should include cell_type and cell_state entries. See documention.
- **design_csv** (Optional[str]) – Path to design matrix as a CSV. Rows should be cells, and columns covariates. First column is cell identifier, and additional column names are covariate identifiers.
- **create_design_mat** (bool) – Determines whether a design matrix is created. Defaults to True.
- **random_seed** (int) – The random seed to be used to initialize variables, defaults to 1234
- **dtype** (dtype) – datatype of the model parameters, defaults to torch.float64

Return type Any

astir.data.**from_loompy_yaml**(*loom_file*, *marker_yaml*, *protein_name_attr='protein'*, *cell_name_attr='cell_name'*, *batch_name_attr='batch'*, *create_design_mat=True*, *random_seed=1234*, *dtype=torch.float64*)

Create an Astir object from a loom file and a marker yaml

Parameters

- **loom_file** (str) – Path to a loom file, where rows correspond to proteins and columns to cells
- **marker_yaml** (str) – Path to input YAML file containing marker gene information. Should include cell_type and cell_state entries. See documention.

- **protein_name_attr** (str) – The attribute (key) in the row attributes that identifies the protein names (required to match with the marker gene information), defaults to protein
- **cell_name_attr** (str) – The attribute (key) in the column attributes that identifies the name of each cell, defaults to cell_name
- **batch_name_attr** (str) – The attribute (key) in the column attributes that identifies the batch. A design matrix will be built using this (if present) using a one-hot encoding to control for batch, defaults to batch
- **create_design_mat** (bool) – Determines whether a design matrix is created. Defaults to True.
- **random_seed** (int) – The random seed to be used to initialize variables, defaults to 1234
- **dtype** (dtype) – datatype of the model parameters, defaults to torch.float64

Return type Any

Returns An object of class *astir_bash.py.Astir* using data imported from the loom files

2.6 astir.models package

2.6.1 Module contents

Classes:

AstirModel(dset, random_seed, dtype[, device])	Abstract class to perform statistical inference to assign.
CellStateModel([dset, const, dropout_rate, ...])	Class to perform statistical inference to on the activation
CellTypeModel([dset, random_seed, dtype, device])	Class to perform statistical inference to assign cells to cell types.
StateRecognitionNet(C, G[, const, ...])	State Recognition Neural Network to get mean of z and standard deviation of z.
TypeRecognitionNet(C, G[, hidden_size])	Type Recognition Neural Network.

class astir.models.**AstirModel**(*dset*, *random_seed*, *dtype*, *device=device(type='cpu')*)

Bases: object

Abstract class to perform statistical inference to assign. This module is the super class of *CellTypeModel* and *CellStateModel* and is not supposed to be instantiated.

Methods:

fit(max_epochs, learning_rate, batch_size, ...)	Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times
get_assignment()	Get the final assignment of the dataset.
get_data()	Get model data
get_losses()	Getter for losses.
get_scdataset()	Getter for the *SCDataset*.
get_variables()	Returns all variables
is_converged()	Returns True if the model converged

fit(*max_epochs*, *learning_rate*, *batch_size*, *delta_loss*, *delta_loss_batch*, *msg*)

Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs

number of times

Return type None

get_assignment()
Get the final assignment of the dataset.

Return type DataFrame

Returns the final assignment of the dataset

get_data()
Get model data

Return type Dict[str, Tensor]

Returns data

get_losses()
Getter for losses.

Return type Tensor

Returns self.losses

get_scdataset()
Getter for the *SCDataset*.

