Outlier Detection

February 10, 2021

Marco Ligia
Technological University Dublin
Ireland
B00139079@mytudlin.ie

Abstract
This paper focuses on distance-based outliers detection methods, reviewing basic principle of the classical approach, and one of its improvements that was developed for detecting outliers in very large datasets – sequential algorithm iOrca. The latter is using a simple indexing technique that makes the mining of outliers in sufficiently large data sets up to an order of magnitude faster compare to traditional approach that requires $O(n^2)$ computation time.

Keywords: Outliers, distance-based methods, iOrca algorithm, indexing technique, very large datasets.

1. Introduction

"Outlier" is the term used to denote an item that differs significantly from the overall average in a set or combination of data. One of the most important question in the problem of identifying outliers is an understanding of why we want to detect the outlier. In this sense, the context of detecting outliers is more important than the method of identification itself.
The question of the best method to approach an outlier comes to mind as soon as we start dealing with the outlier identification problem, but there is no a postulate that says "this technique is the best to detect an outlier". Firstly, it is important to understand why do we need to find the outlier. Therefore, the context of detecting outliers usually defines the technique itself.
Nonetheless, there are a number of methods to approach the outliers:

- **Sorting and Plotting.** Sorting your datasheet is a simple but effective way to highlight unusual values. Simple sorting of data sheet for each variable would make identification of unusually high or low values easier.

- **Statistical tests.** As a rule, statistical tests are used for individual features and extreme values are captured (Extreme-Value Analysis). For example, Z-value or Kurtosis measure can be used to identify if the dataset contains outliers. Here Kurtosis is the average (or expected value in the case of the pdf) of the Z-scores, each taken to the 4th power. Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution.
- **Model tests.** The idea is to build a model that describes the data (for example, linear regression model). Points that strongly deviate from the model (at which the model is very wrong) are anomalies, or outliers.
- **Machine learning methods.** A lot of machine learning algorithms suffer in terms of their performance when outliers are not taken care of. In order to avoid this kind of problem you could, for example, drop them from your sample, cap the values at some reasonable point (based on domain knowledge) or transform the data. But in turn, quality of many such methods, for example the Clustering method is measured by its

ability to detect some or all of the hidden patterns. Clustering can also be used to detect anomalies.

- **Iterative methods.** Methods that consist of iterations, at each of which a group of "especially suspicious objects" is removed. For example, in the n-dimensional feature space, you can remove the convex hull of our points-objects, considering its representatives as outliers. As a rule, the methods of this group are quite time consuming.
- **Metric methods.** Based on the number of publications, these are the most popular methods among researchers. They postulate the existence of some metric in the object space, which helps to find outliers/anomalies. It is intuitively clear that an outlier has few neighbours, while a typical point has many. Therefore, a good measure of anomaly can be, for example, the "distance to the k-th neighbour" (as in the Local Outlier Factor method), the density in the proximity of a point, etc. Specific metrics can be used in these methods, such as Mahalanobis distance which is popular in the case of multivariate outliers detection as this distance takes into account the shape of the observations.

2. Distance-based outlier detection method.

Distance-based outlier detection method belongs to the last group of methods from the previous chapter, the Metric methods. This method could be seen as reviewing the neighbourhood of an object, which is described by a given radius (distance). An object is considered an outlier if its neighbourhood does not have enough other points.
Distance-based outlier detection is the most studied, researched, and implemented method in the area of unsupervised learning. There are many variants of the distance-based methods, based on sliding windows, the number of nearest neighbours, radius and thresholds, and other measures for considering outliers in the data.
Most algorithms of this type take the following parameters as inputs:

- Window size *w*, corresponding to the fixed size on which the algorithm looks for outlier patterns.
- Sliding size s, corresponds to the number of new instances that will be added to the window, and old ones removed.
- The count threshold k of instances when using nearest neighbour computation.
- The distance threshold R used to define the outlier threshold in distances.

