pymds Documentation

Release 0.1.6

David Pattinson

Jul 23, 2018

Contents

Metric multidimensional scaling in python.

CHAPTER 1

Installation

Use pip:

```
pip install pymds
```

CHAPTER 2

Reference

2.1 pymds.DistanceMatrix

class pymds.**DistanceMatrix**(*path_or_array_like*, *na_values='*'*)

A distance matrix.

Parameters

- **path_or_array_like** (*str* or *array-like*) – If *str*, path to csv containing distance matrix. If *array-like*, the distance matrix. Must be square.
- **na_values** (*str*) – How nan is represented in csv file.

Notes

Negative distances are converted to 0.

optimize(*start=None*, *n=2*)

Run multidimensional scaling on this distance matrix.

Parameters

- **start** (*None* or *array-like*) – Starting coordinates. If *start=None*, random starting coordinates are used. If *array-like* must have shape [*m* * *n*,].
- **n** (*int*) – Number of dimensions to embed samples in.

Examples

```
>>> import pandas as pd
>>> from pymds import DistanceMatrix
>>> dist = pd.DataFrame({
...     'a': [0.0, 1.0, 2.0],
```

(continues on next page)

(continued from previous page)

```
...     'b': [1.0, 0.0, 3 ** 0.5],
...     'c': [2.0, 3 ** 0.5, 0.0]} , index=['a', 'b', 'c'])
>>> dm = DistanceMatrix(dist)
>>> pro = dm.optimize(n=2)
>>> pro.coords.shape
(3, 2)
>>> type(pro)
<class 'pymds.mds.Projection'>
```

Returns: *pymds.Projection*

optimize_batch(*batchsize=10*, *returns='best'*, *paralell=True*)

Run multiple optimizations using different starting coordinates.

Parameters

- **batchsize** (*int*) – Number of optimizations to run.
- **returns** (*str*) – If `'all'`, return results of all optimizations, ordered by stress, ascending. If `'best'` return the projection with the lowest stress.
- **parallel** (*bool*) – If `True`, run optimizations in parallel.

Examples

```
>>> import pandas as pd
>>> from pymds import DistanceMatrix
>>> dist = pd.DataFrame({
...     'a': [0.0, 1.0, 2.0],
...     'b': [1.0, 0.0, 3 ** 0.5],
...     'c': [2.0, 3 ** 0.5, 0.0]} , index=['a', 'b', 'c'])
>>> dm = DistanceMatrix(dist)
>>> batch = dm.optimize_batch(batchsize=3, returns='all')
>>> len(batch)
3
>>> type(batch[0])
<class 'pymds.mds.Projection'>
```

Returns

list: Length batchsize, containing instances of *pymds.Projection*. Sorted by stress, ascending.

or

pymds.Projection: Projection with the lowest stress.

Return type *list* or *pymds.Projection*

2.2 pymds.Projection

class pymds.**Projection**(*coords*)

Samples embedded in n-dimensions.

Parameters **coords** (pandas.DataFrame) – Coordinates of the projection.

coords
pandas.DataFrame – Coordinates of the projection.

stress
float – Residual error of multidimensional scaling. (If generated using *Projection.from_optimize_result()*).

classmethod from_optimize_result(*result*, *n*, *m*, *index=None*)
Construct a Projection from the output of an optimization.

Parameters

- **result** (scipy.optimize.OptimizeResult) – Object returned by scipy.optimize.minimize().
- **n** (*int*) – Number of dimensions.
- **m** (*int*) – Number of samples.
- **index** (*list-like*) – Names of samples. (Optional).

Returns *pymds.Projection*

orient_to(*other*, *index=None*, *inplace=False*, *scaling=False*)
Orient this Projection to another dataset.

Orient this projection using reflection, rotation and translation to match another projection using procrustes superimposition. Scaling is optional.

Parameters

- **other** – (*pymds.Projection* or pandas.DataFrame or *array-like*): The other dataset to orient this projection to. If other is an instance of *pymds.Projection* or pandas.DataFrame, then other must have indexes in common with this projection. If *array-like*, then other must have the same dimensions as self.coords.
- **index** (*list-like* or *None*) – If other is an instance of pandas.DataFrame or *pymds.Projection* then orient this projection to other using only samples in index.
- **inplace** (*bool*) – Update coordinates of this projection inplace, or return an instance of *pymds.Projection*.
- **scaling** (*bool*) – Allow scaling. (Not implemented yet).

