TrendyPy

Release 0.1.1

Dogan Askan

Aug 31, 2020

CONTENTS

CHAPTER

ONE

TRENDYPY

TrendyPy is a small Python package for trend line clustering. It is developed to create time series clusters by calculating trend similarity distance with Dynamic Time Warping.

1.1 Installation

You can install TrendyPy with pip.

```
pip install trendypy
```

TrendyPy depends on Pandas, Numpy and fastdtw and works in Python 3.5+.

1.2 Quickstart

Trendy has scikit-learn like api to allow easy integration to existing programs.

```
>>> from trendypy.trendy import Trendy
>>> a = [1, 2, 3, 4, 5] # increasing trend
>>> b = [1, 2.1, 2.9, 4.4, 5.1] # increasing trend
>>> c = [6.2, 5, 4, 3, 2] # decreasing trend
>>> d = [7, 6, 5, 4, 3, 2, 1] # decreasing trend
>>> trendy = Trendy(n_clusters=2)
>>> trendy.fit([a, b, c, d])
>>> print(trendy.labels_)
[0, 0, 1, 1]
>>> trendy.predict([[0.9, 2, 3.1, 4]]) # another increasing trend
[0]
```

Refer to this extensive demo to see it in action or just check API Reference for details.

1.3 Post

The idea is originated from the post Trend Clustering.

CHAPTER

TWO

SEE IN ACTION

In this demo, I'd like to show you how to use TrendyPy in some `stock data` between 2018-01-01 and 2020-06-28. You can download the data from `here` to reproduce the demo.

Let's say we have some stock data from a combination of tech and banking. And, we want to identify an unknown trend if it's a tech stock or banking. For this purpose, we'll use FB (i.e. Facebook), GOOGL (i.e. Google), AMZN (i.e Amazon), BAC (i.e. Bank of America) and WFC (i.e. Wells Fargo) for training data then AAPL (i.e. Apple) and c (i.e. Citigroup) for prediction data.

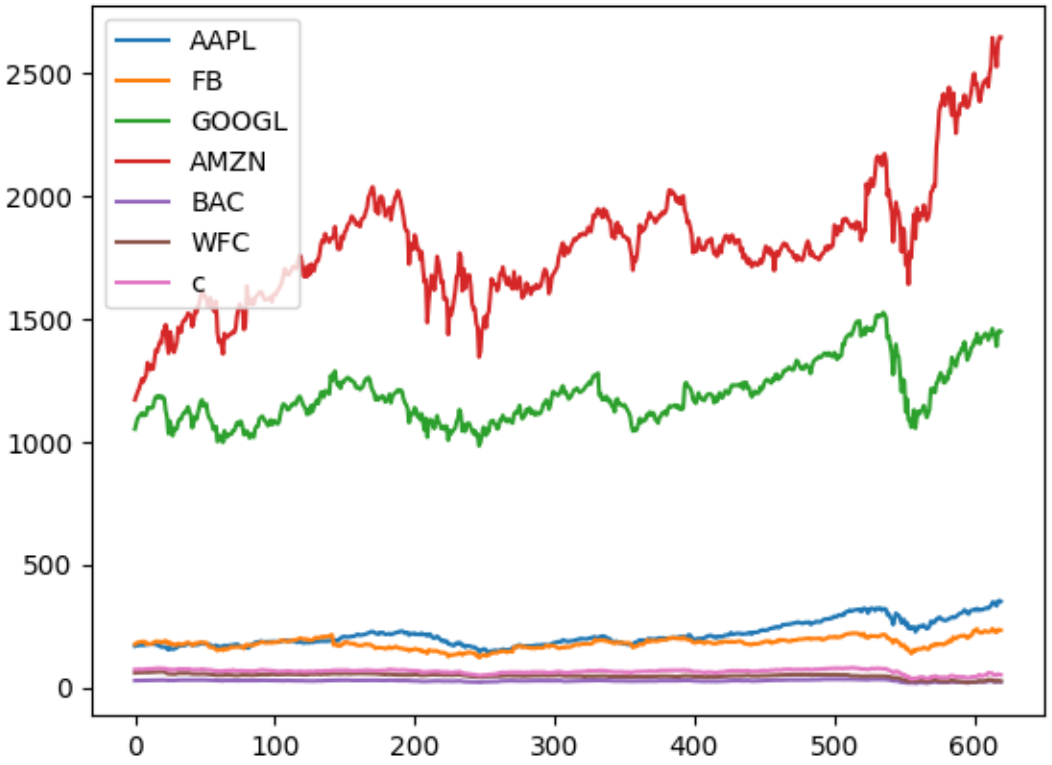
But first, here is how the data looks.

```
In [1]: import pandas as pd

In [2]: import matplotlib.pyplot as plt

In [3]: df = pd.read_csv('stock_data.csv')

In [4]: df.plot()
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc93e6c5c50>
```

