Anomaly Detection using Voronoi k-distance

1st Sofia Jones
Department of Computer Science
University of Northern British Columbia
Prince George, Canada
sjones2@unbc.ca

2nd Weixian Lan
Department of Computer Science
University of Northern British Columbia
Prince George, Canada
wlan@unbc.ca

3rd Jonathan Shaw
Department of Computer Science
University of Northern British Columbia
Prince George, Canada
shaw5@unbc.ca

***Abstract*—We investigate computational methods for generating Voronoi partitions. Using the Voronoi cell data, we are able to perform density-based outlier detection by means of the Voronoi k-distance measure. When optimally implemented, this approach has a linearithmic time complexity.**

***Index Terms*—Voronoi, outlier detection, k-distance, density-based outlier mining**

I. Introduction

A. Voronoi Diagram

Given a metric space (X, d), let $(P_k)_{k=1}^{N}$ be an ordered subset of X with some positive integer N. Moreover, define the *distance from a point $x \in X$ to a set $A \subset X$* by

$$d(x, A) := \inf \{d(x, a) : a \in A\}.$$

Then we define the *Voronoi region* R_k by the set

$$R_k = \{x \in X : d(x, P_k) \leq d(x, P_i)\ \forall i \neq j\}.$$

Informally, the Voronoi diagram is a partition of space into N disjoint regions for which, in each region R_k, one is closer to the datapoint P_k than to any other datapoint. For this investigation, we will be limiting our discussion to Euclidean 2-space.

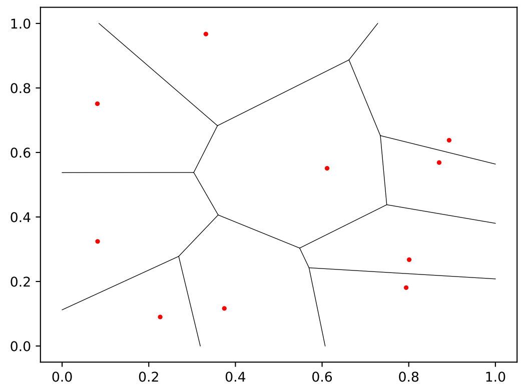

Fig. 1. A voronoi diagram for 10 points in Euclidean 2-space

B. Nearest Neighbor

For a given point P_i, we say that P_j is a *nearest neighbor* of P_i if

$$d(P_i, P_j) = \min_{l \neq i} d(P_i, P_l).$$

That is, P_j is closer to P_i than is any other datapoint. We can extend this definition and say that the *k nearest neighbors* of P_i are the k datapoints $P_{j1}, P_{j2}, ..., P_{jk}$ such that

$$d(P_i, P_j m) = \min_{\substack{l \neq i \\ l \neq j_1, j_2, \ldots, j_{m-1}}} d(P_i, P_l).$$

Informally, the k nearest neighbors of P_i are those k points that are closest to P_i.

C. Outlier Detection

Outlier detection is the practice of finding observations that are the most different from all the others [1, p. 55]. Outlier detection is a complex field with a number of approaches being employed. Common techniques often fall under the following categories:

- Distribution-based: How are the points distributed? Do they follow a normal distribution pattern?
- Distance-based: How far away are points from each other or from a central datapoint?
- Depth-based: Does the datapoint fit inside a certain boundary? If not, it may be an outlier.
- Density-based: How close together is a group of datapoint? Points in sparse regions may be considered outliers.

Our investigation focuses density-based anomaly detection. When performing density-based anomaly detection, one must consider the following:

1) The basic approach to density-based anomaly detection requires calculating the k-distance neighbourhoods of all points. This by itself has a time complexity of $O(N^2)$.
2) When there is variation on density distributions, i.e one cluster of datapoint is sparse and another is dense, outliers may not be accurately uncovered.

There has been a significant effort to amend these shortcomings, but they are somewhat inherent to density-based approaches. Nevertheless, we will explore a density-based way to detect anomalies with a more performant algorithm.

II. Background Related Work

A. Voronoi Diagram Generation

Many methods exist to generate a Voronoi diagram given a set of N points [2]. Some known ones include but not limited to Fortune's algorithm, Bowyer-Watson algorithm, Delaunay triangulation and Lloyd's algorithm. Some of these implementations run with time complexity $O(N \log N)$, which has been proven to be optimal [3, p. 212]. The algorithm we implemented is something of a brute-force method, with time complexity $O(N^2)$. We will discuss this implementation in Part III.

B. Density Based Outlier Detection

An early approach to density-based outlier detection, known as *local outlier factor* (LOF), shares similarities with both kNN classification and density-based clustering algorithms such as DBSCAN. The LOF algorithms assigns to each datapoint a score LOF_k of how "outlier-like" it is based on its relative position to k of its nearest neighbors [4]. Computing LOF_k for every one of the N points in the data set, however, has a time complexity of $O(N^2)$, since there is no immediate way to get the k nearest neighbors of a given point without iterating over the dataset. As we will explore, the Voronoi diagram is the key to constructing an approach that is similar to LOF but vastly more efficient.

C. Voronoi k-distance

As discussed, we can generate the Voronoi diagram for a given N-point dataset in linearithmic time. As it turns out, the Voronoi diagram gives us the information we need to efficiently assign an "outlier-ness" score to each point. To understand how it works, we must discuss an important fact about the Voronoi diagram:

- If k is less than or equal to the number of edges for a given Voronoi region R_i, then each of the k nearest neighbors of P_i defines an edge of the Voronoi region R_i [3, p. 207].

This is a very powerful result. Since every two adjacent Voronoi regions in the diagram share exactly one edge [3, p. 220], we can examine all the m edges of the region R_i to find set $V_{k-NN}(P_i)$ of the k nearest neighbors of P_i. The computational cost of computing the k nearest neighbors of P_i via this method is only $O(\log N)$ [3, p. 220].

Armed with an efficient method of computing the k nearest neighbors of each point, we now wish to assign a measure of outlier-like characteristics to each datapoint. As defined in [5], the *Voronoi k-distance* $V_{k-d}(P_i)$ of a point P_i is

$$V_{k-d}(P_i) = \sum_{Q \in V_{k-NN}(P_i)} \frac{d(P_i, Q)}{k}$$

This real-valued, positive number represents the average distance between P_i and its k nearest neighbors. The bottleneck in computing this value is finding $V_{k-NN}(P_i)$ in logarithmic time. Hence, to find the $V_{k-d}(P_i)$ for every point P_i, we must:

1) Generate the Voronoi diagram for $(P_i)_{i=1}^{N}$ ($O(N \log N)$)
2) Compute $V_{k-d}(P_i)$ for each P_i (N times $O(\log N)$, or $O(N \log N)$)

We conclude that it is possible to compute the Voronoi k-distance for every point in linearithmic time, which is a substantial improvement compared to the quadratic complexity of computing LOF_k for every point.

III. Implementation

Our codebase consists of two components that are independent of one another. First, to obtain a better understanding of Voronoi diagrams and their associated data structures, we built our own Voronoi diagram generation algorithm. Second, using an existing implementation of the Voronoi algorithm for the sake of efficiency and polish, we implemented an outlier-detection program based on the Voronoi k-distance.

A. Voronoi Diagram Generation

Though efficient algorithms are appreciated for real applications, we chose a not-so-efficient but rather simple algorithm [6] to implement for the purpose of this project. The main idea is as follows: the input consists of the points that define the Voronoi diagram, which are referred to as *sites*. *Edges* are those line segments that separate one Voronoi region from another. Each site will have its own Voronoi region, and while that region is updating through iterations, we call it a *cell*. A cell consists of its site and edges, and will turn into an actual Voronoi region at the end of the program. At the beginning, we pick two sites and enclose each with a large cell. Now we start our iterative process on every site. For each of the remaining sites, we dedicate a cell to it, and we find its edges by separating this site from any other site that has its own cell.

Though we were able to grasp the big picture of this algorithm, we faced many challenges in our implementation. First, the data structures required for all the attributes such as edges and cells are very complex. They are nested lists, and while they are iterated on, operations such as insertion, deletion or modification might be performed. Second, when implementing from scratch as we did, we had to implement our own functions for many computational geometry problems. An example is a function that deduces the spatial relationship of a line and a point in a bounded rectangle. Third, with complex data structures, debugging for this program is extremely difficult. Since we had very limited time for this, we still could not detect the bug in the program which produces incomplete edge sets for some inputs (we were, however, able to confirm the correctness of all the edges that *are* present). The code is not perfect: some functions were hard coded but straightforward as if we were doing it with pencil and paper. If we were given more time to improve our implementation, we would employ some functions from the existing libraries rather than build them from scratch.

B. Voronoi k-distance

Given a dataset and a value k, our outlier-detection program builds a Voronoi diagram for the given dataset and assigns each point a Voronoi $k - distance$ value which is represented visually on a color scale. The main output is a figure:

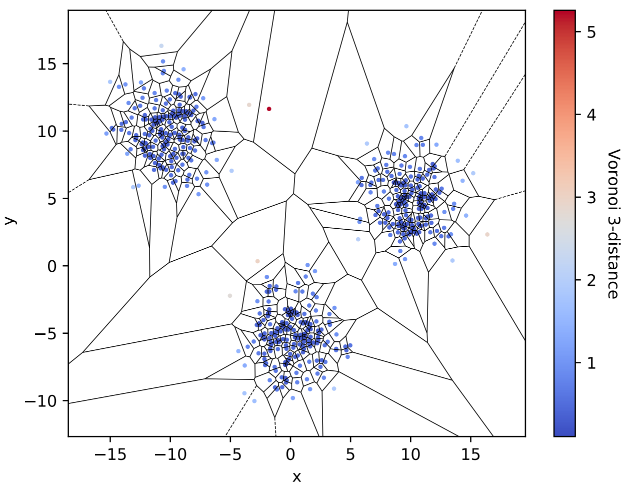

Fig. 2. A sample output for a toy dataset of clustered points (k=3)

In addition to the figure, the program outputs a .csv file containing the following columns:

- An index representing the datapoint
- The x and y coordinates for the datapoint
- The perimeter of the datapoint's Voronoi region
- The Voronoi k-distance of the point
- The "`outliervalue`", which is simply the k-distance rescaled to the range $[0, 10] \in \mathbb{R}$
- The "`outlierclass`", which reads "`outlier`" if the point is in the 95th percentile of "`outliervalue`" and "`not outlier`" otherwise. The 95% threshold is an arbitrary choice, but we feel it is useful for most datasets.
- A list of the indices of the point's Voronoi neighbors
- A list of the edges corresponding to a point's cell

Due to time constraints and the general difficulty of computing the area of n-gons, we were not able to include the area of each Voronoi cell in our output file. However, given some time, this could certainly be added. The output file is sorted in descending order by the magnitude of the `outliervalue`, so points that are the most outlier-like appear earlier in the file. These entries correspond to the points that appear red in color on the output.

C. Methodology

We built the project in Python using an incremental agile approach. We intertwined planning with developing to allow us the flexibility to explore different outlier detection methods. We developed our project in increments as follows:

1) The first iteration simply generated randomized datasets
2) The second iteration generated Voronoi diagrams from random data using existing libraries
3) In the third iteration, we experimented with outlier detection based on infinite regions. The idea was to remove infinite edges and then create a smaller boundary to remove further infinite edges until all outliers could be found. However, this did not prove to be a reliable method as some infinite edges were not outliers at all. From there we thought of looking at areas of Voronoi regions, however area did not prove to be an efficient or fully effective method either.
4) The fourth and final iteration implemented outlier detection by means of the Voronoi k-distance. This iteration also included our very own from-scratch program for generating Voronoi diagrams. This method is the most intuitive and seems to fit very well with Voronoi diagrams. K-distance is an effective way to detect anomalies with performance constraints that can be alleviated by using Voronoi diagrams. Figure 1 shows an output from our from-scratch program.

We were able to test at each stage with the help of *matplotlib*—a powerful data-visualization tool.

D. Datasets

Numerous datasets were used to detect outliers. A brief description of each dataset will be given for reference.

1) Ahmedabad and Air Quality: The `ahmedabad.csv` and `air_qual.csv` datasets contain air quality information samples from various Indian cities [7]. A list of pollutants are measured with an associated air quality index value and a classification of air quality. We used data that had been classified with values of poor, moderate and good to find outliers in those clusters. We chose to investigate the pollutant PM2.5 and how it affects the air quality index. Particulate Matter 2.5(PM2.5) are tiny inhalable particles that are particularly damaging to respiratory function. These fine particles have found to be more damaging than larger particles, PM10, and have a considerable impact on air quality index. Ahmedabad contains only data for one city. On the other hand, `air_qual.csv` is an accumulation of all cities and is classified by Moderate, Good, Poor and Very Poor air quality. This allows us to work will a real large dataset with inherent clusters. A k value of 2 to 4 is recommended for this dataset. There are only 6 classifications so a k value any higher than 6 no additional groups will be generated.

2) Clusters: This dataset has easily defined clusters that can show how our diagrams can handle clusters of data. It fits very nicely with a density based approach to outlier detection.

3) Letter: This dataset was taken from a Harvard unsupervised learning benchmark [8]. It has a fairly even spread and is a good test for us to see if our model can detect outliers in a spread out dataset.

4) Loan: Two datasets were taken from a bank datasets of individuals and companies applying for loans [9]. `loan_count.csv` compares loan counts with completion rates. The more loans the lower completion rate it would seem. However this was always the case in this dataset. `loan_type.csv` grouped the type of customer with their completion rate of loans. Companies with their own income were most likely to complete the loan. A value of k can be supplied for this dataset and a group will be created for each increment of k.

5) Airplane Crashes: This is an almost linear dataset [10] that is a great test for our density based approach. We are still able to find outliers despite the linear dataset. This proves the versatility of our anomaly detection.

IV. Evaluation

A. Voronoi k-distance for different k

Interestingly, the value of k seems to have extremely little effect on the outlier-detection functionality of the algorithm. For the dataset `ahmedabad.csv`, we found that the algorithm detects 68 outlier points for $k = 2$, $k = 3$, and $k = 4$. While the magnitude of the k-distance larger for larger k, the actual distribution of the k-distance (and hence the `outliervalue`) values is nearly identical for various k. Moreover, the list of points sorted by `outliervalue` appears in approximately the same order regardless of k.

The performance of the algorithm does not vary as a function of k: in order to find the k-nearest neighbors for a point P_i, we must compute *all* the Voronoi neighbors of P_i irrespective of k.

B. An observation about Voronoi neighbors

As discussed, Voronoi neighbors always share exactly one edge, and no other points share that edge with either of them. During our research, we began to wonder the following question: for an N-point Voronoi diagram, is there a lower bound on the number of edges a Voronoi region may have? While we did not find a theorem to answer this question, we did notice that for large n, the lower bound seems to be 3 edges.

C. Algorithm shortcomings

The Voronoi k-distance method of outlier detection suffers from the same problems as any other density-based outlier detection system. In particular, it tends to register false positives when cluster density varies.

Notice in figure 3 that the top rightmost data point is seen as an outlier, even though it appears consistent with the linear trend. This is because this region is sparse compared to the more-densely populated region in the bottom right. In reality, the datapoints underneath the imaginary line would probably be considered more "outlier-like" than the former.

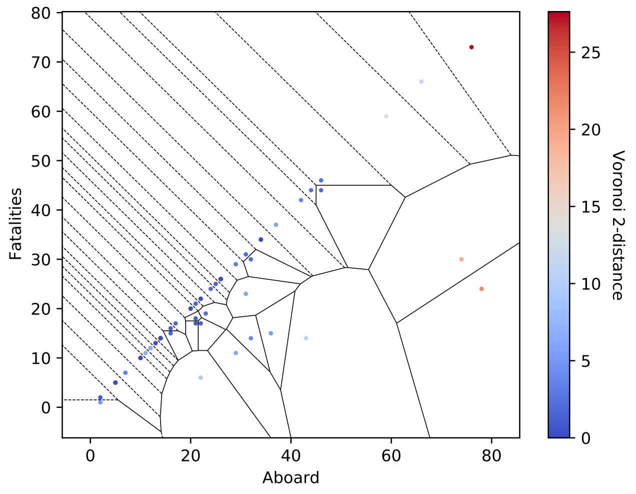

Fig. 3. A dataset in which the algorithm registers a false positive

V. Conclusion

In this discussion, we have only scratched the surface of Voronoi-based approaches to outlier detection. In particular, we have limited ourselves to Euclidean 2-space. However, with a bit of work, the next logical step would be to generalize our algorithm and results to both higher-dimensional Euclidean space and other metric spaces. The reader who wishes to gain more information on general Voronoi partitions may want to consult Preparata and Shamos' exhaustive 1985 treatment of computational geometry [3]. For more information on the Voronoi k-distance, one should consult with Jilin Qu's seminal 2008 conference paper on the topic [5]. The utility of the Voronoi diagram in outlier detection also extends beyond density-based approaches: several authors have discussed using the Voronoi diagram in model-based and statistical approaches [11], [12]. In conclusion, the Voronoi diagram is a fascinating data structure which seems to have a great deal of utility in a variety of data-mining fields. In particular, we believe that the Voronoi k-distance is an underappreciated method of outlier detection. We feel that this topic deserves additional research, and—who knows—maybe one of us will be the person to conduct it in the future.

References

[1] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, *Introduction to Data Mining*, Second edition. NY NY: Pearson, 2019, ISBN: 978-0-13-312890-1.

[2] R. Klein, *Concrete and abstract Voronoi diagrams*, ser. Lecture notes in computer science 400. Berlin: Springer, 1989, 167 pp., OCLC: 20800464, ISBN: 978-3-540-52055-9 978-0-387-52055-1.

[3] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*, ser. Texts and monographs in computer science. New York: Springer-Verlag, 1985, 390 pp., ISBN: 978-0-387-96131-6.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, pp. 93–104, May 2000, ISSN: 0163-5808. DOI: 10.1145/335191.335388. [Online]. Available: https://doi.org/10.1145/335191.335388.

[5] J. Qu, "Outlier detection based on voronoi diagram," in *Proceedings of the 4th International Conference on Advanced Data Mining and Applications*, ser. ADMA '08, Chengdu, China: Springer-Verlag, 2008, pp. 516–523, ISBN: 9783540881919. DOI: 10.1007/978-3-540-88192-6_51. [Online]. Available: https://doi.org/10.1007/978-3-540-88192-6_51.

[6] S. Wolfman. (2000). "Algorithm for generation of voronoi diagrams," University of Washington Computer Science & Engineering, [Online]. Available: https://courses.cs.washington.edu/courses/cse326/00wi/projects/voronoi.html (visited on 12/11/2020).

[7] (Aug. 2020). "Air quality data in india (2015 - 2020)," Kaggle.com, [Online]. Available: https://kaggle.com/rohanrao/air-quality-data-in-india (visited on 12/12/2020).

[8] M. Goldstein, "Unsupervised anomaly detection benchmark," Oct. 6, 2015, Publisher: Harvard Dataverse type: dataset. DOI: 10.7910/DVN/OPQMVF. [Online]. Available: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/OPQMVF (visited on 12/12/2020).

[9] R. Pilliard Hellwig, *Mse survey open data (2018) [csv]*, Sep. 2019. DOI: 10.6084/m9.figshare.9848153.v1. [Online]. Available: https://figshare.com/articles/dataset/MSE_Survey_Open_Data_2018_CSV_/9848153/1.

[10] (Sep. 2016). "Airplane crashes since 1908," Kaggle.com, [Online]. Available: https://kaggle.com/saurograndi/airplane-crashes-since-1908 (visited on 12/12/2020).

[11] M. Luis, F.-T. Arsene, N. Laurent, and M. Schoenauer, "Anomaly detection with the voronoi diagram evolutionary algorithm," *arXiv:1610.08640 [cs]*, Oct. 27, 2016. arXiv: 1610.08640. [Online]. Available: http://arxiv.org/abs/1610.08640 (visited on 12/10/2020).

[12] C. E. Zwilling and M. Y. Wang, "Multivariate voronoi outlier detection for time series," in *2014 IEEE Healthcare Innovation Conference (HIC)*, Seattle, WA, USA: IEEE, Oct. 2014, pp. 300–303, ISBN: 978-1-4673-6364-8. DOI: 10.1109/HIC.2014.7038934. [Online]. Available: http://ieeexplore.ieee.org/document/7038934/ (visited on 12/10/2020).