Examples

```
>>> import numpy as np
>>> import pandas as pd
>>> from pymds import Projection
>>> array = np.random.randn(10, 2)
>>> pro = Projection(pd.DataFrame(array))
>>> # Flip left-right, rotate 90 deg and translate
>>> other = np.fliplr(array)
>>> other = np.dot(other, np.array([[0, -1], [1, 0]]))
>>> other += np.array([10, -5])
>>> oriented = pro.orient_to(other)
>>> (oriented.coords.values - other).sum() < 1e-6
True
```

Returns If inplace=False.

Return type *pymds.Projection*

plot(***kwds*)

Plot the coordinates in the first two dimensions of the projection.

Removes axis and tick labels, and sets the grid spacing to 1 unit. One way to display the grid is to use Seaborn:

Parameters ****kwds** – Passed to `pandas.DataFrame.plot.scatter()`.

Examples

```
>>> from pymds import DistanceMatrix
>>> import pandas as pd
>>> import seaborn as sns
>>> sns.set_style('whitegrid')
>>> dist = pd.DataFrame({
...     'a': [0.0, 1.0, 2.0],
...     'b': [1.0, 0.0, 3 ** 0.5],
...     'c': [2.0, 3 ** 0.5, 0.0]} , index=['a', 'b', 'c'])
>>> dm = DistanceMatrix(dist)
>>> pro = dm.optimize()
>>> ax = pro.plot(c='black', s=50, edgecolor='white')
```

Returns `matplotlib.axes.Axes`

plot_lines_to(*other*, *index=None*, ***kwds*)

Plot lines from samples shared between this projection and another dataset.

Parameters

- **other** – (*pymds.Projection* or `pandas.DataFrame` or *array-like*): The other dataset to plot lines to. If other is an instance of *pymds.Projection* or `pandas.DataFrame`, then other must have indexes in common with this projection. If *array-like*, then other must have the same dimensions as *self.coords*.
- **index** (*list-like* or *None*) – Only draw lines between samples in index. All elements in index must be samples in this projection and other.
- ****kwds** – Passed to `matplotlib.collections.LineCollection`.

Examples

```
>>> import numpy as np
>>> from pymds import Projection
>>> pro = Projection(np.random.randn(50, 2))
>>> R = np.array([[0, -1], [1, 0]])
>>> other = np.dot(pro.coords, R)  # Rotate 90 deg
>>> ax = pro.plot(c='black', edgecolor='white', zorder=20)
>>> ax = pro.plot_lines_to(other, linewidths=0.3)
```

Returns `matplotlib.axes.Axes`

CHAPTER 3

Background

Multidimensional scaling aims to embed samples as points in n-dimensional space, where the distances between points represent distances between samples in data.

In this example, edges of a triangle are specified by setting the distances between three vertices a, b and c. These data can be represented perfectly in 2-dimensions.

```
import pandas as pd
from pymds import DistanceMatrix

# Distances between the vertices of a right-angled triangle
dist = pd.DataFrame({
    'a': [0.0, 1.0, 2.0],
    'b': [1.0, 0.0, 3 ** 0.5],
    'c': [2.0, 3 ** 0.5, 0.0]},
    index=['a', 'b', 'c'])

# Make an instance of DistanceMatrix
dm = DistanceMatrix(dist)

# Embed vertices in two dimensions
projection = dm.optimize(n=2)
```

In data where distances between samples cannot be represented perfectly in the number of dimensions used, residual error will exist among the distances between samples in the space and the distances in the data.

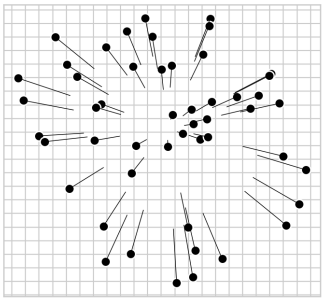
Error in MDS is also known as *stress*.

CHAPTER 4

Usage

The following example demonstrates some simple pymds features.

```
from pymds import DistanceMatrix

from numpy.random import uniform, seed
from scipy.spatial.distance import pdist, squareform

import seaborn as sns
sns.set_style('whitegrid')

# 50 random 2D samples
seed(1234)
samples = uniform(low=-10, high=10, size=(50, 2))

# Measure pairwise distances between samples
dists = squareform(pdist(samples))

dists_shrunk = dists * 0.65

# Embed
original = DistanceMatrix(dists).optimize()
shrunk = DistanceMatrix(dists_shrunk).optimize()

shrunk.orient_to(original, inplace=True)

original.plot(c='black', edgecolor='white', s=50)
original.plot_lines_to(shrunk, linewidths=0.5, colors='black')
```



Index