The outputs are outliers that can be thought of as labels or scores (based on neighbours and distance).
If we wish to approach this problem more formally, then first we could define an outlier score for each point *x* in dataset *D* as the distance to the k-th nearest neighbour of x in D or, it can be defined as sum of the distances of the k nearest neighbours of x in D, where D is distance. Then the outlier detection problem is as follows: given parameters t and k, find the top t points in D in terms of their outlier scores1.
In this form the statement of the outlier detection task was used in many research fields to find anomalies such as Seismology, astronomy, cosmonautics and more.
However, the problem of distance-based outlier detection is difficult to solve efficiently in very large datasets because of potential quadratic time complexity. The traditional nested loop distance-based algorithm requires O(n2) (with n = |D|) computation time. This is infeasible for large datasets. Because of it, distance-based methods have a number of "improvements", such as **Orc**a algorithm, which maintains the t-th largest outlier score and use it as a cut off threshold. Another one is helps in the nearest neighbour search for outlier detection. Knorr and Ng present an idea of mapping the existing data in order to split them into multiple cells first, that they

called hyper rectangles and only the corners of the minimum bounding rectangles (MBR), and then store the data for the corners of the minimum bounding triangles to save the disc space. As soon as test is needed along the way, all MBRs are pruned, which means they cannot contain the nearest neighbours of the test point. However, the complexity of this approach only increases with growing dimension of the initial dataset, and it increases exponentially with the dimension of the dataset. This leaves the approach inefficient for large datasets again.

3. Improvement of the existing algorithm – iORCA – fast outlier detection using indexing

In the Orca algorithm mentioned in the previous section, the pruning rate is achieved by carefully selecting the "cutoff threshold". If the cutoff grows very slowly, it does many redundant comparisons for inliers. To mitigate this problem, the authors of iOrca (indexed Orca) doe as follows:

- the cutoff is updated faster,
- the data is rearranged in a way that for every point, the nearest neighbours are found in approximately constant time,
- it does not perform unnecessary disk accesses to make sure the previous steps are completed,
- the index is built quickly and does not need the entire data to be kept in memory.

The authors realised that for the cutoff updates to run faster, the outliers should be processed as early as possible. Hence, in the proposed indexing method, firstly a random point R from D as a reference point is selected, and then calculate the distance of all the other points in distance D from R. Then the points in the database are reordered with increasing distance from R.

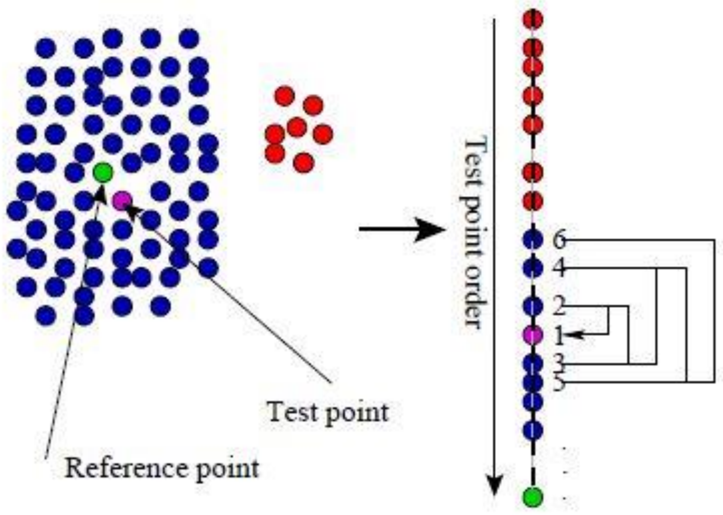


Figure 1: Explanation of the indexing process. The figure on the left shows a dataset, where the "normal" elements are shown in blue, and the outliers are in red. The reference and test points are also shown. The figure on the right shows the order in which the test points are processed. Points farthest from the reference point being processed first. – from (Bhaduri et al.)

Fig. 1 shows the index for a dataset. The left part shows the original dataset. The blue circles represent the inlier points while the red ones denote the outliers. The green circle at the center

