Finding Outliers In Big Data

Kunyu Ye
New York University
Brooklyn, NY
ky1081@nyu.edu

Tianhan Yuan
New York University
Brooklyn, NY
ty936@nyu.edu

Xiaoyan Zhang
New York University
Brooklyn, NY
xz1133@nyu.edu

ABSTRACT

This paper explores approaches for finding null values and outliers in big dataset. Algorithms involved include block nested-loop method and index based method.

KEYWORDS

Bigdata, Outlier, K-means Clustering, Index-Based Algorithm

ACM Reference format:
Kunyu Ye, Tianhan Yuan, and Xiaoyan Zhang. 2018. Finding Outliers In Big Data. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 3 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

The general purpose of this project is to analyze a big data set, with great volume and complexity, through certain techniques and technologies that are specifically developed for "big data" analytic in order to unearth certain implications within this data set, and to process the data set into desired result for data analysis. In this project, the goal is to discover and identify NULL values and outliers within a data set. Entries that contains NULL values are those that contain empty fields. Outliers are entries that are considerably dissimilar or in inconsistent with the remainder of the data, or the existence of which is impossible or defies common sense.

Outliers and NULL values could negatively interfere with statistical analyses; the data might falsely reflect on the target situations or behaviors and, therefore, lead to inaccurate analyses. So dealing with NULL and outliers is a crucial step performed prior to data analyses. NULL values and outliers are to be dealt separately with different approaches. As NULLs are just empty values, their mere existence is disruptive. So they are simply to be eliminated upon discovery. On the other hand, although outliers ought to be removed from the original data set, they should be collected and preserved. In some occasions, they might be able to provide valuable information if further investigations are conducted regarding these outliers. This will be further discussed later in this report.

2 PROBLEM EXPLANATION AND FORMULATION

In this project, the main problem is to analyze the null values and outlier in big dataset. Null value can be displayed by column or row to have an overall view of specific column or row. And the main idea of finding outlier is choose a proper variable k and then calculate the Kth nearest neighbour of each record. Eventually, it is easy to sort out several records with biggest Kth-Nearest-Neighbour Value as the outliers. Certainly, these outliers will be further checked and explained.

3 METHODS, ARCHITECTURE AND DESIGN

3.1 Discovering Null Values

To find the null value in dataset, the idea is simply traverse the whole file, split the given tsv format file into lines and then split each line properly into column fields. The number of columns in each row will be firstly compared with the standard number, no matter the actual number of the current row is more or fewer than the standard number, it will be judged as abnormal column. On the other hand, if the number of columns in current row matches the standard, then each column field will be checked one by one to see if its content is equal to null or empty or just blank.

Two sets of MapReduce are applied here to better show the test result and verify each other. The first set of MapReduce aims at showing the details of each row where the number of the column containing null values will be printed out in a list. In this set, the intermediate result between mapper and reducer is in format "RowNumber [tab] ColumnNumber" where [tab] is the delimiter for key and value in mapreduce. In the reducing part, the records with same RowNumber(key) will be gathered and processed to be a list in same reducer. So, the final result is in format "RowNumber has x null values: a, b, ..."

Being slightly different, the mapper of the second set of MapReduce produces the intermediate result in format "ColumnNumber 1" and in reducing part, the number of records with the same ColumnNumber(key) will be counted as the final result. The final result is in format "ColumnNumber has x null values".

3.2 Discovering Outlier with Block Nested-Loop Method

To find the outlier in big data, in theory, what to do is to traverse each point and for each point, the distance between this point and every other point will be calculated and stroed in array in order. But considering storing N-1 pairs data with each record of N records would be extremely space-consuming and in that case, most of the pairs are not really needed later, it is reasonable to only store top M pairs with each records, making sure M is always larger than the number which will be chosen as threshold later. Then, it is time to adjust the threshold value to pick only one pair from M pairs of every record and then sort out the records with biggest N pairs of value.

3.3 Discovering Outlier with Index Based Method

The reason nested-loop map-reduce program is slow is that it needs too many calculations. The time complexity of this algorithm is O(logN*N). In order to improve the calculation speed, we need to try other algorithms.

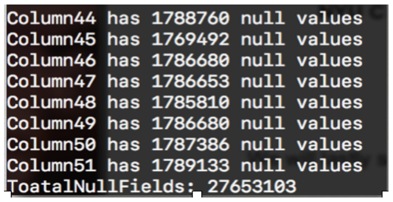
The index based algorithm is a very fast algorithm to find nearest neighbors. The key point of this algorithm is a spatial index tree. A spatial index tree is a space-partitioning data structure like R tree or K-D tree. We don't need to implement the data structure by ourselves. There is a machine learning package in python called *scikit-learn.* It can help us to build a k-d tree. But the problem is that there isn't any build-in package in spark. Without build-in package, we find it is hard to use k-d tree for rdd data type. With a lot of research, we finally figure out a solution.

We can divide the implement procedure into four steps. The first step is data pre-processing. In this step, we need to read data, using map and filter function to clean the data and get a rdd. The second step is to build a K-D tree. But ahead of this, we need to collect the data from rdd first because we can not build K-D tree using rdd. So we use collect function in spark to collect the data and then using sklearn package to build the K-D tree. This step is not parallelizable. The third step is to broadcast the K-D tree object. The purpose of this step is to make the fourth step parallelizable. The fourth step is to use K-D tree built before to calculate the nearest neighbors. In this step, the rdd data type can access the K-D tree object because we broadcast it in the third step. So this step is parallelizable.

3.4 A further improvement based on Index Based Method

The index based algorithm is a fast method, it can handle millions of rows of data set. But for more than 10 millions rows of data set, it needs too much time. This is why we make another improvement based on this algorithm.

What we do is that before building the K-D tree, we do some data selection first. The purpose of this step is to select some candidate outliers. In this step, if we can select some candidate outliers, then we only need to calculate the nearest neighbor only for the candidate outliers. For example, if we have a 10 millions rows data set and after selection we get 1 million rows of data, then we can use index-based algorithm introduced in the previous section to calculate. We think clustering method is a good fit here to do this job. If we can divide the data into several clusters, then we think only the data points that are far away from their cluster center can be possible outliers. We only collect all these candidate data points and then use index-based algorithm to calculate the outliers. Here we use build-in k-means method to finish this job.



```
Column44 has 1788760 null values
Column45 has 1769492 null values
Column46 has 1786680 null values
Column47 has 1786653 null values
Column48 has 1785810 null values
Column49 has 1786680 null values
Column50 has 1787386 null values
Column51 has 1789133 null values
ToatalNullFields: 27653103
```

Figure 1: Display Null Values by Column

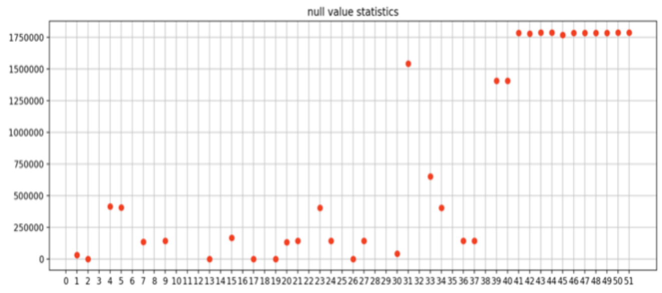


Figure 2: Null Values Distribution by Column

4 RESULT AND ANALYSIS

4.1 Discovering Null Values

Since it is found that several datasets only few rows, it is reasonable to just skip that kind of dataset. Additionally, some datasets cannot be split properly because the delimiter used in the dataset are mixed, which contain both tab and blank. For those dataset, it is also difficult to do the test on. Finally, all the other datasets remain in good condition to be tested. Dataset with million rows or more will be paid more attention to because those kind of dataset makes more sense in big data context.

The result are in two format: one is displayed by row and the other is displayed by column(See Figure 1). For more intuitive result, the result is used again to plot a null value distribution figure(See Figure 2). And both of the two MapReduce will provide the summary for the total number of the null values in the dataset, which can be compared to verify the correctness of the test.

4.2 Discovering Outlier with Block Nested-Loop Method

There are kinds of test in this category: One is to apply this approach on some relatively small datasets to get the proper result. The other one is try to apply this approach to part of some relatively big dataset to see how much time it will take. The reason for this is that the complexity of this block nested-loop algorithm is O(N*N) who requires too heavy computation when processing big data. So in our experiment, this approach can finish the test on dataset contains 5000 rows in approximate 15 minutes but when the amount of data goes higher, the test could take much more time. That's also the reason why the Index Based Method is developed.

4.2.1 The Fair Student Funding Dataset. This is a rather smaller dataset which contains about 1570 entries. The following three columns are selected for this analysis: "Weighted Registers"(unit

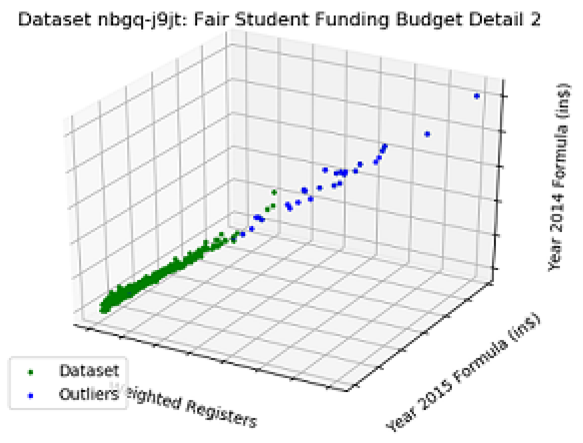
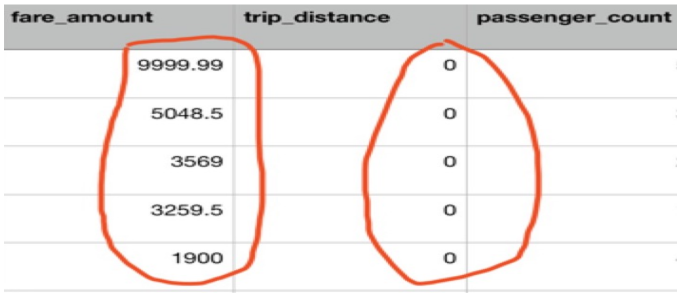


Figure 3: Visualization of Outliers in Fair Student Funding Dataset by Three Columns

used for funding calculation), "Fiscal Year 2015 Funding", and "Fiscal Year 2014 Funding". The *Block Nested-Loop Algorithm*, run on MapReduce, is used for processing this dataset. One single mapper and ten reducers are configured, and the total time cost is about 10 minutes. The top 30 results are selected as outliers. As a result of running this algorithm, most of the top 30 results are outliers. Some of the top 30 results are not outliers due to the fact that *Block Nested-Loop Algorithm* is based on distance calculation. The top results are only the ones selected from the sorted result list, and some of their distances are not as big as the outliers. Despite there are yet any valuable information mined from these outliers, a visualization(See Figure 3) of the result is created to show the significance of this analysis.

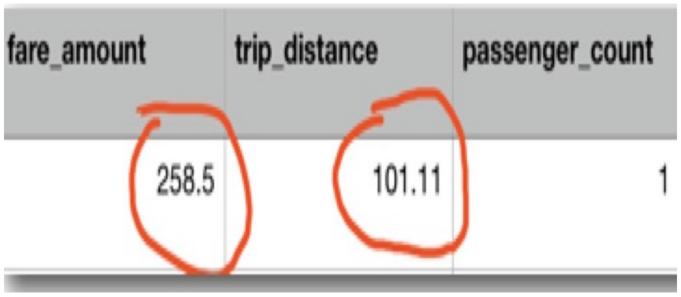
4.3 Discovering Outlier with Index Based Method

To test the efficiency of this algorithm, we use a taxi related data set which contains more than 1.2 million rows. We can get results in 10 minutes in Spark. For comparison, the nested-loop method will run 12 minutes for 10 thousands rows. So we think it is a big improvement. Also we found the top outliers are meaningful. Here we choose 2 columns from this taxi data to analyze. The first attribute is the amount of fare. The second attribute is trip distance. The trip distance and fare amount are positive correlation. The longer trip distance will have a higher fare amount. The Figure 4 is top 5 outliers. It is impossible to charge passengers thousands of dollors for 0 trip distance. The Figure 5 is one row of top 20 outliers. I think the reason why this row is selected as an outlier is that it is not common for a passenger to take a more than 100 miles taxi trip.



fare_amount	trip_distance	passenger_count
9999.99	0	
5048.5	0	
3569	0	
3259.5	0	
1900	0	

Figure 4: Top 5 Outliers



fare_amount	trip_distance	passenger_count
258.5	101.11	1

Figure 5: One Row of Top 20 Outliers

4.4 The Result of The Further Improvement

We compared the result of the improved algorithm and the index based algorithm. For the same taxi data set, we find that the result of top 10 outliers is completely same. For the top 20 outliers, there are only 1 to 2 different outliers. So we think the further improvement algorithm is feasible.

5 REALATED WORKS AND REFERENCES

S. Ramaswamy, R. Rastog, and K. Shim. 2000. Efficient Algorithms for Mining Outliers from Large Data Sets. ACM New York, NY, USA