

Learned indexes for multidimensional data using Z-order-curve

Andris Prokofjevs

Master-Basismodul in Department of Informatics – Database Technology

Supervisor:

Dr. Sven Helmer

Zurich, January 2020

Andris Prokofjevs
Cullmanstrasse 26, 8006, Zurich
Tel.: +41 077 816 48 98
Email: andrisprokofjevs@gmail.com
Matriculation number: 18-796-755

Table of Contents

1. Introduction.....	3
2. Background.....	3
2.1. Learned index.....	3
2.2. Multidimensional data with z-order.....	3
3. Multidimensional learned indexes.....	4
4. Evaluation.....	7
5. Conclusion & Outlook.....	8
References.....	9
Table of Figures.....	9

1. Introduction

This work addresses learned index approach to the database indexing of multidimensional data. Indexing of a database is a vital process in order to speed up query evaluation. Classical approaches use index structures like B+ Trees or R+ Trees. However, as good as these methods proved themselves to be, they ignore the dependency between data distribution and indexes. Learned index approaches address this dependency and provide a possibility to learn them for further use in query performance optimization.

This work attempts to discover and use patterns hidden in multidimensional data distribution. Goal of this paper is to explore possibilities Z-Order curves and machine learning give us to learn indexes of different datasets.

The paper is structured as follows: In the second section background concepts of learned indexes and z-order curve will be presented. In the section 3 usage of learned indexes for multidimensional data is discussed and a few examples are provided. Section 4 is concentrated around the evaluation of proposed solution. Finally, the work is concluded and some outlook is provided in section 5.

2. Background

Before moving forward, it is important to note some basic concepts around which this work is build.

2.1. Learned index

Learned index is a term proposed in (Kraska, Beutel, Chi, Dean, & Polyzotis, 2018). Authors suggest that existing index structures can be seen as models to map a key to the position of corresponding record. Learned index in their definition is an index structure where classical approaches are replaced with other models, especially machine learning ones. The key reasoning for this suggestion is the ability of machine learning models to learn order, structure and distribution of the lookup keys and efficiently predict expected index of a record. Important to mention is, that (Kraska, Beutel, Chi, Dean, & Polyzotis, 2018) concentrated on 1-dimensional data for their work on learned indexes. This poses one crucial difference: 1-D data can be sorted in a sequential order, what cannot be done with multidimensional data. For this purpose Z-order-curve is used.

2.2. Multidimensional data with z-order

In the work of (Wang, Fu, Xu, & Lu, 2019), while referring to (Kraska, Beutel, Chi, Dean, & Polyzotis, 2018), authors proposed their solution for using learned indexes with n-dimensional data. They named it learned Z-order Model, which essentially is a combination of Z-order curve and learned index.

Z-order curve as it is put in (Wang, Fu, Xu, & Lu, 2019) is “a space filling curve ... that maps a multidimensional vector space to a 1D space. It provides a linear order for all data points in the multidimensional space. Each multidimensional vector can be converted into a unique integer called Z-address”. Z-address is computed by bitwise

interleaving of the input points. More information on the process of computation can be found in (Tropf & Herzog, 1981).

Important for this work are the features of the Z-order-curve, which we will use further. While providing linear order, Z-order-curve introduces some rapid value increases in Z-values even in places where input data may not differ so much. Example for that would be the difference in Z-values of 3 following points: (0, 510), (0,511), (0,512). The corresponding Z-values for these points will be: 174760, 174762 and 524288 respectively. As we can clearly observe there is a relatively big value jump in Z-value after a monotonous increase prior the jump. Values after the jump also increase monotonically until next jump.

These jumps can be visually seen in the **Error! Reference source not found.** In general, it follows a pattern of a jump every $2^n * k$ values of y, where n is the integer representing the power of 2 and k is an integer representing how many jumps are generated with resulting y value. For example, if we look on y-axis in Figure 1, first jump of z-value is introduced in 2^0 (1st) value of y, where following along the first column of y axis z-value jumps from 0 to 2. Second jump occurs in 2^1 (2nd) value of y, with z-value jumping from 2 to 8. Next jump occurs in 2^2 (4th) value of y: z-value goes from 10 to 32. The 3rd value of y also makes a jump, which is smaller, than the one in 4th and can be located by adding k to the above used formula. That way at least smaller jumps like this occur every $2^1 * k$ values of y. Thus every 1st, 2nd, 3rd and so on values.

In general, the bigger is the n, the bigger jump Z-value takes. And k represents kth jump of a determined by n amplitude. This characteristic of Z-order-curve is crucial for further work.

3. Multidimensional learned indexes

Goal to be achieved is to find a function that could predict an index based on the given input values. In order to reach this goal firstly x and y are converted to z-values, giving a linear order of values, that can be sorted and stored in the database. This provides a guarantee that $\text{index}(z\text{-value}_1) < \text{index}(z\text{-value}_2)$, given that $z\text{-value}_1 < z\text{-value}_2$. Next step is to learn a function, which when providing z-value as an input, gives out approximated index of this z-value as output. To put it another way, dependencies between the z-values and index of records in the database should be learned, so that we could approximate an index of a particular z-value, without looking for its index among all possibilities, but by using learned relationships. Thus, a process is needed that could act like a function to convert z-value into its index, based on the learned indexes of already existing z-values.

For the purpose of this work following 3 datasets are used: generated uniform distribution of x and y - Figure 2, generated gaussian x and y distribution – Figure 3 and a real world data, which represents a world map of a surface, where the coordinates of points represent elevated non-ocean surface of the earth - Figure 4.