is the randomly chosen reference point. The right part of the figure shows the index. The index is formed as a one dimensional list of ordered points where the ordering is determined by the distance of all the dataset's points measured from the reference point (i.e., the distances are sorted from largest to shortest). This way they avoid going through the whole unordered set. The points are tested along this index order. This is done to achieve the following effect:
The random choice of R, makes it likely that R will be one of the inlier points if there are more inliers than outliers (just because there are more inliers than outliers). Hence, there is a better chance that the points farthest from R will be the outliers. Because the processing of points is organised in a way that the distance to R is decreasing, the probability of the outliers being processed first will be high. This leads to a faster growth of the cutoff threshold. So, the first requirements stated above is satisfied. Re-ordering the dataset for testing is advantageous for more than that though. For ordered points, it is intuitively clear that iOrca can be stopped and terminated much earlier compared to other state of the art methods by using the following termination criterion, that the authors call the Stopping rule:
$\|x_t - y\| + \|x_{(n-k+1)} - y\| < c;$
where y is the reference point, c is the current cutoff threshold defined by user, x_t is any tested point, and $x_{(n-k+1)}$ is the k-th nearest neighbour of y. $\|*\|$ is the 2-norm of a vector. If the inequality is satisfied, then iOrca can be terminated.

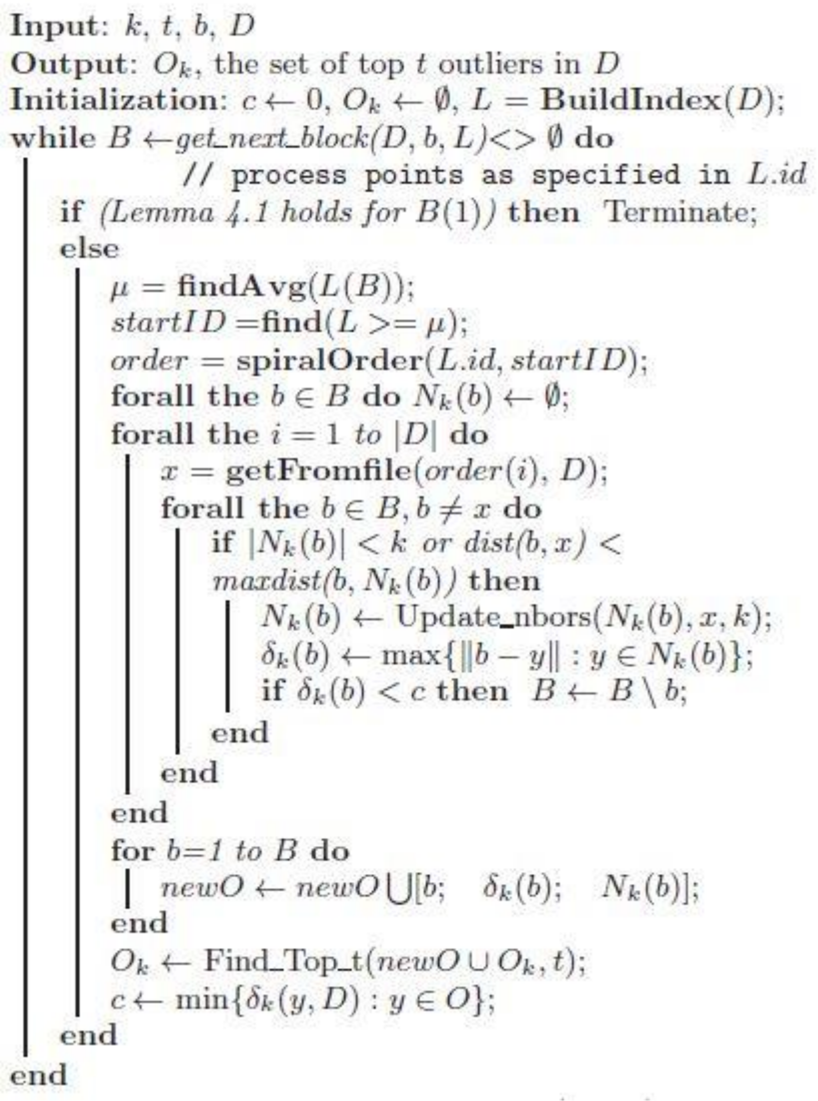
For this method, upon indexing the distances are pre-computed and available in memory. This ensures that checking if the condition holds is very fast and causes very little overhead. To satisfy the second requirement stated earlier, the search for the nearest neighbours of a test point should be done in a constant amount of time. For this, the authors again apply the index. The search starts every time not from the start of the dataset for every test point, but from the location it lies along the index and then gradually "spreading" out.
Figure 1 shows one of the test points T (given in magenta) in the dataset. It also shows where T lies along the index. So, the search for the nearest neighbour starts out from point T by with first checks made at the nearest two points (top and bottom) and then continuing further until the score of that point drops below the threshold (when this is satisfied, point is pruned) or the test point has a score higher than the t-th largest outlier (in which case the outlier list and the cutoff are updated). Since the index prescribes a total ordering of the points projected along one dimension, it is likely that, for a test point x_t, if x_i is closer to x_t than x_j along the index, then it will be so when the actual distances are also computed i.e. if

