PicturedRocks Documentation

Release 0.3.1+0.gffaf29f.dirty

Umang Varma, Anna Gilbert

Feb 07, 2019

Contents:

PicturedRocks is a python package that implements information-theoretic feature selection algorithms for scRNA-seq analysis.

CHAPTER 1

Installing

Please ensure you have Python 3.6 or newer and have *numba* and *scikit-learn* installed. The best way to get Python and various dependencies is with Anaconda or Miniconda. Once you have a conda environment, run `conda install numba scikit-learn`. Then use pip to install PicturedRocks and all additional dependencies:

```
pip install picturedrocks
```

To install the latest code from github, clone our github repository. Once inside the project directory, instal by running `pip install -e .`. The `-e` option will point a symbolic link to the current directory instead of installing a copy on your system.

1.1 Feature Selection Tutorial

In this Jupyter notebook, we'll walk through the information-theoretic feature selection algorithms in PicturedRocks.

[1]:
```
import numpy as np
import scanpy.api as sc
import picturedrocks as pr
```

[2]:
```
adata = sc.datasets.paul15()
```

```
WARNING: In Scanpy 0.*, this returned logarithmized data. Now it returns␣
↪non-logarithmized data.
... storing 'paul15_clusters' as categorical
```

[3]:
```
adata
```

[3]:
```
AnnData object with n_obs × n_vars = 2730 × 3451
    obs: 'paul15_clusters'
    uns: 'iroot'
```

The `process_clusts` method copies the cluster column and precomputes various indices, etc. If you have multiple columns that can be used as target labels (e.g., different treatments, clusters via different clustering algorithms or parameters, or demographics), this sets and processes the given columns as the one we're currently examining.

This is necessary for supervised analysis and visualization tools in PicturedRocks that use cluster labels.

[4]:
```
pr.read.process_clusts(adata, "paul15_clusters")
```

[4]:
```
AnnData object with n_obs × n_vars = 2730 × 3451
    obs: 'paul15_clusters', 'clust', 'y'
    uns: 'iroot', 'num_clusts', 'clusterindices'
```

Normalize per cell and log transform the data

[5]:
```
sc.pp.normalize_per_cell(adata)
```

[6]:
```
sc.pp.log1p(adata)
```

The `make_infoset` method creates a `SparseInformationSet` object with a discretized version of the data matrix. It is useful to have only a small number of discrete states that each gene can take so that entropy is a reasonable measurement. By default, `make_infoset` performs an adaptive transform that we call a recursive quantile transform. This is implemented in `pr.markers.mutualinformation.infoset.quantile_discretize`. If you have a different discretization transformation, you can pass a transformed matrix directly to `SparseInformationSet`.

[7]:
```
infoset = pr.markers.makeinfoset(adata, True)
```

Because this dataset only has 3451 features, it is computationally easy to do feature selection without restricting the number of features. If we wanted to, we could do either supervised or unsupervised univariate feature selection (i.e., without considering any interactions between features).

[8]:
```
# supervised
mim = pr.markers.mutualinformation.iterative.MIM(infoset)
most_relevant_genes = mim.autoselect(1000)
```

[9]:
```
# unsupervised
ue = pr.markers.mutualinformation.iterative.UniEntropy(infoset)
most_variable_genes = ue.autoselect(1000)
```

At this stage we can slice our `adata` object as `adata[:,most_relevant_genes]` or `adata[:,most_variable_genes]` and create a new `InformationSet` object for this sliced object. We don't need to do that here since there are not a lot of genes but will do so anyway for demonstration purposes.