After the transformation of this datasets into Z-values the distributions for respective data are generated. These are shown in Figure 5, Figure 6 and Figure 7. As we can see, while Z-value distribution for uniform data is a continuous line, gaussian distribution as

well as world map data distribution have multiple levels and value jumps. These bigger jumps are the result of the blank spaces in the data distribution in x and y space. For gaussian dataset these are 4 clearly distinguished regions, which corresponds to 4 corners in $x - y$ space of gaussian dataset with no values in it.¹ At the same time in gaussian data smaller bumps of z -values along continuous lines could be seen, which are not explained by blank spaces of $x - y$ data. These bumps are rather the feature introduces by Z -order-curve that was explained earlier: the jumps appear every $2^n * k$ values of y . In the world map dataset same jumps of both natures can be observed, however it is not so easy to visually identify the connection between regions on $x - y$ and z -value visualizations.

To get a reference point to compare with, linear regression is fitted for all datasets, which obviously gives good results for uniform distribution dataset, however shows itself not so effective for other datasets. (See Figure 8, Figure 9 and Figure 10.) Evaluating this result, it gets obvious for one, that the technique proposed in (Wang, Fu, Xu, & Lu, 2019) and named Multi-staged Model proves itself extremely helpful. Multi-staged Model essentially suggests separating data into multiple groups where index values could be searched and predicted by classical search algorithms and linear regressions.

To perform this, separation is done with help of machine learning algorithms. In the example of (Wang, Fu, Xu, & Lu, 2019) neural network was used for this task. In the process of this research variety of strategies were considered e.g. value shift clustering and mean shift clustering. However, first successful prototype was built by using hierarchical clustering.

The final algorithm functions as follows. For the example dataset world map, x and y values are converted into z -values as a first step. Next, a hierarchical clustering algorithm is applied to the dataset, using x and y coordinates as inputs, to identify densely populated areas. This allows to create clusters by excluding blank spaces seen in $x - y$ dimensions. The visualization of clusters created for world map dataset can be seen in Figure 11. In Figure 11, 9 clusters can be observed. This number of clusters was determined based on previously build dendrogram visualization. See Figure 12.²³ An example visualization of z -value distribution in one of produced clusters is presented in Figure 13. Z -value jumps can be clearly observed in Figure 13. These jumps are the

¹ 5 jumps and 4 continuous regions can be distinguished. 4 jumps correspond corners of $x - y$. 5th jump is believed to be due to a few coordinates seen in the bottom of the $x - y$ gaussian visualization. Z -values for these points should be significantly larger than for other points, which explains the final 5th jump without a long continuous line after it.

² Distance between vertical lines in dendrogram represents distance between clusters, which these vertical lines represent. From top – large distance / few clusters, to bottom – almost no distance / every point is a cluster on its own. In order to determine number of clusters with dendrogram, we make an imaginary horizontal line across the vertical lines where meaningful distances start to occur. Afterwards the number of times a vertical line crosses the horizontal ones should be used as number of clusters. This is highly subjective matter on where to draw a line and depends highly on specific dataset and goals of a research.

³ Determining number of clusters involves human judgment and can be seen as a downside of this clustering approach, however every clustering method requires some type of input from user. Drawing a line could be seen as relatively simple task even for an unexperienced user. Also, it could potentially be automated using threshold values to determine which distances between clusters are acceptable and which are not and draw a line if the threshold is exceeded. Nevertheless, human perception is seen as preferable, because of the need of adjustment every time a new dataset comes into consideration.

same ones we saw in e.g. gaussian distribution visualization in Figure 6 as relatively small. They are a result of the Z-order-curve feature described in section 2.2.⁴

Clearly such cluster as represented in Figure 13 cannot be efficiently used for linear regression prediction. For this reason, next level of clustering is implemented. To do so, locations of jumps have to be identified. The jump between index values 400 and 600, observed in Figure 13 will be used as an example. Figure 14 represents the same jump, but in form actual data. The change in Z_value as well as Database_index can be clearly distinguished. X value changes rapidly as well, but it happens multiple times across the dataset and occurs not always in a connection with Z_value and Database_index rapid increases. Despite Y is changing just a little, it is the point of interest. If one calculates where the jump is supposed to be, according to formula from section 2.2, she will find out, that one of the jumps has to happen after the y value 8192. ($2^{13} * 1$) This value lays between y values 8118 and 8194, where jump represented by Figure 14 occurs. In the similar way other jumps can be located. E.g. the jump seen in Figure 13 between index values of 0 and 200 is actually happening between y values 4090 and 4261. Using the formula exact y value after which jump is expected is $2^{12} * 1 = 4096$. If these two y values⁵ are used as upper and lower bounds new smaller cluster can be produced. Visualization of this cluster is shown in Figure 15. Over distribution of this form a linear regression can be learned and efficiently used.

So, in order to find all relevant Z-value jumps based on y value boundaries algorithm iteratively goes through all clusters generated by hierarchical clustering and determines max and min values for y in these clusters. For the cluster visualized in Figure 13 maximal y value is 2623 and minimal y values is 12186. These 2 values represent y boundaries for this cluster. Next it determines maximal n for formula in section 2.2. so that resulting y value is still in boundaries of the class. Here it is $n = 13^6$. If the n would be 14, then jump is approximated to be in y value 16384, which is above upper y boundary of the parent cluster.

Next the cluster is divided into sub-clusters. This happens based on the knowledge, that each 2^{13} values of y a jump occurs. So, the first sub-cluster is determined as all records of actual cluster, which have y values less or equal than 2^{13} . As a result, a first sub-cluster is set with y boundaries of 2623 (minimal y value we had in parent cluster) and 8192 (an y value where jump occurs). Next sub-cluster continues the pattern and sets minimal y value as a maximal y value of previous sub-cluster + 1. In this example it will be 8193 as minimal y value for second cluster. The upper boundary is determined depending on whether or not the next jump occurs within the boundaries of parent cluster. In this example upper boundary of parent cluster is 12186. The next jump if $n = 13$ is on $2^{13} * 2 = 16384$. So, the upper boundary for second sub-cluster will be set to 12186, because 16384 exceeds the upper boundary of 12186 of a parent cluster. In the case if upper boundary of parent class would be greater than 16384, 16384 would be

⁴ Important to mention for this step of a process is, that clustering as such, cannot predict a cluster of a new record. Clustering only can arrange existing points into separate clusters. For this reason, a classification algorithm of k-nearest-neighbors is learned on top of the hierarchical clustering. In summary a classification algorithm is trained over the records that one already has. As a second step a classification algorithm is trained to predict a cluster of a record, given its x and y values. This way, each new record undergoes the process of classification in order to be assigned to one of the clusters identified by clustering algorithm

⁵ 4096 and 8192

⁶ $2^{13} = 8192$

used as upper boundary for second sub-cluster and third sub-cluster would be generated with minimal y value of $16384 + 1$.

