Local Outlier Factor | Simple Example By Hand

doedotdev
Apr 26, 2018 · 5 min read ★

Local Outlier Factor value is a commonly used anomaly detection tool. It takes a local approach to better detect outliers about their neighbors, whereas a global strategy, might not be the best detection for datasets that fluctuate in density.

Before we get started, I am going to assume you know a bit about DBSCAN and K Nearest Neighbor algorithms. However, I am confident you can make it through without needing an incredibly deep understanding of the algorithms.

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin...

en.wikipedia.org

k-nearest neighbors algorithm - Wikipedia

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and...

en.wikipedia.org

This is the implementation by hand, on a minimal and straightforward dataset. However, seeing it by hand I feel gives the best understanding initially.

Here is my implementation.

Local Outlier Factor | Simple Python Example

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

medium.com

. . .

Local Outlier Factor Calculation in 6 Steps

- Distance Calculation
- Kth-Nearest Neighbor Distance Calculation
- K-Nearest Neighbor Calculation
- Local Reachability Density (LRD) Calculation
- Local Outlier Factor Calculation
- Analysis

. . .

The Problem

Calculate the Local Outlier Factor (LOF) for each point and find the top 2 outliers. Use a “K” value of 2 and Manhattan Distance as the distance function.

```
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
```

Distance Calculation

Multiple distance functions out there, but as I specified in the problem above, we are going to be using Manhattan to allow easy calculations by hand.

How to calculate the Euclidean and Manhattan distance?

I have a practice problem that I am working on (artificial intelligence), but am unable to calculate the Euclidean and...

math.stackexchange.com

```
manhattanDistance(a,b) = 1
manhattanDistance(a,c) = 2
manhattanDistance(a,d) = 3
manhattanDistance(b,c) = 1
manhattanDistance(b,d) = 4
manhattanDistance(c,d) = 3
```

. . .

Kth-Nearest Neighbor Distance Calculation

As prescribed in the problem, we are going to use a K value of 2. This means we need to find the Kth, 2nd, nearest neighbor of each point. That is the 2nd closest point to each.

```
manhattanDistance(a,b) = 1
manhattanDistance(a,c) = 2
manhattanDistance(a,d) = 3

the second nearest neighbor of a is c.

Therefore,

kthNearestNeighbor(a) = c
kthNearestNeighbor(b) = c (a and c are same distance, choose either)
kthNearestNeighbor(c) = a
kthNearestNeighbor(d) = c (c and a are same distance, choose either)
```

Now calculate the distance of each point to it's kth, in our case 2nd, nearest neighbor.

```
distanceToKthNearestNeighbor(a) = manhattanDistance(a,c) = 2
distanceToKthNearestNeighbor(b) = manhattanDistance(b,c) = 1
distanceToKthNearestNeighbor(c) = manhattanDistance(c,a) = 2
distanceToKthNearestNeighbor(d) = manhattanDistance(d,c) = 3
```

K-Nearest Neighbor Calculation

Find the K, in our case 2, nearest neighbors of each value. You have already found the 2nd above, we just need the first in the set.

```
kNearestSet(a) = { b, c }
kNearestSet(b) = { a, c }
kNearestSet(c) = { b, a }
kNearestSet(d) = { a, c }

How many items are in each set?

kNearestSetCount(a) = 2
kNearestSetCount(b) = 2
kNearestSetCount(c) = 2
kNearestSetCount(d) = 2
```

You can do some of these steps in whatever order you like, however it will begin to make sense why I spell out the process.

. . .

Local Reachability Density (LRD) Calculation

LRD is the estimated distance at which a point can be found by its neighbors (NOT THE OPPOSITE, read that 2x). So if a neighbor were to reach out LRD value distance in any direction, it would be likely/ most optimal to find that individual point.
The LRD is the count of the items in the K nearest neighbor set which is calculated above as `kNearestSetCount(point)`, over the `reachDistance` of the point to all the values in it's set, which is `kNearestSet` above.

$$LRD(a) = \frac{kNearestSetCount(a)}{reachDistance(b <- a) + reachDistance(c <- a)}$$

What is reachDistance? The max value of the Kth, 2nd in our case, nearest neighbor of the point and the Manhattan distance of between the point and it's neighbor, two things we already calculated.