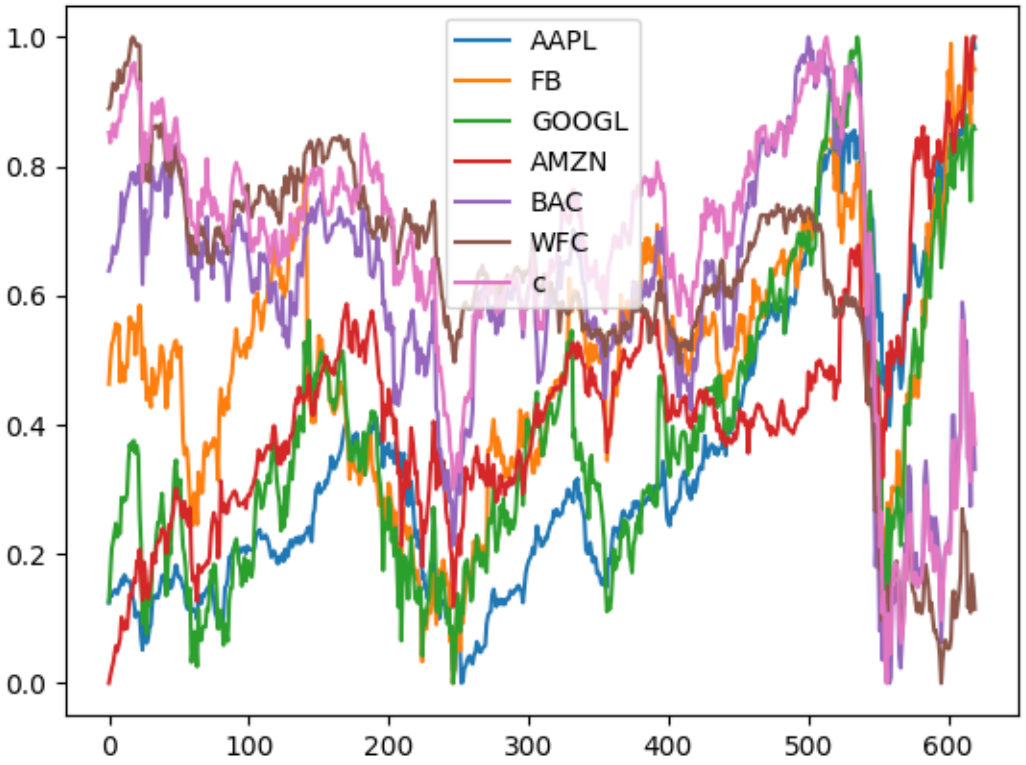


If we cluster like this, the expensive stocks like GOOGL and AMZN will alone constitute one cluster which it's clearly not intended. So, let's scale first.

```
In [5]: from trendypy import utils

In [6]: df = df.apply(utils.scale_01)

In [7]: df.plot()
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc93e1a3390>
```



It's a bit apparent that BAC, WFC and c are different than the others. Let's put sectors side by side to see the difference better.