After the sub-clusters are determined a linear regression is trained over each of them and R^2 of these linear regressions evaluated. If the R^2 is greater than 0.8 the cluster is seen as good enough and is written in the list of final clusters.⁷ In the case of R^2 being less than 0.8, another round of sub-clustering occurs, with n set to $n-1$. In the above example n was 13, so in the following round it will be set to 12. In this way smaller clusters are produced, eliminating additional jumps and allowing R^2 to rise, due to greater linearity of Z -values.

If no $R^2 > 0.8$ can be achieved with $n > 9$. Algorithm stops sub-clustering. The reason for that is the absence of convergence of R^2 towards higher values in these sub-clusters of $n \leq 9$. Enormous number of sub-clusters is produced if process is not stopped, without reasonable improvement of accuracy. The nature of this not converging clusters have to be evaluated in the future.

To summarize: Firstly, dataset is converted from x - y space to z -value space. Secondly, clusters are identified with hierarchical clustering algorithm. Next, sub-clusters are identified with usage of Z -value-curve property, which provides jumps every $2^n * k$ of y values. If the generated sub-cluster is good enough it is saved. If not, further search is conducted, but only until $n \geq 10$.

A new record when needs to be assigned to an index, firstly undergoes the transformation from $x - y$ to z -value space. As a second step, a classification algorithm is used to determine the cluster of the record based on its x and y values. In the third step, a sub-cluster with respective y boundaries is found and linear regression model of this sub-cluster is used for a prediction of index value for this record.

Worth to mention are the edge case scenarios. Two of them are possible: First – the record does not fall into any of the sub-clusters created using y as z -value jump values as boundaries. In this case a sub-cluster with nearest boundary value to the y value of a record should be used. Second - If the record is classified by k -nearest-neighbors into the wrong cluster, most likely a subclass with suitable y boundaries will be found and index value will be predicted based on this sub-cluster. Both scenarios are expected to produce a prediction of index value with greater error.

4. Evaluation

The evaluation is performed on gaussian distribution and world map datasets. Reason for excluding normal distribution dataset is the relative simplicity of a distribution. A linear regression trained over this dataset without any transformations and/or clustering provides $R^2 = 0.99$.

Both datasets have 10.000.000 records and were clustered with 9 clusters. In the world map (gaussian distribution) dataset, there are 31 (160) sub-cluster linear regression models. 13 (10) of them have R^2 above 0.8. Average R^2 for all models is 0.77. (0.46)

⁷ Threshold of 0.8 is subjectively chosen as good enough, because higher values of threshold produce unreasonably high amount of sub-clusters, without substantially improving prediction accuracy.

Total amount of space on the hard drive taken by models is 592KB (1.7MB). Distributed as following: Table storing cluster names and boundaries, which is used for choosing right model based on the input record values – 4KB (16KB), k-nearest-neighbors model – 464KB (464KB), all sub-cluster linear models take 124KB (1.23MB) of space on a hard drive.

Computer running the algorithm has Intel Core i7-8550U 1.8GHz CPU and 8G RAM, runs Windows 10 Home Version 1809. Code is written in python 3.7.

A prediction of a query with 10.000 records were executed in 0.8766 (2.1348) seconds, and yielded average absolute prediction error of index of 223358 (854521). This means on average model predicts 223358 (854521) indexes either up or down from the actual true index. To put it in perspective, average % error of a prediction is also calculated. To do so, absolute error is divided with the number of possible indexes in dataset, which is 10.000.000. Result is 2.23% (8.54%) error rate.

The statement of (Wang, Fu, Xu, & Lu, 2019) in their work that “a typical B-tree costs $O(\log n)$ lookup time and $O(n)$ index memory size, while a highly tuned learned model has the potential to achieve $O(1)$ time complexity and $O(1)$ index memory size.” is true for this work as well. Same as for their neural network solution, this clustering solution has potential of achieving $O(1)$ time. However, this is extremely dependent on the dataset provided. The linear Z-value distribution of uniform distribution dataset can potentially reach $O(1)$ with both approaches relatively easily, but the more complex the input dataset gets, the more time will be needed to predict indexes learned on it.

5. Conclusion & Outlook

In this work an alternative learned index approach for multidimensional data using Z-order-curve was presented. Although it showed itself relatively good for world map dataset it also gave less accurate results for gaussian dataset. Exact reasoning for this difference should be further investigated. Possible reason is the high density of points in gaussian distribution in the middle of x – y space, which does not allow a proper clustering in this region.

Nevertheless, the results suggest the potential usage of this or similar approach being possible for the reduction of search space by indicating the window of most likely range of indexes given particular z-value as an input. What in turn should reduce search time for conventional data structures. This work also shows, that results are highly dependent on the distribution of x – y in the input dataset. With more linear and less concentrated datasets yielding substantially better results.

As a future work, it would be of high interest to compare the performance of this solution with conventional B+ trees and the solutions of (Kraska, Beutel, Chi, Dean, & Polyzotis, 2018) and (Wang, Fu, Xu, & Lu, 2019). Also, further improvement and investigation on algorithm presented in this paper should be done. First of all, investigating if there are other features of Z-order-curve, apart from Z-value jumps dependent on y values, that could be used to improve accuracy. Secondly it may be possible to introduce one more layer of machine learning models, learned on the errors made by linear regressions.