Here is an example of `b <- a`.

```
reachDistance(b <- a) =
       max{distanceToKthNearestNeighbor(b), manhattanDistance(a,b)}

reachDistance(b <- a) = max{1,1}

reachDistance(b <- a) = 1
```

Here is `c <- a`.

```
reachDistance(c <- a) =
       max{distanceToKthNearestNeighbor(c), manhattanDistance(a,c)}

reachDistance(c <- a) = max{2,2}
```

```
reachDistance(c <- a) = 2
```

We know `reachDistance`, we can complete the `LRD` calculation.

$$LRD(a) = \frac{kNearestSetCount(a)}{reachDistance(b \leftarrow a) + reachDistance(c \leftarrow a)}$$

$$LRD(a) = \frac{2}{(1 + 2)}$$

$$LRD(a) = .667$$

Calculate the LRD for the following points.

$$LRD(b) = \frac{kNearestSetCount(b)}{reachDistance(a \leftarrow b) + reachDistance(c \leftarrow b)} = .5$$

$$LRD(c) = \frac{kNearestSetCount(c)}{reachDistance(a \leftarrow c) + reachDistance(b \leftarrow c)} = .667$$

$$LRD(d) = \frac{kNearestSetCount(d)}{reachDistance(a \leftarrow d) + reachDistance(c \leftarrow d)} = .33$$

Remember reachDistance is different from LRD, and you need one to calculate the other!

. . .

Local Outlier Factor Calculation

The final `LOF` value of each point can now be calculated. The `LOF` of a point `p` is the sum of the `LRD` of all the points in the set `kNearestSet(p)` `*` the sum of the `reachDistance` of all the points of the same set, to the point `p`, all divided by the number of items in the set, `kNearestSetCount(p)`, squared.

```
Reminder calculated above:
kNearestSet(a) = { b, c }
kNearestSetCount(a) = 2

          [LRD(b) + LRD(c)] * [reachDist(b <- a) + reachDist(c <- a)]
LOF(a) = ------------------------------------------------------------
                kNearestSetCount(a)*kNearestSetCount(a)

LOF(a) = [.5 + .667] * [1 + 2] / (2 * 2)

LOF(a) = 3.501 / 4

LOF(a) = .87
```

Now that we see a full one worked out, I will quickly show the next 3 points `LOF`.

```
          [LRD(a) + LRD(c)] * [reachDist(a <- b) + reachDist(c <- b)]
LOF(b) = ------------------------------------------------------------
                kNearestSetCount(b)*kNearestSetCount(b)


LOF(b) = 1.33

          [LRD(b) + LRD(a)] * [reachDist(a <- c) + reachDist(b <- c)]
LOF(c) = ------------------------------------------------------------
                kNearestSetCount(c)*kNearestSetCount(c)

LOF(c) = .87

          [LRD(a) + LRD(c)] * [reachDist(a <- d) + reachDist(c <- d)]
LOF(d) = ------------------------------------------------------------
                kNearestSetCount(d)*kNearestSetCount(d)

LOF(d) = 2
```

. . .

Analysis

So what does this mean?

```
LOF(a) = .87
LOF(b) = 1.33
LOF(c) = .87
LOF(d) = 2
```

Well, it depends on the data.

While a `LOF` value of 1 or less is a good indicator of an inlier, we are here to calculate and **probably** remove outliers or anomalies.

Do you have a tight, clean, and uniform dataset? Then a `LOF` value of 1.05 could be an outlier.

Do you have a sparse dataset, varying in density, with many local fluctuations specific to that local cluster? Then a `LOF` value of 2 could still be an inlier.

So, it depends. There are many different variations/ additions to this base algorithm. However, we set out to find and prune the top 2 outliers from out set, which turned out to be b and d.

Any questions or mistakes, please comment.

. . .

More to come with anomaly detection with python and data science!

Data Science Data Data Analysis Data Analytics Python