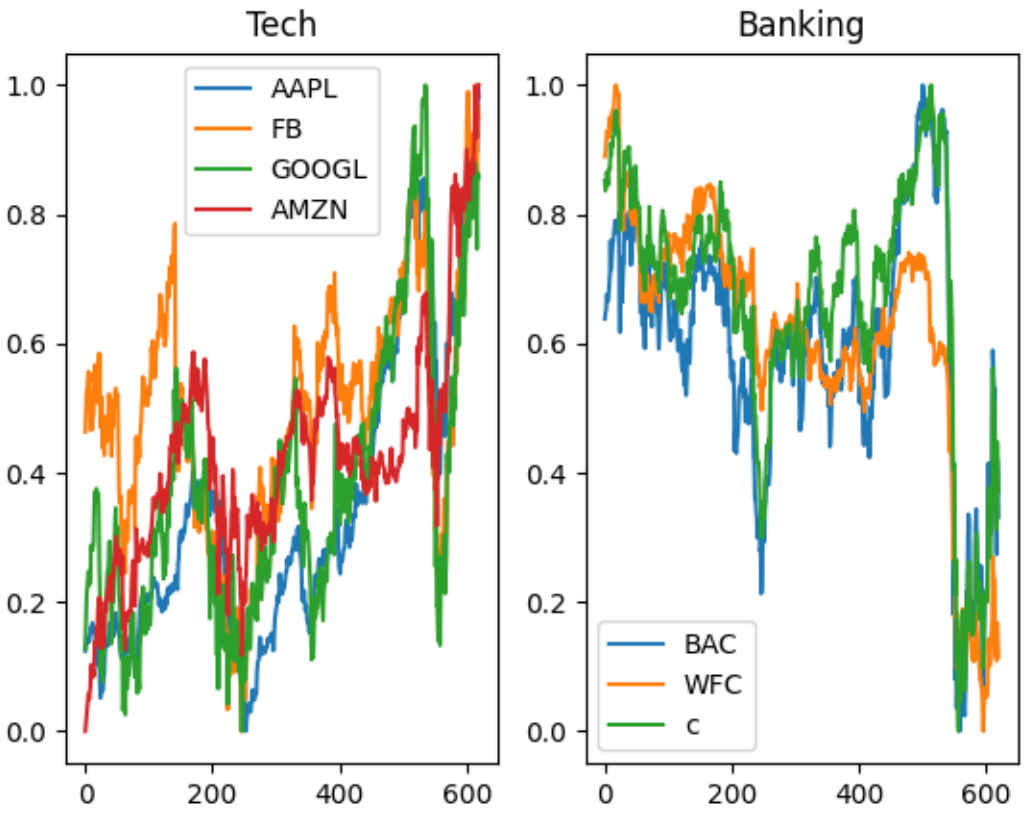
```
In [8]: fig, axes_ = plt.subplots(nrows=1, ncols=2)

In [9]: axes_[0].set_title('Tech')
Out[9]: Text(0.5, 1.0, 'Tech')

In [10]: axes_[1].set_title('Banking')
Out[10]: Text(0.5, 1.0, 'Banking')

In [11]: df[['AAPL', 'FB', 'GOOGL', 'AMZN']].plot(ax=axes_[0])
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc93e13eeb8>

In [12]: df[['BAC', 'WFC', 'c']].plot(ax=axes_[1])
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc93e0d5d68>
```



Now, we can use the training data to fit. Remember, we're setting AAPL and c aside to predict later and only fit by using the rest.

```
In [13]: from trendypy.trendy import Trendy

In [14]: trendy = Trendy(n_clusters=2) # 2 for tech and banking

In [15]: trendy.fit([df.FB, df.GOOGL, df.AMZN, df.BAC, df.WFC])

In [16]: trendy.labels_
Out[16]: [0, 0, 0, 1, 1]
```

You can also use fit_predict method for this purpose, it's essentially the same.

```
In [17]: trendy.fit_predict([df.FB, df.GOOGL, df.AMZN, df.BAC, df.WFC])
Out[17]: [0, 0, 0, 1, 1]
```

As expected, it successfully assigns FB, GOOGL and AMZN into the first cluster (i.e. `0`) and BAC and WFC into the second (i.e. `1`). So, we can name `0` as tech and `1` as banking.