1.1.1 Supervised Feature Selection

Let's jump straight into supervised feature selection. Here we will use the `CIFE` objective

[10]:
```
adata_mr = adata[:,most_relevant_genes]
infoset_mr = pr.markers.makeinfoset(adata_mr, True)
```

[11]:
```
cife = pr.markers.mutualinformation.iterative.CIFE(infoset_mr)
```

[12]:
```
cife.score[:20]
```

[12]:
```
array([1.15097367, 1.11953242, 1.04094902, 0.98779889, 0.89294165,
       0.82332825, 0.80324986, 0.79920393, 0.69325805, 0.68464788,
       0.66075136, 0.65738939, 0.6331728 , 0.62632343, 0.61487087,
       0.60934578, 0.59525158, 0.59172139, 0.58504638, 0.57874063])
```

[13]:
```
top_genes = np.argsort(cife.score)[::-1]
print(adata_mr.var_names[top_genes[:10]])
```

```
Index(['Prtn3', 'Mpo', 'Ctsg', 'Elane', 'Car2', 'Car1', 'H2afy', 'Calr',
       'Blvrb', 'Fam132a'],
      dtype='object')
```

Let's select 'Mpo'

[14]:
```
ind = adata_mr.var_names.get_loc('Mpo')
```

[15]:
```
cife.add(ind)
```

Now, the top genes are

[16]:
```
top_genes = np.argsort(cife.score)[::-1]
print(adata_mr.var_names[top_genes[:10]])
```

```
Index(['Car1', 'Apoe', 'H2afy', 'Fam132a', 'Car2', 'Mt1', 'Blvrb', 'Srgn',
       'Mt2', 'Prtn3'],
      dtype='object')
```

Observe that the order has changed based on redundancy (or lack thereof) with 'Mpo'. Let's add 'Car1'

[17]:
```
ind = adata_mr.var_names.get_loc('Car1')
cife.add(ind)
```

[18]:
```
top_genes = np.argsort(cife.score)[::-1]
print(adata_mr.var_names[top_genes[:10]])
```

```
Index(['Apoe', 'Ptprcap', 'Gpr56', 'Myb', 'Mcm5', 'Uqcrq', 'Lyar', 'Cox5a',
       'S100a10', 'Snrpd1'],
      dtype='object')
```

If we want to select the top gene repeatedly, we can use `autoselect`

[19]:
```
cife.autoselect(5)
```

To look at the markers we've selected, we can examine `cife.S`

[20]:
```
cife.S
```

[20]:
```
[1, 5, 34, 694, 904, 290, 597]
```

[21]:
```
adata_mr.var_names[cife.S]
```

[21]:
```
Index(['Mpo', 'Car1', 'Apoe', 'Srm', 'Atp5g3', 'Ncl', 'Rps3'], dtype='object')
```

User Interface

This process can also done manually with a user-interface allowing you to incorporate domain knowledge in this process. Use the `View` dropdown to look at heatplots for candidate genes and already selected genes.

[22]:
```
im = pr.markers.interactive.InteractiveMarkerSelection(adata_mr, cife, dim_red="umap",
↪ show_genes=False)
```

Data type cannot be displayed: text/html, text/vnd.plotly.v1+html

```
Running umap on cells...
```

```
/home/umang/anaconda3/envs/fastpr/lib/python3.6/site-packages/sklearn/metrics/
↪pairwise.py:257: RuntimeWarning:

invalid value encountered in sqrt
```

[23]:
```
im.show()
```

```
Output()
```

Note, that because we passed the same `cife` object, any genes added/removed in the interface will affect the `cife` object.

[24]:
```
adata_mr.var_names[cife.S]
```

[24]:
```
Index(['Mpo', 'Car1', 'Apoe', 'Srm', 'Atp5g3', 'Ncl', 'Rps3'], dtype='object')
```

1.1.2 Unsupervised Feature Selection

This works very similarly. In the example below, we'll autoselect 5 genes and then run the interface. Note that although the previous section would not work without cluster labels, the following code will.

[25]:
```
cife_unsup = pr.markers.mutualinformation.iterative.CIFEUnsup(infoset)
```

[26]:
```
cife_unsup.autoselect(5)
```

(If you ran the example above, this will load faster because the t_SNE coordinates for genes and cells have already been computed. You can also customize which plots are displayed with keyword arguments (e.g., `InteractiveMarkerSelection(..., show_genes=False)`). Future versions may allow arbitrary plots.