$\|x_t - y\| > \|x_i - y\| < \|x_j - y\|$, and from this it can be expected that

$\|x_t - x_i\| > \|x_t - x_j\|$.

This resulst in a fast search of the nearest neighbors, as later was demonstrated by experimental study on very large datasets. The suggested method maps a way of testing each point by traversing the dataset points in a spiral order. Loading the entire dataset for testing each point one by one is "I/O expensive". So, the authors suggested processing the test points in blocks. This way they can be tested simultaneously by creating a spiral search order for that entire test block. The indexing scheme allows to do this by loading a block of index distances (they are the distances of the test points to y), computing the average distance and then starting the spiral search from the location where this average lies along the index. Since after ordering, the test blocks are expected to be similar, testing them using the same spiral order helps speed up the search as well.

Finally, the indexing intuitively simple, and cheap to compute. It can be entirely loaded in memory at runtime since it only consists of *n* floating point numbers and the associate integer indices. iOrca (short for indexed Orca) was estimated to be upto an order of magnitude faster than Orca and costs less disk I/O. The pseudo-code of iOrca is shown in Fig. 2 (Algorithm 2). The important moment is the choice of R which is critical in determining the effectiveness of the pruning.



```
Input: $k$, $t$, $b$, $D$
Output: $O_k$, the set of top $t$ outliers in $D$
Initialization: $c \leftarrow 0$, $O_k \leftarrow \emptyset$, $L$ = BuildIndex($D$);
while $B \leftarrow$ get_next_block($D, b, L$) <> $\emptyset$ do
            // process points as specified in $L.id$
    if (Lemma 4.1 holds for $B(1)$) then Terminate;
    else
        $\mu$ = findAvg($L(B)$);
        $startID$ = find($L >= \mu$);
        $order$ = spiralOrder($L.id, startID$);
        forall the $b \in B$ do $N_k(b) \leftarrow \emptyset$;
        forall the $i = 1$ to $|D|$ do
            $x$ = getFromfile($order(i), D$);
            forall the $b \in B, b \neq x$ do
                if $|N_k(b)| < k$ or $dist(b, x) <$
                $maxdist(b, N_k(b))$ then
                    $N_k(b) \leftarrow$ Update_nbors($N_k(b), x, k$);
                    $\delta_k(b) \leftarrow \max\{\|b - y\| : y \in N_k(b)\}$;
                    if $\delta_k(b) < c$ then $B \leftarrow B \setminus b$;
                end
            end
        end
        for $b$=1 to $B$ do
            $newO \leftarrow newO \bigcup [b;\ \delta_k(b);\ N_k(b)]$;
        end
        $O_k \leftarrow$ Find_Top_t($newO \cup O_k, t$);
        $c \leftarrow \min\{\delta_k(y, D) : y \in O\}$;
    end
end
```

Fig. 2. Pseudocode of Algorithm 2: Indexed Orca (iOrca) – From (Bhaduri et al.)

Complexity analysis was performed on the proposed algorithm. Clearly, for building the index, *n* distance computations are needed, each taking O(d) time. As the authors stated from the experiemnts, the sorting takes O(n log n) time, giving a total running time of O(n log n+nd). n floating point numbers are required to be stored as the index, so the space requirement is O(n). The disk space is also preserved, and even more so if iOrca is terminated early.

4. Experiments.

It was stated by the authors, that iOrca produced exactly the same outliers on the tested datasets, as those obtained from Orca. Hence the accuracy was not contested at the time. The testing was performed on the following real datasets:

“1. Covertype: contains 581,012 instances and 10 features
2. Landsat : contains 275,465 instances and 60 features
3. MODIS: contains 15,900,968 instances and 7 features
4. CarrierX: contains 97,814,864 instances and 19 features”.

The following default values for the parameters were used: block size, b = 1000, t = 30 and k = 5.

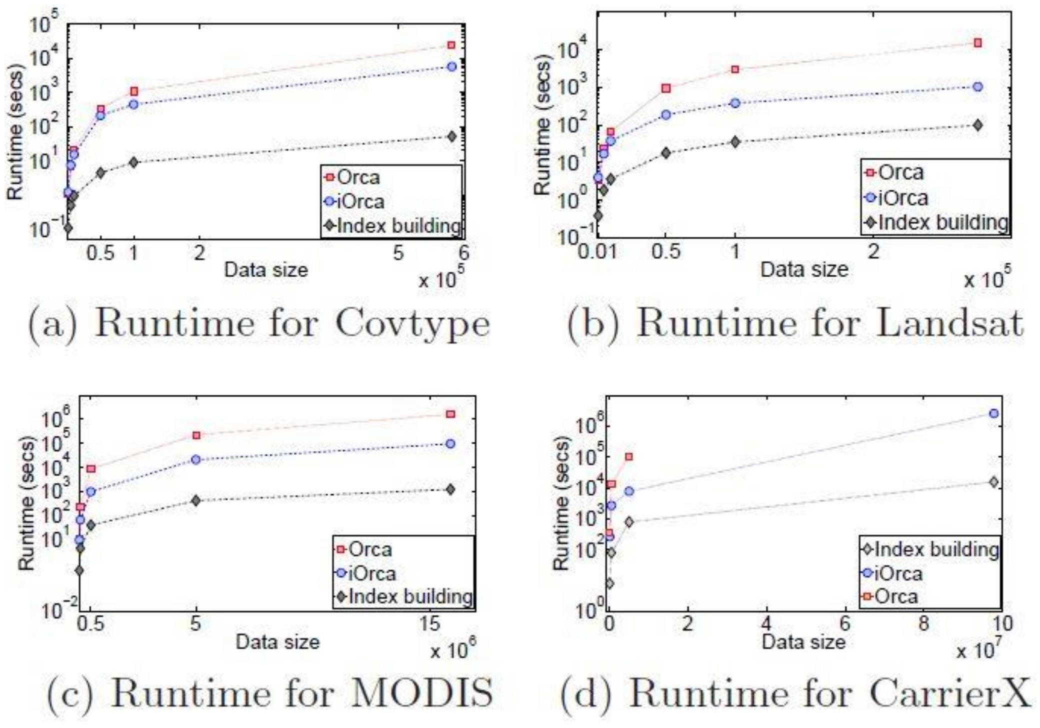


Figure 3: Figure showing running time for the 4 datasets ((a) - (d)). As clearly shown for each dataset, iOrca beats Orca in running time. From (Bhaduri et al.)

Fig. 3 ((a)–(d)) presents the results for run-time for the 4 datasets. Each of the Figures (a) - (d) shows the running time (in log scale) along the y-axis for Orca (red), iOrca (blue circles) and the index building phase of iOrca (black diamond). The x-axis are random samples from these datasets. From the figures it is clear that iOrca outperforms Orca; “the decrease in running time is from 2 to 17 times compared to Orca”.

The decrease in runtime occurs because: (1) the indexing method orders the points in such a manner that those which are farthest from a common reference point are processed first. Since these points are the potential outliers, the cut-off increases much faster (compared to Orca), leading to faster pruning of inliers the spiral spread helps to find the nearest neighbours for each test point much faster than going through the data in the original order, and the early stopping criterion helps to avoid testing all the data points.

iOrca was repeatedly tested and provided a benchmarking for various indexed schemes [, yet its performance was rarely contested (Fig.4). Experimental results demonstrated that it is more accurate than existing distance-based definitions of outliers. Various researcher commended its build index process as very simple and virtually not having cost time for building indexing scheme. The only reported problem was iORCA’s pivot selection method (Xu et al., 2016).

However, responding to this issue new Algorithms still used probabilistic or approximate outliers density approach.

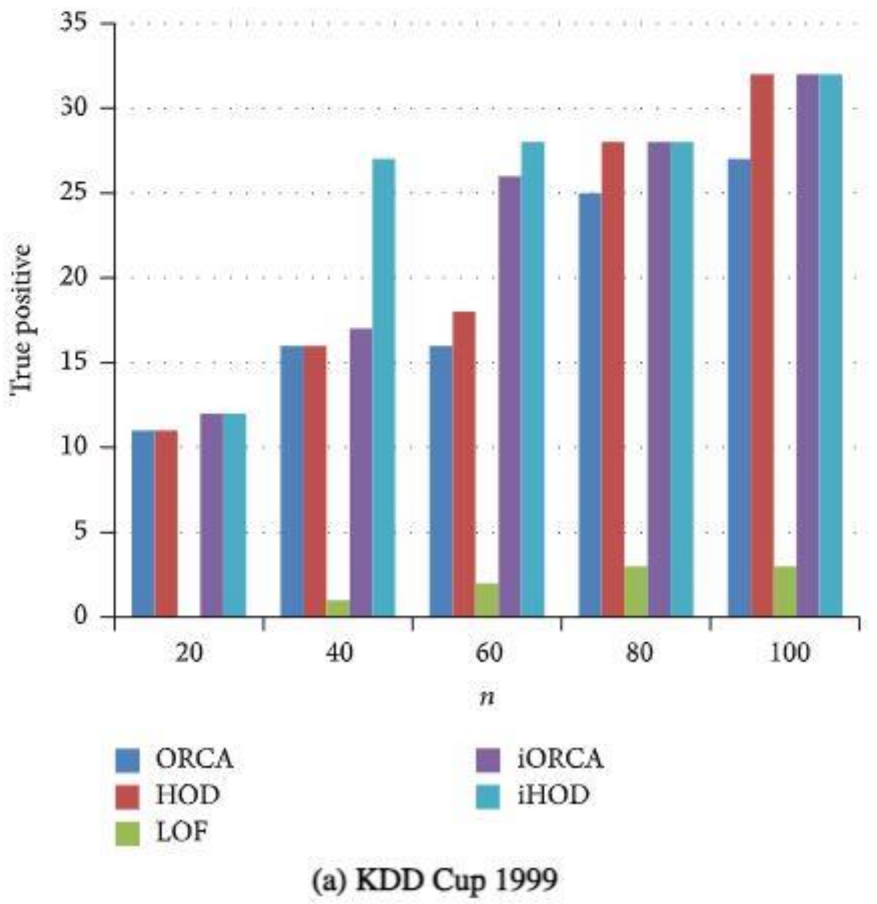


Fig.4 – Comparison by “True positive values” with various values n of five algorithms for KDD dataset – from [Xu et al., 2016].

5. Conclusions

In this paper methods of distance-based outlier detection were considered. As it is follows from the sources, the distance-based methods can be significantly improved via correctly defined indexing scheme that allows for early and sufficient pruning of dataset. On a minus-side, the algorithm makes several probability-based choices, from which is can be deducted that there is a possibility that the testing may go wrong. However, the tests performed by the authors on a very large datasets all demonstrated the superiority of the scheme of indexed Orca (iOrca), and at the same a good agreement of the known benchmarks.

References

[1] Bhaduri, K., Matthews, B., & Gianella, C., Algorithms for speeding up distance-based outlier detection, Proceedings of the 17th ACM SIGKDD international conference on Knowledge, 2011.

[2] Xu,H., Mao,R., Liao, H., Zhang, H., Lu, M., & Chen, G., Index Based Hidden Outlier Detection in Metric Space, Scientific Programming, vol. 2016, Article ID 8048246, 14 pages, 2016. https://doi.org/10.1155/2016/8048246.

[3] Ninghao Liu, Donghwa Shin, Xia Hu, Contextual Outlier Interpretation, Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), 2018.

[4] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, Jörg Sander, LOF: Identifying Density-Based Local Outliers, ACM SIGMOD RecordMay 2000.

[5] Lewison, Outliers detection with Isolation Forest, https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e

[6] Knorr & Ng, Algorithms for Mining Distance-Based Outliers in Large Datasets, Proceedings of the 24th VLDB Conference New York, USA, 1998.

[7] T. Fuertes, Outliers: looking for a needle in a haystack, 2016, https://quantdare.com/outliers-looking-for-a-needle-in-a-haystack/