References

- Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. (2018, Apr 30). The Case for Learned Index Structures.
- Tropf, H., & Herzog, H. (1981). Multidimensional Range Search in Dynamically Balanced Trees.
- Wang, H., Fu, X., Xu, J., & Lu, H. (2019). Learned Index for Spatial Queries.

Table of Figures

Figure 1	10
Figure 2	10
Figure 3	11
Figure 4	11
Figure 5	12
Figure 6	12
Figure 7	13
Figure 8	13
Figure 9	14
Figure 10	14
Figure 11	15
Figure 12	15
Figure 13	16
Figure 14	16
Figure 15	17

	0	1	2	3	4	5	6	7
0	0	1	4	5	16	17	20	21
1	2	3	6	7	18	19	22	23
2	8	9	12	13	24	25	28	29
3	10	11	14	15	26	27	30	31
4	32	33	36	37	48	49	52	53
5	34	35	38	39	50	51	54	55
6	40	41	44	45	56	57	60	61
7	42	43	46	47	58	59	62	63

Figure 1

Z-value represented along the x and y axes.

X and Y axes represent x and y values in spatial data. Values in the table are the Z-values

Source: https://en.wikipedia.org/wiki/Z-order_curve

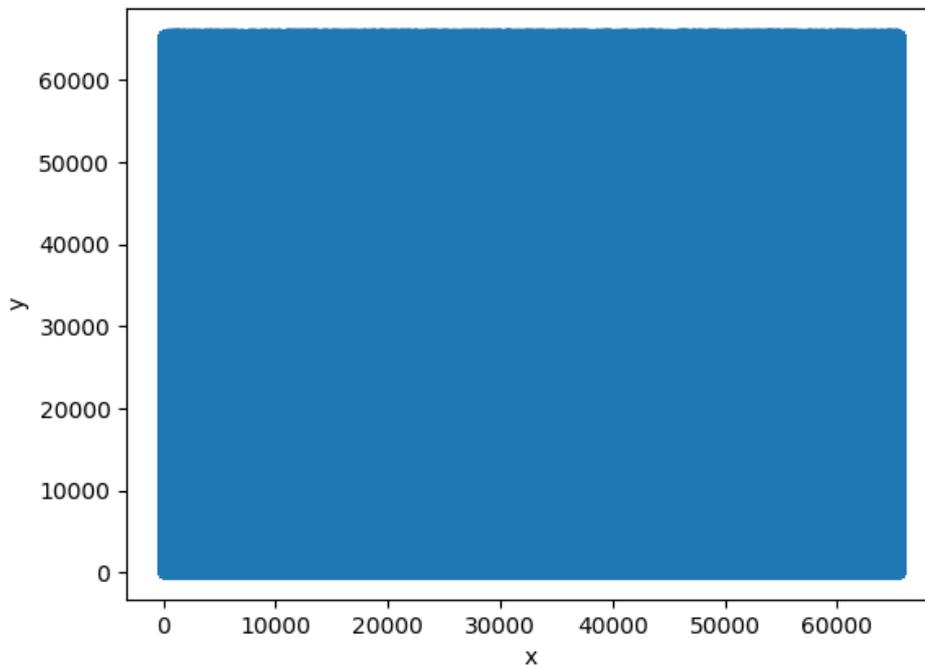


Figure 2

Visualization of a gaussian uniform dataset, using x and y axes.

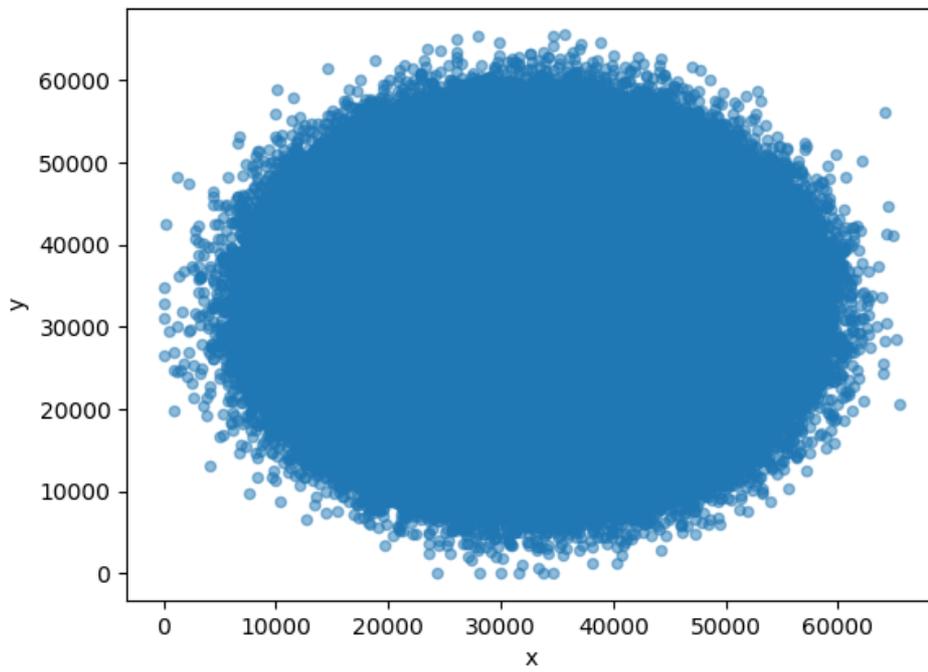


Figure 3

Visualization of a gaussian distribution dataset, using x and y axes.