[27]:
```
im_unsup = pr.markers.interactive.InteractiveMarkerSelection(adata, cife_unsup, show_
↪genes=False, show_cells=False, dim_red="umap")
```

Data type cannot be displayed: text/html, text/vnd.plotly.v1+html

[28]:
```
im_unsup.show()
```

```
Output()
```

1.1.3 Binary Feature Selection

We can also perform feature selection specifically for individual class labels (e.g., clusters). This is done by changing the `SparseInformationSet`'s `y` array. In the example below, we will target the class label "2Ery". Notice that the features selected by MIM (MIM doesn't consider redundancy) are only those that are informative about "2Ery" in particular.

Binary (i.e., not multiclass) feature selection can be performed with any information-theoretic feature selection algorithm (e.g., CIFE, JMI, MIM).

[29]:
```
# since we are changing y anyway, the value of include_y (True in the line below)␣
↪doesn't matter
infoset2 = pr.markers.makeinfoset(adata, True)
infoset2.set_y((adata.obs['clust'] == '2Ery').astype(int).values)
```

[30]:
```
mim2 = pr.markers.mutualinformation.iterative.MIM(infoset2)
```

[31]:
```
im2 = pr.markers.interactive.InteractiveMarkerSelection(adata, mim2, show_genes=False,
↪ dim_red="umap")
```

Data type cannot be displayed: text/html, text/vnd.plotly.v1+html

```
Running umap on cells...
```

```
/home/umang/anaconda3/envs/fastpr/lib/python3.6/site-packages/sklearn/metrics/
↪pairwise.py:257: RuntimeWarning:

invalid value encountered in sqrt
```

[32]:
```
im2.show()
```

```
Output()
```

1.2 Reading data

In addition to various functions for reading input data in scanpy, various methods in *picturedrocks* need cluster labels.

`picturedrocks.read.`**`process_clusts`**(*adata*, *name='clust'*, *copy=False*)

Process cluster labels from an obs column

This copies *adata.obs[name]* into *adata.obs["clust"]* and precomputes cluster indices, number of clusters, etc for use by various functions in PicturedRocks.

Parameters

- **adata** (*anndata.AnnData*) –
- **copy** (*bool*) – determines whether a copy of *AnnData* object is returned

Returns object with annotation

Return type anndata.AnnData

Notes

The information computed here is lost when saving as a *.loom* file. If a *.loom* file has cluster information, you should run this function immediately after `sc.read_loom`.

`picturedrocks.read.`**`read_clusts`**(*adata*, *filename*, *sep=', '*, *name='clust'*, *header=True*, *copy=False*)

Read cluster labels from a csv into an obs column

Parameters

- **adata** (*anndata.AnnData*) – the *AnnData* object to read labels into

- **filename** (*str*) – filename of the csv file with labels
- **sep** (*str, optional*) – csv delimiter
- **name** (*str, optional*) – destination for label is adata.obs[name]
- **header** (*bool*) – deterimes whether csv has a header line. If false, it is assumed that data begins at the first line of csv
- **copy** (*bool*) – determines whether a copy of *AnnData* object is returned

Returns object with cluster labels

Return type anndata.AnnData

Notes

- Cluster ids will automatically be changed so they are 0-indexed
- csv can either be two columns (in which case the first column is treated as observation label and merging handled by pandas) or one column (only cluster labels, ordered as in `adata`)

1.3 Preprocessing

The preprocessing module provides basic preprocessing tools. To avoid reinventing the wheel, we won't repeat methods already in scanpy unless we need functionality not available there.

`picturedrocks.preprocessing.`**`pca`**(*data*, *dim=3*, *center=True*, *copy=False*)

Runs PCA

Parameters

- **data** (*anndata.AnnData*) – input data
- **dim** (*int, optional*) – number of PCs to compute
- **center** (*bool, optional*) – determines whether to center data before running PCA
- **copy** – determines whether a copy of *AnnData* object is returned

Returns object with `obsm["X_pca"]`, and `varm["PCs"]` set

Return type anndata.AnnData

1.4 Plotting

`picturedrocks.plot.`**`genericplot`**(*celldata*, *coords*, ***scatterkwargs*)

Generate a figure for some embedding of data

This function supports both 2D and 3D plots. This may be used to plot data for any embedding (e.g., PCA or t-SNE). For example usage, see code for *pcafigure*.

Parameters

- **celldata** (*anndata.AnnData*) – data to plot
- **coords** (*numpy.ndarray*) – (N, 2) or (N, 3) shaped coordinates of the embedded data
- ****scatterkwargs** – keyword arguments to pass to `Scatter` or `Scatter3D` in *plotly* (dictionaries are merged recursively)

`picturedrocks.plot.`**`genericwrongplot`**(*celldata*, *coords*, *yhat*, *labels=None*, ***scatterkwargs*)

Plot figure with incorrectly classified points highlighted

This can be used with any 2D or 3D embedding (e.g., PCA or t-SNE). For example code, see *pcawrongplot*.

Parameters

- **celldata** (*anndata.AnnData*) – data to plot
- **coords** (*numpy.ndarray*) – (N, 2) or (N, 3) shaped array with coordinates to plot
- **yhat** (*numpy.ndarray*) – (N, 1) shaped array of predicted *y* values
- **labels** (*list, optional*) – list of axis titles
- ****scatterkwargs** – keyword arguments to pass to `Scatter` or `Scatter3D` in *plotly* (dictionaries are merged recursively)

`picturedrocks.plot.`**`pcafigure`**(*celldata*, ***scatterkwargs*)

Make a 3D PCA figure for an AnnData object

Parameters

- **celldata** (*anndata.AnnData*) – data to plot
- ****scatterkwargs** – keyword arguments to pass to `Scatter` or `Scatter3D` in *plotly* (dictionaries are merged recursively)

`picturedrocks.plot.`**`pcawrongplot`**(*celldata*, *yhat*, ***scatterkwargs*)

Generate a 3D PCA figure with incorrectly classified points highlighted

Parameters

- **celldata** (*anndata.AnnData*) – data to plot
- **yhat** (*numpy.ndarray*) – (N, 1) shaped array of predicted *y* values
- ****scatterkwargs** – keyword arguments to pass to `Scatter` or `Scatter3D` in *plotly* (dictionaries are merged recursively)

`picturedrocks.plot.`**`plotgeneheat`**(*celldata*, *coords*, *genes*, *hide_clusts=False*, ***scatterkwargs*)

Generate gene heat plot for some embedding of AnnData

This generates a figure with multiple dropdown options. The first option is "Clust" for a plot similar to *genericplot*, and the remaining dropdown options correspond to genes specified in *genes*. When *celldata.genes* is defined, these drop downs are labeled with the gene names.

Parameters

- **celldata** (*anndata.AnnData*) – data to plot
- **coords** (*numpy.ndarray*) – (N, 2) or (N, 3) shaped coordinates of the embedded data (e.g., PCA or tSNE)
- **genes** (*list*) – list of gene indices or gene names
- **hide_clusts** (*bool*) – Determines if cluster labels are ignored even if they are available

`picturedrocks.plot.`**`umapfigure`**(*adata*, ***scatterkwargs*)

1.5 Selecting Markers

PicturedRocks current implements two categories of marker selection algorithms:

- mutual information-based algorithms

- 1-bit compressed sensing based algorithms

1.5.1 Mutual information

TODO: Explanation of how these work goes here.

Before running any mutual information based algorithms, we need a discretized version of the gene expression matrix, with a limited number of discrete values (because we do not make any assumptions about the distribution of gene expression). Such data is stored in `picturedrocks.markers.InformationSet`, but by default, we suggest using `picturedrocks.markers.makeinfoset()` to generate such an object after appropriate normalization

Iterative Feature Selection

All information-theoretic feature selection methods in PicturedRocks are greedy algorithms. In general, they implement the abstract class `IterativeFeatureSelection` class. See *Supervised Feature Selection* and *Unsupervised Feature Selection* for specific algorithms.

class picturedrocks.markers.mutualinformation.iterative.**IterativeFeatureSelection**(*infoset*)

Abstract Class for Iterative Feature Selection

add(*ind*)

Select specified feature

Parameters **ind** (*int*) – Index of feature to select

autoselect(*n_feats*)

Auto select features

This automatically selects *n_feats* features greedily by selecting the feature with the highest score at each iteration.

Parameters **n_feats** (*int*) – The number of features to select

remove(*ind*)

Remove specified feature

Parameters **ind** (*int*) – Index of feature to remove

Supervised Feature Selection

class picturedrocks.markers.mutualinformation.iterative.**MIM**(*infoset*)

class picturedrocks.markers.mutualinformation.iterative.**CIFE**(*infoset*)

class picturedrocks.markers.mutualinformation.iterative.**JMI**(*infoset*)

Unsupervised Feature Selection

class picturedrocks.markers.mutualinformation.iterative.**UniEntropy**(*infoset*)

class picturedrocks.markers.mutualinformation.iterative.**CIFEUnsup**(*infoset*)

Auxiliary Classes and Methods

class picturedrocks.markers.**InformationSet**(*X*, *has_y=False*)

Stores discrete gene expression matrix

Parameters

- **X** (*numpy.ndarray*) – a (num_obs, num_vars) shape array with `dtype int`
- **has_y** (*bool*) – whether the array *X* has a target label column (a *y* column) as its last column

class picturedrocks.markers.**SparseInformationSet**(*X*, *y=None*)

Stores sparse discrete gene expression matrix

Parameters

- **X** (*scipy.sparse.csc_matrix*) – a (num_obs, num_vars) shape matrix with `dtype int`
- **has_y** (*bool*) – whether the array *X* has a target label column (a *y* column) as its last column

picturedrocks.markers.**makeinfoset**(*adata*, *include_y*, *k=5*)

Discretize data and make a Sparse InformationSet object

Parameters

- **adata** (*anndata.AnnData*) – The data to discretize. By default data is discretized as *round(log2(X + 1))*.
- **include_y** (*bool*) – Determines if the *y* (cluster label) column in included in the *InformationSet* object

Returns An object that can be used to perform information theoretic calculations.

Return type *SparseInformationSet*

1.5.2 Interactive Marker Selection

class picturedrocks.markers.interactive.**InteractiveMarkerSelection**(*adata*, *feature_selection*, *disp_genes=10*, *connected=True*, *show_cells=True*, *show_genes=True*, *dim_red='tsne'*)

Run an interactive marker selection GUI inside a jupyter notebook

Parameters

- **adata** (*anndata.AnnData*) – The data to run marker selection on. If you want to restrict to a small number of genes, slice your anndata object.
- **feature_selection** (picturedrocks.markers.mutualinformation.iterative.IterativeFeatureSelection) – An instance of a interative feature selection algorithm class that corresponds to *adata* (i.e., the column indices in *feature_selection* should correspond to the column indices in *adata*)

- **disp_genes** (*int*) – Number of genes to display as options (by default, number of genes plotted on the tSNE plot is *3 * disp_genes*, but can be changed by setting the *plot_genes* property after initializing.
- **connected** (*bool*) – Parameter to pass to *plotly.offline.init_notebook_mode*. If your browser does not have internet access, you should set this to False.
- **show_cells** (*bool*) – Determines whether to display a tSNE plot of the cells with a drop-down menu to look at gene expression levels for candidate genes.
- **show_genes** (*bool*) – Determines whether to display a tSNE plot of genes to visualize gene similarity
- **dim_red** (*{"tsne", "umap"}*) – Dimensionality reduction algorithm

> **Warning:** This class requires modules not explicitly listed as dependencies of picturedrocks. Specifically, please ensure that you have *ipywidgets* installed and that you use this class only inside a jupyter notebook.

`picturedrocks.markers.interactive.`**`cife_obj`**(*H*, *i*, *S*)

The CIFE objective function for feature selection

Parameters

- **H** (*function*) – an entropy function, typically the bound method H on an instance of InformationSet. For example, if *infoset* is of type *picturedrocks.markers.InformationSet*, then pass *infoset.H*
- **i** (*int*) – index of candidate gene
- **S** (*list*) – list of features already selected

Returns the candidate feature's score relative to the selected gene set *S*

Return type float

1.6 Measuring Feature Selection Performance

This module can be used to evaluate feature selection methods via K-fold cross validation.

class `picturedrocks.performance.`**`FoldTester`**(*adata*)

Performs K-fold Cross Validation for Marker Selection

FoldTester can be used to evaluate various marker selection algorithms. It can split the data in *K* folds, run marker selection algorithms on these folds, and classify data based on testing and training data.

Parameters **adata** (*anndata.AnnData*) – data to slice into folds

classify(*classifier*)

Classify each cell using training data from other folds

For each fold, we project the data onto the markers selected for that fold, which we treat as test data. We also project the complement of the fold and treat that as training data.

Parameters **classifier** – a classifier that trains with a training data set and predicts labels of test data. See *NearestCentroidClassifier* for an example.

Note: The *classifier* should not attempt to modify data in-place. Any preprocessing should be done on a copy.

loadfolds(*file*)

Load folds from a file

The file can be one saved either by `FoldTester.savefolds()` or `FoldTester.savefoldsandmarkers()`. In the latter case, it will not load any markers.

See also:

`FoldTester.loadfoldsandmarkers()`

loadfoldsandmarkers(*file*)

Load folds and markers

Loads a folds and markers file saved by `FoldTester.savefoldsandmarkers()`

Parameters **file** (*str*) – filename to load from (typically with a `.npz` extension)

See also:

`FoldTester.loadfolds()`

makefolds(*k=5*, *random=False*)

Makes folds

Parameters

- **k** (*int*) – the value of K
- **random** (*bool*) – If true, *makefolds* will make folds randomly. Otherwise, the folds are made in order (i.e., the first `ceil(N / k)` cells in the first fold, etc.)

savefolds(*file*)

Save folds to a file

Parameters **file** (*str*) – filename to save (typically with a `.npz` extension)

savefoldsandmarkers(*file*)

Save folds and markers for each fold

This saves folds, and for each fold, the markers previously found by `FoldTester.selectmarkers()`.

Parameters **file** (*str*) – filename to save to (typically with a `.npz` extension)

selectmarkers(*select_function*)

Perform a marker selection algorithm on each fold

Parameters **select_function** (*function*) – a function that takes in an `AnnData` object and outputs a list of gene markers, given by their index

Note: The *select_function* should not attempt to modify data in-place. Any preprocessing should be done on a copy.

validatefolds()

Ensure that all observations are in exactly one fold

Returns

Return type bool

class picturedrocks.performance.**NearestCentroidClassifier**

Nearest Centroid Classifier for Cross Validation

Computes the centroid of each cluster label in the training data, then predicts the label of each test data point by finding the nearest centroid.

test(*Xtest*)

train(*adata*)

class picturedrocks.performance.**PerformanceReport**(*y*, *yhat*)

Report actual vs predicted statistics

Parameters

- **y** (*numpy.ndarray*) – actual cluster labels, (N, 1)-shaped numpy array
- **yhat** (*numpy.ndarray*) – predicted cluster labels, (N, 1)-shaped numpy array

confusionmatrixfigure()

Compute and make a confusion matrix figure

Returns confusion matrix

Return type *plotly figure*

getconfusionmatrix()

Get the confusion matrix for the latest run

Returns array of shape (K, K), with the [i, j] entry being the fraction of cells in cluster i that were predicted to be in cluster j

Return type numpy.ndarray

printscore()

Print a message with the score

show()

Print a full report

This uses *iplot*, so we assume this will only be run in a Jupyter notebook and that *init_notebook_mode* has already been run.

wrong()

Returns the number of cells misclassified.

picturedrocks.performance.**kfoldindices**(*n*, *k*, *random=False*)

Generate indices for k-fold cross validation

Parameters

- **n** (*int*) – number of observations
- **k** (*int*) – number of folds
- **random** (*bool*) – determines whether to randomize the order

Yields *numpy.ndarray* – array of indices in each fold

picturedrocks.performance.**merge_markers**(*ft*, *n_markers*)

picturedrocks.performance.**truncatemarkers**(*ft*, *n_markers*)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

R

S

T

U

V

W