Now, let's make predictions on the prediction data that we set aside earlier (i.e. AAPL, c).

```
In [18]: trendy.predict([df.AAPL]) # expecting `0` since AAPL is a part of tech
Out[18]: [0]

In [19]: trendy.predict([df.c]) # expecting `1` since c is a part of banking
```

(continues on next page)

(continued from previous page)

```
Out[19]: [1]
```

As seen above, it correctly predicts trends.

You can easily pickle the model object to be used later with to_pickle method.

```
In [20]: trendy.to_pickle('my_first_trendy.pkl')
```

And, that's all.

CHAPTER

THREE

API REFERENCE

3.1 trendypy.trendy

class trendy.**Trendy**(*n_clusters*, *algorithm=<function fastdtw_distance>*)

Bases: object

Estimator to cluster trend-lines and assign new lines accordingly.

Notes

Scaling and missing values need to be handled externally.

Parameters

- **n_clusters** (*int*) – The number of clusters to form.
- **algorithm** (*callable*) – Algorithm to calculate the difference. Default is fast DTW with Euclidean.

Example

```
>>> a = [1, 2, 3, 4, 5] # increasing trend
>>> b = [1, 2.1, 2.9, 4.4, 5.1] # increasing trend
>>> c = [6.2, 5, 4, 3, 2] # decreasing trend
>>> d = [7, 6, 5, 4, 3, 2, 1] # decreasing trend
>>> trendy = Trendy(n_clusters=2)
>>> trendy.fit([a, b, c, d])
>>> print(trendy.labels_)
[0, 0, 1, 1]
>>> trendy.predict([[0.9, 2, 3.1, 4]]) # another increasing trend
[0]
```

labels_ = None

cluster_centers_ = None

fit(*X*)

Compute clustering based on given distance algorithm.

Parameters **X** (*array of arrays*) – Training instances to cluster.

Example

```
>>> a = [1, 2, 3, 4, 5] # increasing
>>> b = [1, 2.1, 2.9, 4.4, 5.1] # increasing
>>> c = [6.2, 5, 4, 3, 2] # decreasing
>>> d = [7, 6, 5, 4, 3, 2, 1] # decreasing
>>> trendy = Trendy(2)
>>> trendy.fit([a, b, c, d])
>>> print(trendy.labels_)
[0, 0, 1, 1]
```

predict(*X*)

Predict the closest cluster each sample in X belongs to.

Parameters **X** (*array of arrays*) – New data to predict.

Returns Index of the cluster each sample belongs to.

Return type list

Example

```
>>> a = [1, 2, 3, 4, 5] # increasing
>>> b = [1, 2.1, 2.9, 4.4, 5.1] # increasing
>>> c = [6.2, 5, 4, 3, 2] # decreasing
>>> d = [7, 6, 5, 4, 3, 2, 1] # decreasing
>>> trendy = Trendy(2)
>>> trendy.fit([a, b, c, d])
>>> trendy.predict([[0.9, 2, 3.1, 4]])
[0]
>>> trendy.predict([[0.9, 2, 3.1], [7, 6.6, 5.5, 4.4]])
[0, 1]
```

assign(*X*)

Alias of *predict()*

fit_predict(*X*)

Compute cluster centers and predict cluster index for each sample.

Parameters **X** (*array of arrays*) – Training instances to cluster.

Returns predicted labels

Return type list

Example

```
>>> a = [1, 2, 3, 4, 5] # increasing
>>> b = [1, 2.1, 2.9, 4.4, 5.1] # increasing
>>> c = [6.2, 5, 4, 3, 2] # decreasing
>>> d = [7, 6, 5, 4, 3, 2, 1] # decreasing
>>> trendy = Trendy(2)
>>> trendy.fit_predict([a, b, c, d])
[0, 0, 1, 1]
```

to_pickle(*path*)

Pickle (serialize) object to a file.