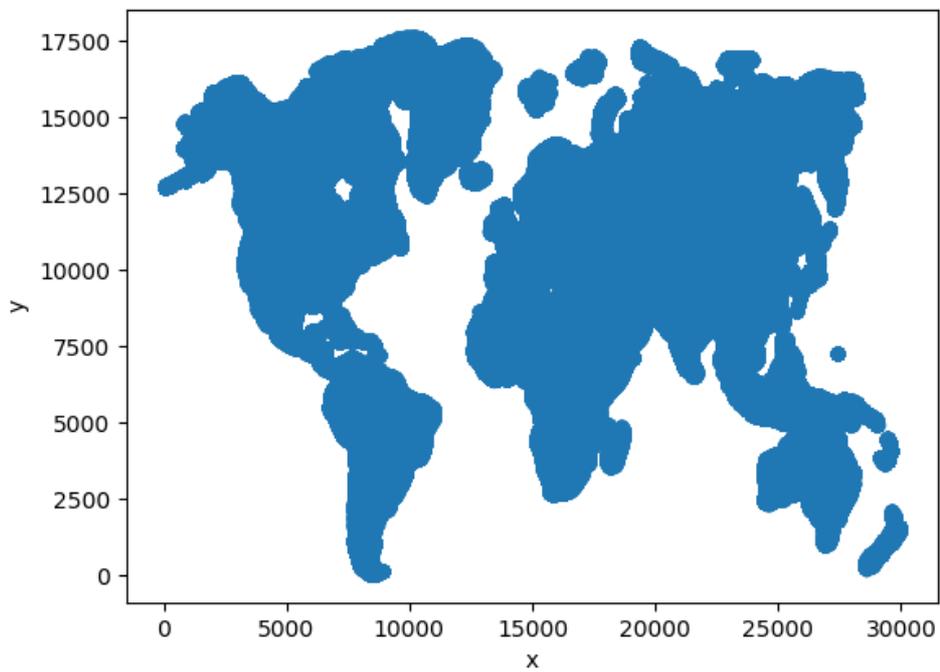


Figure 4

Visualization of a world data dataset, using x and y axes.

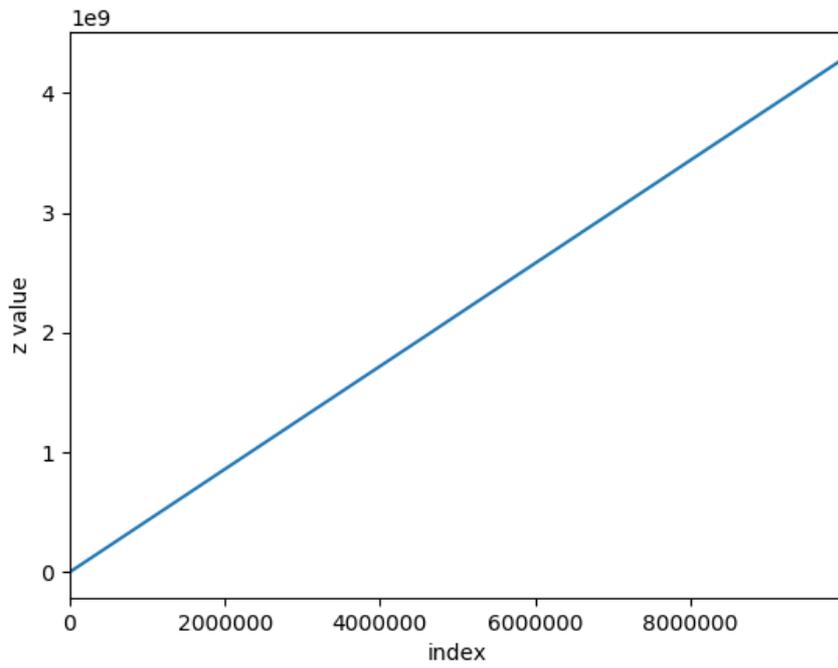


Figure 5

Z-value distribution over the indexes of a uniform distribution dataset.

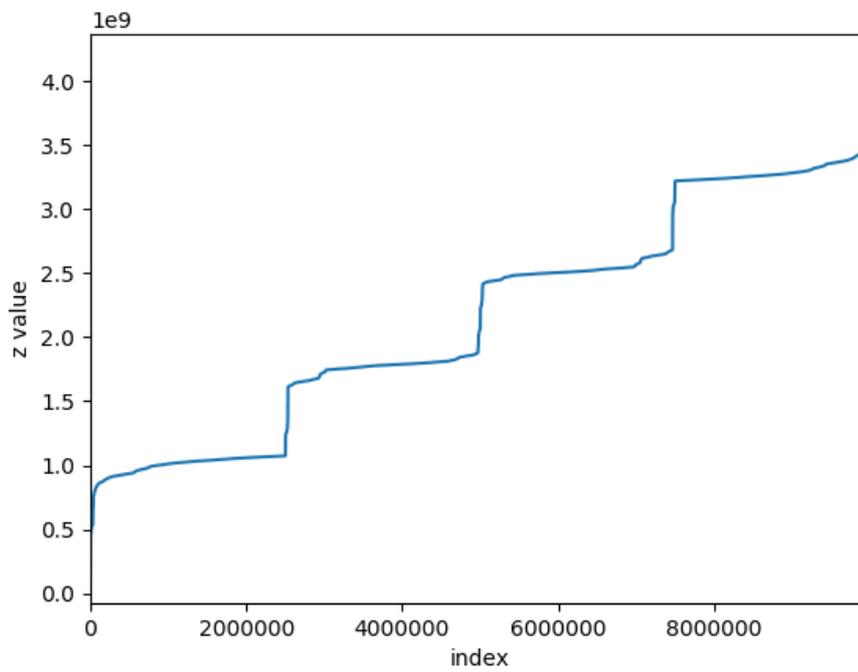


Figure 6

Z-value distribution over the indexes of a gaussian distribution dataset.

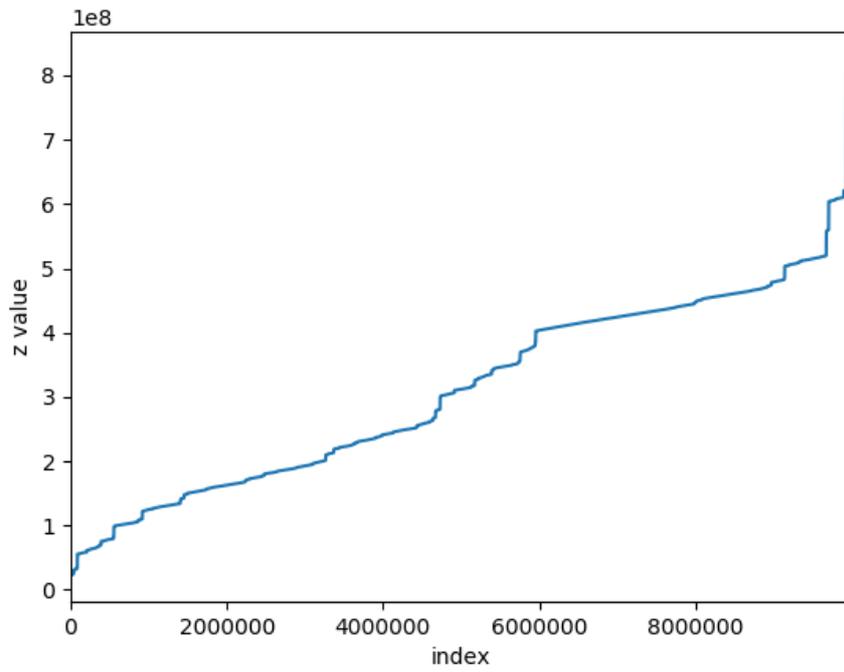


Figure 7

Z-value distribution over the indexes of a world map dataset.

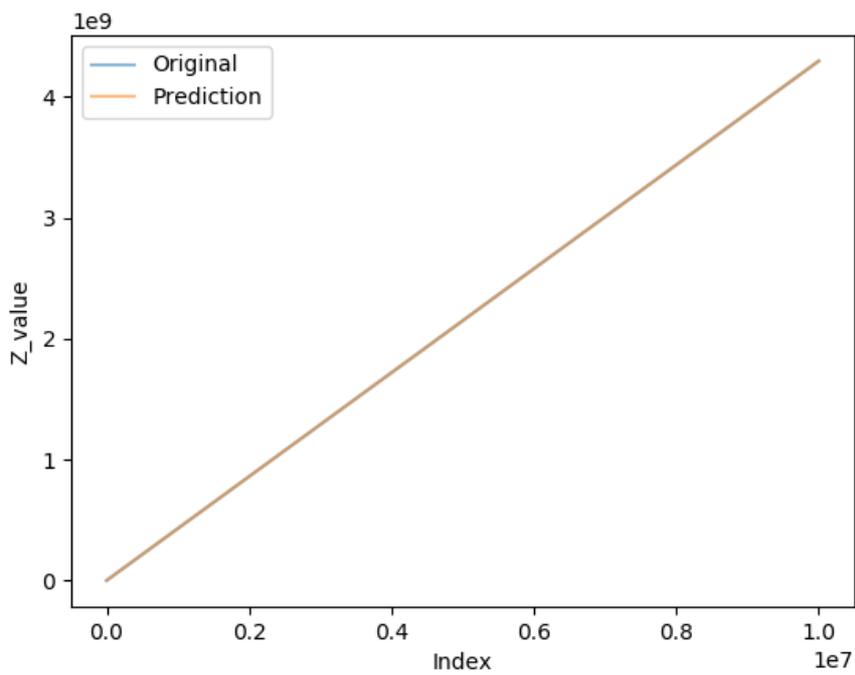


Figure 8

Z-value distribution over the indexes of a uniform distribution dataset. (blue line)
 Z-value distribution over the predicted indexes of a uniform distribution dataset. (orange line)

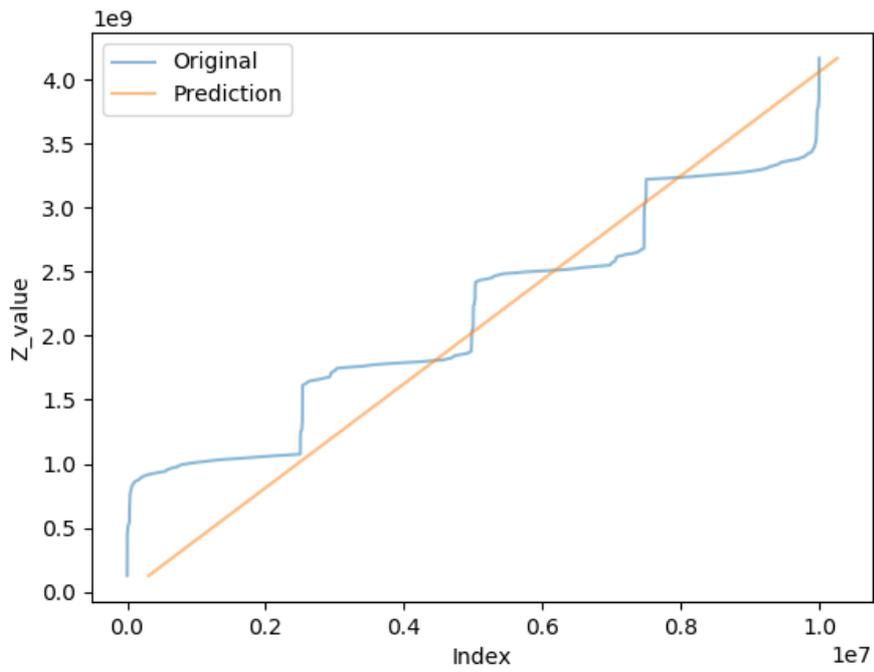


Figure 9

Z-value distribution over the indexes of a gaussian distribution dataset. (blue line)
 Z-value distribution over the predicted indexes of a gaussian distribution dataset. (orange line)

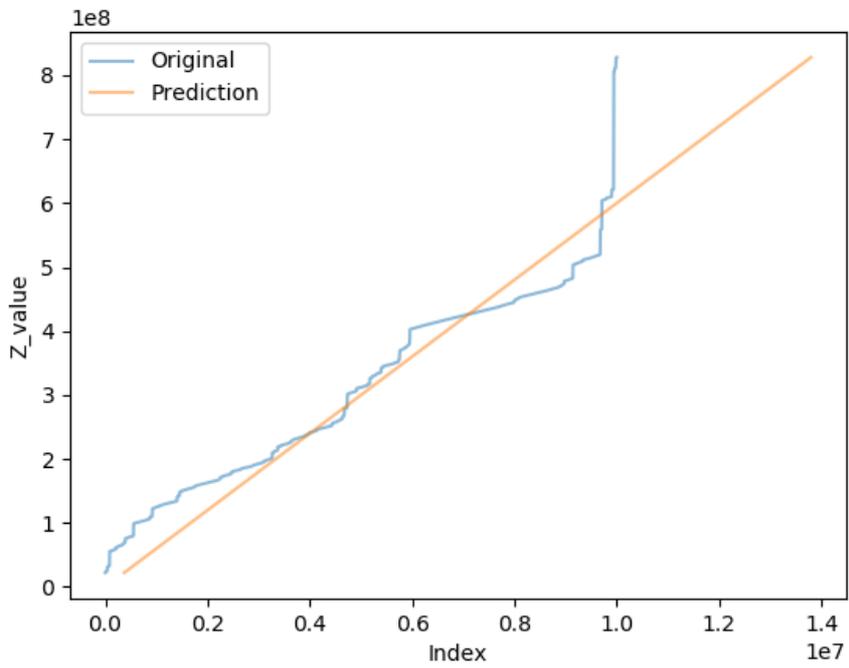


Figure 10

Z-value distribution over the indexes of a world map dataset. (blue line)
 Z-value distribution over the predicted indexes of a world map dataset. (orange line)

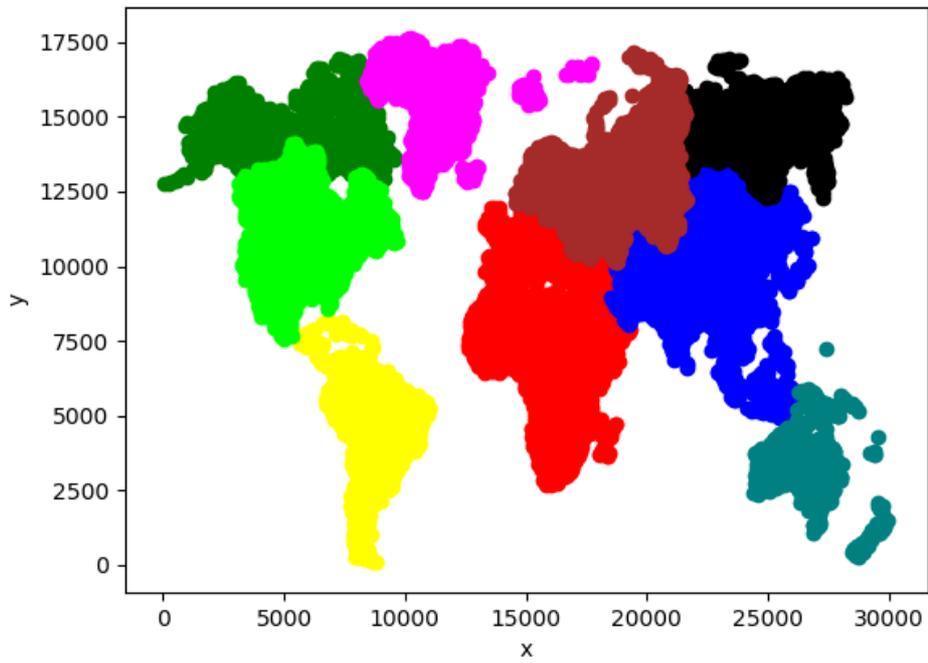


Figure 11

Visualization of a world data dataset, using x and y axes. Colored for the representation of different clusters found with hierarchical clustering algorithm.

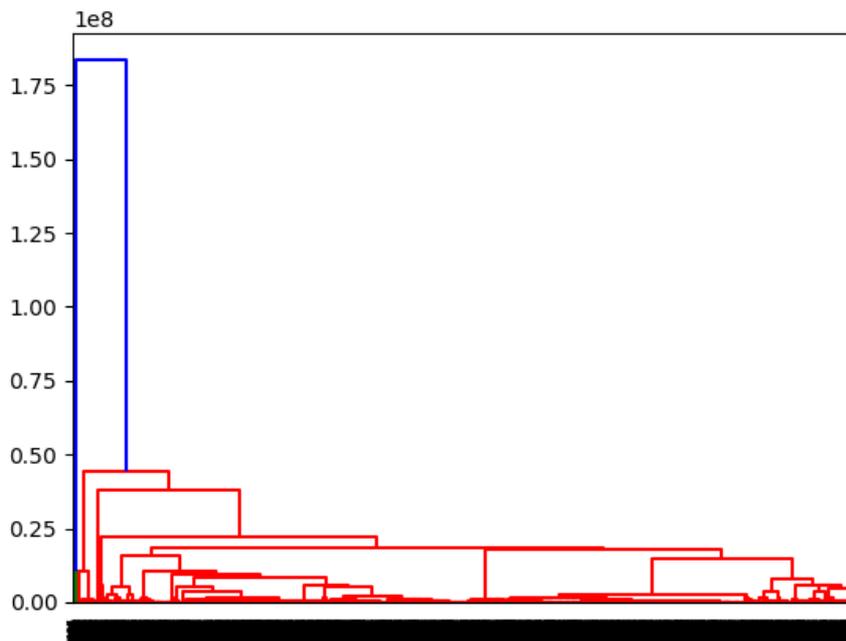


Figure 12

Dendrogram based on world map dataset.

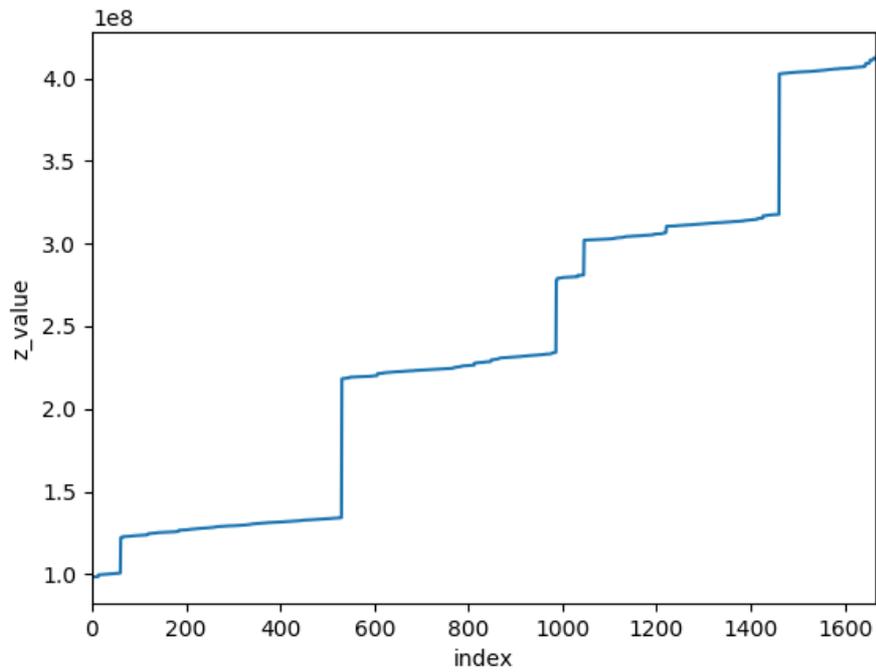


Figure 13

Z-value distribution over the indexes of a world map dataset reduced to only the region represented by single cluster.

	Database_index	x	y	z_value	cluster
519	1385853.00000	15631.00000	8003.00000	133898335.00000	0.00000
520	1387199.00000	15634.00000	8100.00000	133925156.00000	0.00000
521	1388176.00000	15818.00000	8089.00000	133944006.00000	0.00000
522	1388426.00000	15779.00000	8142.00000	133948589.00000	0.00000
523	1388590.00000	15822.00000	8150.00000	133952124.00000	0.00000
524	1388933.00000	15917.00000	7724.00000	133958897.00000	0.00000
525	1389281.00000	15918.00000	7770.00000	133965532.00000	0.00000
526	1393405.00000	16314.00000	7785.00000	134049222.00000	0.00000
527	1393919.00000	16252.00000	7811.00000	134059354.00000	0.00000
528	1395236.00000	16334.00000	7916.00000	134084852.00000	0.00000
529	1397405.00000	15997.00000	8123.00000	134127579.00000	0.00000
530	1400522.00000	16128.00000	8118.00000	134187560.00000	0.00000
531	3367226.00000	12741.00000	8194.00000	218189849.00000	0.00000
532	3367240.00000	12760.00000	8193.00000	218190146.00000	0.00000
533	3367741.00000	12772.00000	8327.00000	218223674.00000	0.00000
534	3368290.00000	12805.00000	8280.00000	218374801.00000	0.00000
535	3369260.00000	12980.00000	8309.00000	218394418.00000	0.00000
536	3369986.00000	12861.00000	8413.00000	218408947.00000	0.00000
537	3371073.00000	12995.00000	8439.00000	218429999.00000	0.00000
538	3373072.00000	13177.00000	8335.00000	218469867.00000	0.00000

Figure 14

Screenshot of a dataset world map cluster data. Actual index (519 – 538) is the index of a record in the cluster. Database_index is the index of a record in the dataset. X, y, z_value and cluster columns represent respective dimensions and cluster of a record in a dataset.

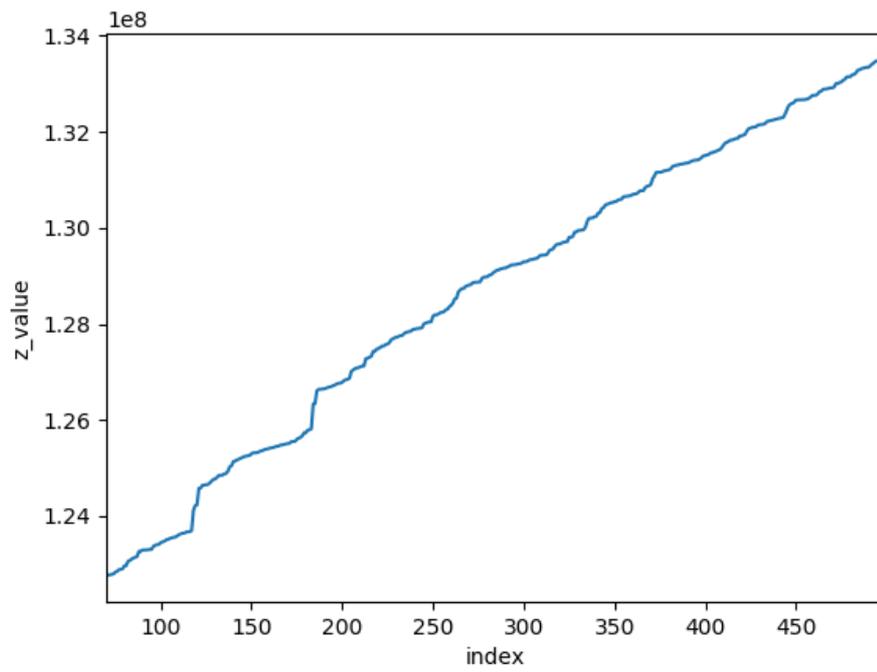


Figure 15

Z-value distribution over the indexes of a world map dataset reduced to only the region represented by single cluster identified using formula from section 2.2.