Return type *SCDataset*

Returns *self._dset*

get_variables()
Returns all variables

Return type Dict[str, Tensor]

Returns self._variables

is_converged()
Returns True if the model converged

Return type bool

Returns self._is_converged

class astir.models.**CellStateModel**(*dset=None*, *const=2*, *dropout_rate=0*, *batch_norm=False*, *random_seed=42*, *dtype=torch.float64*, *device=device(type='cpu')*)
Bases: *astir.models.abstract.AstirModel*

Class to perform statistical inference to on the activation of states (pathways) across cells

Parameters

- **dset** (Optional[*SCDataset*]) – the input gene expression dataset, defaults to None
- **const** (int) – See parameter const in *astir.models.StateRecognitionNet()*, defaults to 2
- **dropout_rate** (float) – See parameter dropout_rate in *astir.models.StateRecognitionNet()*, defaults to 0
- **batch_norm** (bool) – See parameter batch_norm in *astir.models.StateRecognitionNet()*, defaults to False
- **random_seed** (int) – the random seed number to reproduce results, defaults to 42

- **dtype** (dtype) – torch datatype to use in the model, defaults to torch.float64
- **device** (device) – torch.device's cpu or gpu, defaults to torch.device("cpu")

Methods:

diagnostics()	Run diagnostics on cell type assignments
fit([max_epochs, learning_rate, batch_size, ...])	Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times
get_correlations()	Returns a C (# of pathways) X G (# of proteins) matrix where each element represents the correlation value of the pathway and the protein
get_final_mu_z([new_dset])	Returns the mean of the predicted z values for each core
get_recognet()	Getter for the recognition net
load_hdf5(hdf5_name)	Initializes Cell State Model from a hdf5 file type

diagnostics()
Run diagnostics on cell type assignments

See *astir.Astir.diagnostics_cellstate()* for full documentation

Return type DataFrame

fit(*max_epochs=50*, *learning_rate=0.001*, *batch_size=128*, *delta_loss=0.001*, *delta_loss_batch=10*, *msg=''*)
Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times

Parameters

- **max_epochs** (int) – number of train loop iterations, defaults to 50
- **learning_rate** (float) – the learning rate, defaults to 0.01
- **batch_size** (int) – the batch size, defaults to 128
- **delta_loss** (float) – stops iteration once the loss rate reaches delta_loss, defaults to 0.001
- **delta_loss_batch** (int) – the batch size to consider delta loss, defaults to 10
- **msg** (str) – iterator bar message, defaults to empty string

Return type None

get_correlations()
Returns a C (# of pathways) X G (# of proteins) matrix where each element represents the correlation value of the pathway and the protein

Return type array

Returns matrix of correlation between all pathway and protein pairs.

get_final_mu_z(*new_dset=None*)
Returns the mean of the predicted z values for each core

Parameters **new_dset** (Optional[*SCDataset*]) – returns the predicted z values of this dataset on the existing model. If None, it predicts using the existing dataset, defaults to None

Return type Tensor

Returns the mean of the predicted z values for each core

get_recognet()
Getter for the recognition net

Return type *StateRecognitionNet*

Returns the recognition net

load_hdf5(*hdf5_name*)
Initializes Cell State Model from a hdf5 file type

Parameters **hdf5_name** (str) – file path

Return type None

class astir.models.**CellTypeModel**(*dset=None*, *random_seed=1234*, *dtype=torch.float64*, *device=device(type='cpu')*)
Bases: *astir.models.abstract.AstirModel*

Class to perform statistical inference to assign cells to cell types.

Parameters

- **dset** (Optional[*SCDataset*]) – the input gene expression dataframe
- **random_seed** (int) – the random seed for parameter initialization, defaults to 1234
- **dtype** (dtype) – the data type of parameters, should be the same as *dset*, defaults to torch.float64

Methods:

diagnostics(cell_type_assignments, alpha)	Run diagnostics on cell type assignments
fit([max_epochs, learning_rate, batch_size, ...])	Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times
get_celltypes([threshold, assignment_type, ...])	Get the most likely cell types.
get_recognet()	Getter for the recognition net.
load_hdf5(hdf5_name)	Initializes Cell Type Model from a hdf5 file type
plot_clustermap([plot_name, threshold, ...])	Save the heatmap of protein content in cells with cell types labeled.
predict(new_dset)	Feed *new_dset* to the recognition net to get a prediction.

diagnostics(*cell_type_assignments*, *alpha*)
Run diagnostics on cell type assignments