Parameters **path** (*str*) – file path where the pickled object will be stored

Example

To save a **.pkl* file:

```
>>> t1 = Trendy(n_clusters=2)
>>> t1.fit([[1, 2, 3], [2, 3, 3]])
>>> t1.to_pickle(path='trendy.pkl')
```

To load the same object later:

```
>>> import pickle, os
>>> pkl_file = open('trendy.pkl', 'rb')
>>> t2 = pickle.load(pkl_file)
>>> pkl_file.close()
>>> os.remove('trendy.pkl')
```

3.2 trendypy.algos

Algorithms for the package.

algos.**dtw_distance**(*x*, *y*, *d=<function distance_euclidean>*, *scaled=False*)

Returns the distance of two arrays with dynamic time warping method.

Parameters

- **x** (*iter*) – input array 1
- **y** (*iter*) – input array 2
- **d** (*func*) – distance function, default is euclidean
- **scaled** (*bool*) – should arrays be scaled (i.e. 0-1) before calculation

Returns

distance, 0.0 means arrays are exactly same, upper limit is positive infinity

Return type float

References

https://en.wikipedia.org/wiki/Dynamic_time_warping

Examples

```
>>> dtw_distance([1, 2, 3, 4], [1, 2, 3, 4])
0.0
>>> dtw_distance([1, 2, 3, 4], [0, 0, 0])
10.0
>>> dtw_distance([1, 2, 3, 4], [0, 2, 0, 4])
4.0
>>> dtw_distance([1, 2, 3, 4], [10, 20, 30, 40])
90.0
```

(continues on next page)

(continued from previous page)

```
>>> dtw_distance([1, 2, 3, 4], [10, 20, 30, 40], scaled=True)
0.0
```

algos.**fastdtw_distance**(*x*, *y*, *d=<function distance_euclidean>*)

Dynamic Time Warping (DTW) algorithm with an O(N) time and memory complexity.

Parameters

- **x** (*iter*) – input array 1
- **y** (*iter*) – input array 2
- **d** (*func*) – distance function, default is euclidean

Returns

distance, 0.0 means arrays are exactly same, upper limit is positive infinity

Return type float

References

https://pypi.org/project/fastdtw/

Examples

```
>>> fastdtw_distance([1, 2, 3, 4], [1, 2, 3, 4])
0.0
>>> fastdtw_distance([1, 2, 3, 4], [0, 0, 0])
10.0
>>> fastdtw_distance([1, 2, 3, 4], [0, 2, 0, 4])
4.0
>>> fastdtw_distance([1, 2, 3, 4], [10, 20, 30, 40])
90.0
```

3.3 trendypy.utils

Utility functions for the package.

utils.**scale_01**(*x*)

Scales array to 0-1.

Parameters **x** (*iter*) – 1d array of float

Returns scaled 1d array

Return type np.array

Example

```
>>> scale_01([1, 2, 3, 5]).tolist()
[0.0, 0.25, 0.5, 1.0]
```

utils.**distance_abs**(*x*, *y*)

Returns absolute distance.

Parameters

- **x** (*float*) – input 1
- **y** (*float*) – input 2

Returns |x-y|

Return type float

Example

```
>>> distance_abs(5, 7)
2.0
>>> distance_abs(4, 1)
3.0
```

utils.**distance_euclidean**(*x*, *y*)

Returns Euclidean distance.

Parameters

- **x** (*float or iter*) – input 1
- **y** (*float or iter*) – input 2

Returns Euclidean distance

Return type float

References

https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html

Examples

```
>>> x, y = 1, 2
>>> distance_euclidean(x, y)
1.0
>>> x, y = [1, 2], [4, 6]
>>> distance_euclidean(x, y)
5.0
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX