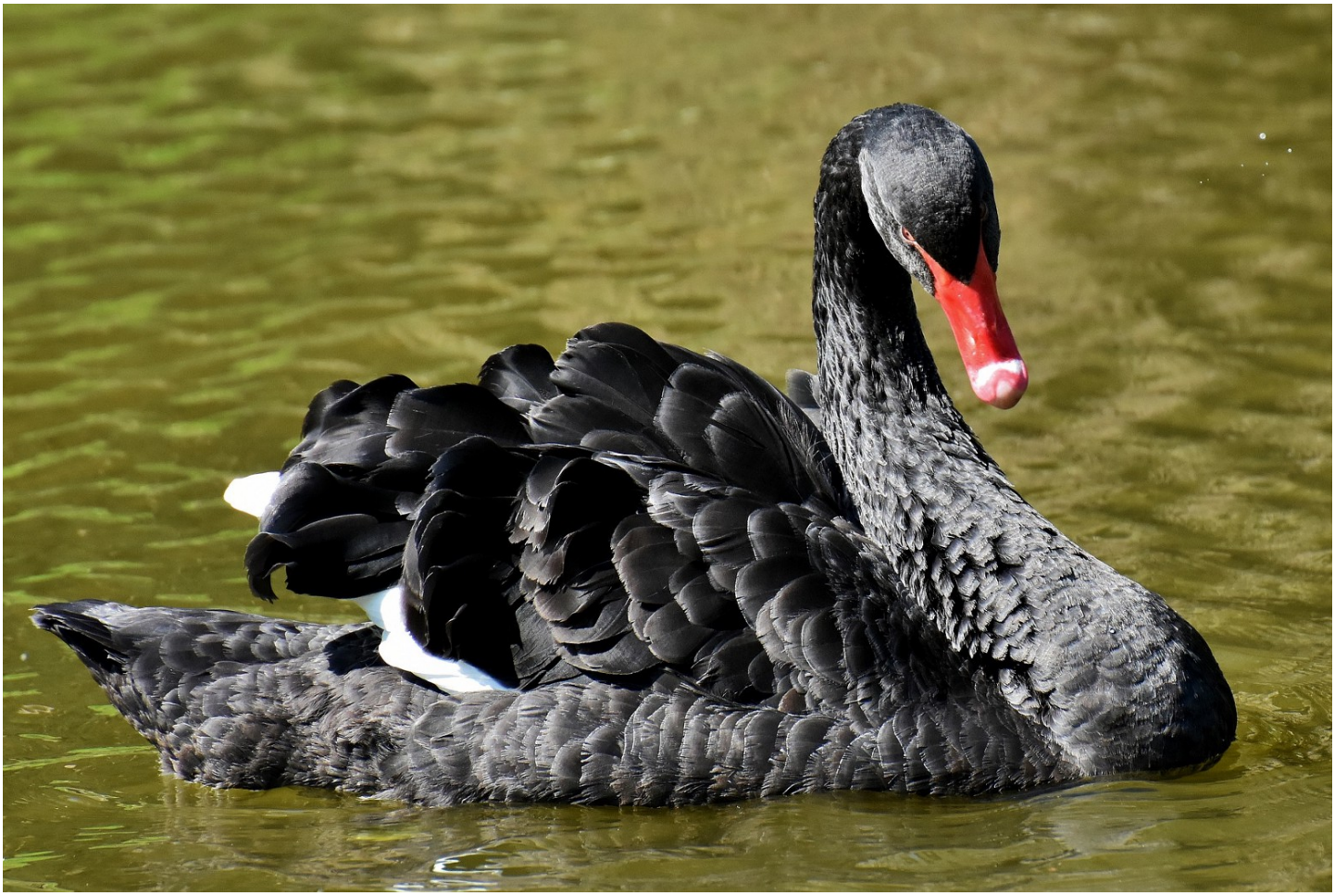
This is your **last** free story this month. Sign up and get an extra one for free.

Local Outlier Factor | Simple Python Example

doedotdev
Nov 20, 2018 · 4 min read ★



Most of you are here because you read my Local Outlier Factor | Example By Hand article. If you haven't, go ahead and check it out here.

Local Outlier Factor | Example By Hand

Local Outlier Factor value is a commonly used anomaly detection tool. It takes a local approach to better detect...

medium.com

The good news is, the implementation is easier than all that paper stuff. However I think it is good to know the basics before scaling it.

The Code

The code lives here.

mtngt/local_outlier_factor

Local Outlier Factor in Python. Contribute to mtngt/local_outlier_factor development by creating an account on GitHub.

github.com

It is tested pretty well, and works well. However these things are still on the todo list.

- Support more distance functions
- Test for decimal values (x = 2.5 and y = 3.53635423) as right now it only works for whole number points)
- Stress tests for hundreds of thousands of points
- Optimize for hundreds of thousands of points

The Problem

In the following examples I will be using manhattan distance and a k value of 2 on the following coordinates.

```
Point (X,Y)
a     (0,0)
b     (0,1)
c     (1,1)
d     (3,0)

(Working on a markdown generator for these basic charts.)

2-|
  |
1-|*b *c
  |
0-|*a_________*d
  |   |   |   |
  0   1   2   3
```

Sample Runs

This is implemented as an ordered dict. However, I googled around for the most common forms I saw X,Y coordinates in and accept four different ways to input coordinates.

Input as OrderedDict: Note, this is the only method to name your coordinates. Every other input method will name the coordinates for you.

```
coords_as_ordered_dict = OrderedDict([
    ('a', OrderedDict([
        ('x', 0),
        ('y', 0)
    ])),
    ('b', OrderedDict([
        ('x', 0),
        ('y', 1)
    ])),
    ('c', OrderedDict([
        ('x', 1),
        ('y', 1)
    ])),
    ('d', OrderedDict([
        ('x', 3),
        ('y', 0)
    ]))
])

test_lof = lof.LOF(coords_as_ordered_dict, lof.LOF.CONST_MANHATTAN,
2)
lofs = test_lof.print_all_lof()
```

Output:

```
a: 0.8749999999999999
b: 1.3333333333333333
c: 0.8749999999999999
d: 2.0
```

. . .

Input as Array of Tuples: Notice we aren't giving the points names anymore. They will name themselves.

```
coords_as_array_of_tuples = [(0, 0), (0, 1), (1, 1), (3, 0)]

test_lof = lof.LOF(coords_as_array_of_tuples,
lof.LOF.CONST_MANHATTAN, 2)
lofs = test_lof.print_all_lof()
```

Output:

```
coord_0_x_0_y_0: 0.8749999999999999
coord_1_x_0_y_1: 1.3333333333333333
coord_2_x_1_y_1: 0.8749999999999999
coord_3_x_3_y_0: 2.0
```

. . .

Input as CSV File of one X,Y pair per line: Notice we aren't giving the points names anymore. They will name themselves.

csv file: *test.csv*

```
0,0
0,1
1,1
3,0
```

Code with input as csv file name: Notice we aren't giving the points names anymore. They will name themselves.

```
test_lof = lof.LOF('test.csv', lof.LOF.CONST_MANHATTAN, 2)
lofs = test_lof.print_all_lof()
```

Output:

```
coord_0_x_0_y_0: 0.8749999999999999
coord_1_x_0_y_1: 1.3333333333333333
```

```
coord_2_x_1_y_1: 0.8749999999999999
coord_3_x_3_y_0: 2.0
```

. . .

Input as X and Y Array: Notice we aren't giving the points names anymore. They will name themselves.

```
x = [0, 0, 1, 3]
y = [0, 1, 1, 0]
coords_as_x_y_array = [x, y]

test_lof = lof.LOF(coords_as_x_y_array, lof.LOF.CONST_MANHATTAN, 2)
lofs = test_lof.print_all_lof()
```

Output:

```
coord_0_x_0_y_0: 0.8749999999999999
coord_1_x_0_y_1: 1.3333333333333333
coord_2_x_1_y_1: 0.8749999999999999
coord_3_x_3_y_0: 2.0
```

. . .

Other Methods

But wait, there is more.

Sorting LOFs Ascending:

```
test_lof = lof.LOF(self.coords, lof.LOF.CONST_MANHATTAN, 2)
lofs = test_lof.get_lof_sorted_filtered(False)
```

Output: Sorted Ascending

```
('a', 0.8749999999999999)
('c', 0.8749999999999999)
('b', 1.3333333333333333)
('d', 2.0)
```

. . .

Sorting LOFs Descending:

```
test_lof = lof.LOF(self.coords, lof.LOF.CONST_MANHATTAN, 2)
lofs = test_lof.get_lof_sorted_filtered(True)
```

Output: Sorted Descending

```
('d', 2.0)
('b', 1.3333333333333333)
('a', 0.8749999999999999)
('c', 0.8749999999999999)
```

. . .

Filtering LOFs in Range: Values greater than 1 and less than 2

```
test_lof = lof.LOF(self.coords, lof.LOF.CONST_MANHATTAN, 2)
lofs = test_lof.get_lof_sorted_filtered(False, 1, 2)
```

Output: The only value greater than 1 and less than 2

```
('b', 1.3333333333333333)
```

. . .

Filtering LOFs in Range Descending: Values greater than 0 and less than 2

```
test_lof = lof.LOF(self.coords, lof.LOF.CONST_MANHATTAN, 2)
lofs = test_lof.get_lof_sorted_filtered(True, 0, 2)
```

Output:

```
('b', 1.3333333333333333)
('a', 0.8749999999999999)
('c', 0.8749999999999999)
```

. . .

Get All Data About a Coordinate

Code:

```
test_lof = lof.LOF(self.coords, lof.LOF.CONST_MANHATTAN, 2)
test_lof.print_all_data()
```

Output:

This will show you the following info

- X coordinate
- Y coordinate
- K nearest nodes
- Distance to its K nearest nodes
- local outlier factor
- local reachability distance * (ONLY IF IT WAS NEEDED TO BE CALCULATED TO SOLVE THE FINAL LOF PROBLEM. YOU RARELY HAVE TO SOLVE THIS FOR EVERY POINT)

```
{
    "a": {
        "x": 0,
        "y": 0,
        "k_nearest_nodes_distances": {
            "b": 1,
            "c": 2
        },
        "local_outlier_factor": 0.8749999999999999,
        "local_reachability_distance": 0.6666666666666666
    },
    "b": {
        "x": 0,
        "y": 1,
        "k_nearest_nodes_distances": {
            "a": 1,
            "c": 1
        },
        "local_reachability_distance": 0.5,
        "local_outlier_factor": 1.3333333333333333
    },
    "c": {
        "x": 1,
        "y": 1,
        "k_nearest_nodes_distances": {
            "b": 1,
            "a": 2
        },
        "local_reachability_distance": 0.6666666666666666,
        "local_outlier_factor": 0.8749999999999999
    },
    "d": {
        "x": 3,
        "y": 0,
        "k_nearest_nodes_distances": {
            "a": 3,
            "c": 3
        },
        "local_outlier_factor": 2.0
    }
}
```


Wrap Up

I got a lot of views and requests on my implementation by hand article so I felt this was worth it.

Local Outlier Factor | Example By Hand

Local Outlier Factor value is a commonly used anomaly detection tool. It takes a local approach to better detect...

medium.com

I hope you enjoyed. If this performs well I will look into improving, optimizing, and packaging this up.

If you find any issues just tell me or open a pull request!

Data Science Python Analytics Science Math

About Help Legal

Get the Medium app