See *astir.Astir.diagnostics_celltype()* for full documentation

Return type DataFrame

fit(*max_epochs=50*, *learning_rate=0.001*, *batch_size=128*, *delta_loss=0.001*, *delta_loss_batch=10*, *msg=''*)
Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times

Parameters

- **max_epochs** (int) – number of train loop iterations, defaults to 50
- **learning_rate** (float) – the learning rate, defaults to 0.01

- **batch_size** (int) – the batch size, defaults to 128
- **delta_loss** (float) – stops iteration once the loss rate reaches delta_loss, defaults to 0.001
- **delta_loss_batch** (int) – the batch size to consider delta loss, defaults to 10
- **msg** (str) – iterator bar message, defaults to empty string

Return type None

get_celltypes(*threshold=0.7*, *assignment_type='threshold'*, *prob_assign=None*)

Get the most likely cell types. A cell is assigned to a cell type if the probability is greater than threshold. If no cell types have a probability higher than threshold, then "Unknown" is returned.

Parameters

- **assignment_type** (str) – either 'threshold' or 'max'. If threshold, type assignment is based on whether the probability threshold is above prob_assignment. If 'max', type assignment is based on the max probability value or "unknown" if there are multiple max probabilities. Defaults to 'threshold'.
- **threshold** (float) – the probability threshold above which a cell is assigned to a cell type, defaults to 0.7

Return type DataFrame

Returns a data frame with most likely cell types for each

get_recognet()

Getter for the recognition net.

Return type *TypeRecognitionNet*

Returns the trained recognition net

load_hdf5(*hdf5_name*)

Initializes Cell Type Model from a hdf5 file type

Parameters **hdf5_name** (str) – file path

Return type None

plot_clustermap(*plot_name='celltype_protein_cluster.png'*, *threshold=0.7*, *figsize=(7.0, 5.0)*, *prob_assign=None*)

Save the heatmap of protein content in cells with cell types labeled.

Parameters

- **plot_name** (str) – name of the plot, extension(e.g. .png or .jpg) is needed, defaults to "celltype_protein_cluster.png"
- **threshold** (float) – the probability threshold above which a cell is assigned to a cell type, defaults to 0.7
- **figsize** (Tuple[float, float]) – the size of the figure, defaults to (7.0, 5.0)

Return type None

predict(*new_dset*)

Feed *new_dset* to the recognition net to get a prediction.

Parameters **new_dset** (DataFrame) – the dataset to be predicted

Return type array

Returns the resulting cell type assignment

class astir.models.**StateRecognitionNet**(*C*, *G*, *const=2*, *dropout_rate=0*, *batch_norm=False*)
Bases: torch.nn.modules.module.Module

State Recognition Neural Network to get mean of z and standard deviation of z. The neural network architecture looks like this: G -> const * C -> const * C -> G (for mu) or -> G (for std). With batch normal layers after each activation output layers and dropout activation units

Parameters

- **C** (int) – the number of pathways
- **G** (int) – the number of proteins
- **const** (int) – the size of the hidden layers are const times proportional to C, defaults to 2
- **dropout_rate** (float) – the dropout rate, defaults to 0
- **batch_norm** (bool) – apply batch normal layers if True, defaults to False

Methods:

forward(x)	One forward pass of the StateRecognitionNet

Attributes:

—

forward(*x*)
One forward pass of the StateRecognitionNet

Parameters **x** (Tensor) – the input to the recognition network model

Return type Tuple[Tensor, Tensor]

Returns the value from the output layer of the network

training: bool

class astir.models.**TypeRecognitionNet**(*C*, *G*, *hidden_size=20*)
Bases: torch.nn.modules.module.Module

Type Recognition Neural Network.

Parameters

- **C** (int) – number of classes
- **G** (int) – number of features
- **hidden_size** (int) – size of hidden layers, defaults to 10

Methods:

forward(x)	One forward pass.

Attributes:

—

forward(*x*)
One forward pass.

Parameters **x** (Tensor) – the input vector

Return type Tensor

Returns the calculated cost value

training: bool

2.7 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

I

L

M

N

P

R

S

T