Null and Outlier Detection in Large Datasets

DSGA 1004 Project Final Report

Zhe Huang
New York University
zh1087@nyu.edu

Daoyang Shan
New York University
ds5471@nyu.edu

Yueqiu Sun
New York University
ys3202@nyu.edu

1 INTRODUCTION

In data analytics, it's a common practice to identify data points that we consider as interference to our analysis, such as instances containing null value or obvious outliers. Such instances are likely caused by erroneous measurement and thus should be removed from the data set to avoid statistical problems in the following analysis. Finding null points could be a trivial task, but we do need to think about some challenges for outliers. For example, how do we define an outlier without knowing the underlying distribution of data points? How to find an efficient algorithm that can be applied to huge data set? In our project we mainly focus on efficient unsupervised algorithms to investigate outliers with the help of big-data tools.

2 PREVIOUS WORKS

Extensive works on the exploration of outliers have been done by researchers. Chandola, Banerjee and Kumar discussed multiple outlier detection algorithms and their applications in their paper [1]. Despite the difference between different algorithm, we can divide those algorithms into two categories: algorithm for numerical data and algorithm for categorical data, due to the innate difference between these two types of data, especially the 'definition of outliers'. In our work, we'll also divide our problem into two sections and work on both of them with distinct strategies.

2.1 Numerical Data

In a data set with majority of attributes being numerical, outliers are generally defined as data points 'far away' from majority, and distance metrics like Euclidean distance are normally used here. One simple yet effective algorithm used here is k-means clustering. Although not designed for outlier detection, clustering can be adjusted for such task [4]. A common procedure includes generating clusters, selecting a distance threshold, and identify all points that exceed such threshold from their corresponding centroids as outliers. Clustering algorithm doesn't assume any statistical assumption on data set, and can be (in most of the cases) easily explained and visualized. However, it has multiple drawbacks, including low efficiency (clustering is generally NP Hard) and cluster instability caused by outliers [2].

2.2 Categorical Data

Distance based definition of outliers fails for categorical data because it makes no sense to calculate 'distance' two categories. Instead, frequency based definition is used, by which outliers are defined as points that represent a rare combination of categories. AVF (Attribute Value Frequency) algorithm, which mainly calculate a score that represents the 'rarity' for each data point and identify those points with high rarity as outliers, is one of the algorithms designed for categorical data [3]. AVF score for data point i is defined as:

$$\sum_{j=1}^{m} \text{frequency of } j\text{th attribute value of } x_i \text{ in column } j$$

We see that lower AVF score implies higher rarity. Again, AVF assumes no statistical distribution on data set, and are easy to be implemented.

2.3 Hybrid Data

The amount of works on data sets with both categorical and numerical data is surprisingly small. A proposed algorithm by some authors first splits data set into chunks based on its distribution of categorical data combinations, and applies algorithms for numerical data for each chunk. However, such algorithm obviously suffers from the potential huge amount of possible combination and the difficulty of comparison between combinations. In our project we don't explore further into this algorithm.

3 PROBLEM FORMULATION

In our project we have 50 data sets from which we need to identify null points and outliers (if possible) and all of them are extracted from totally different realms, which makes us hard to manually inspect each of them and design individual strategies. Therefore we propose to process all of them with a highly generalized automated procedure with almost no examination and extraction of domain knowledge. In another word, we aim to identify null points and outlier from any given data set without even looking into it for domain knowledge. Such action will indeed harm the accuracy for a specific data set, but guarantee the overall efficiency and performance.

4 METHODOLOGY

Our project implementation can be summarized as following: first we pass all data set to a cleaner script, remove those elements that we can't handle (usually because we don't have the required domain knowledge for that) and identify if this data set is generally numerical, generally categorical or hybrid. We pass all data sets that we believe to be generally numerical to clustering algorithm(algorithm for numerical data) and pass the rest (after some preprocessing) to AVF algorithm(algorithm for categorical data).

4.1 Data Cleaning

The first step is to automate the process of data cleaning. First we identify those columns with a large percentage of null or empty values (in the final version, 50%) and drop those columns. Indeed in real world those columns may contain important information and even they're strong indicators of outlier, however in the highly generalized scenario those columns only introduce uncertainty, making the following steps hard to process. For example, if we keep a column with a large portion of blanks without knowing anything about the domain knowledge, should we identify those blanks as null points or set some default value for them in the outlier detection algorithm?

Second, we classify all columns into three types: numerical, categorical and other. Numerical column indicates that distance metric is meaningful on this column, categorical column contains small enough number of unique values (in our implementation, number of unique values is less than 10% of the number of total values), while other means this column can't be classified as either numerical or categorical. Typical 'other column' includes address, email or timestamp. Since we can't assume any domain knowledge, we have to drop all 'other columns'. For the rest of two types, if more than 75% are numerical, we drop all categorical columns and apply clustering on it in the following step. Otherwise, we apply binning on all numerical columns to make our data set ready for AVF algorithm. We use QuantileDiscretizer from ml.feature package in PySpark for binning with default number of bins as 10. Indeed we ignore this data set if no column is left.

4.2 Clustering

We use PySpark to implement k-Means Clustering algorithm. First we identify all these rows with null or empty value within as null points. After we filter all those points (rows), we specify the range of k on which we search for the optimal k (in our implementation, 2 to 10). For each candidate k, we apply clustering algorithm on it and get the corresponding WSSSE score (Within Set Sum of Squared Error) and find the optimal k, which brings the largest second derivative of WSSSE curve at that k. Note that like other applications of clustering, we must normalize all columns to avoid the effect of different scales.

After we fix the k we want to use, we assign each point to a cluster and calculate its distance from the centroid. With a specified outlier threshold, we find those points with largest distance from centroid as outliers. We also use PCA based visualization method to validate the correctness, which we'll discuss in section 5.

4.3 AVF

Our first implementation of AVF is completed with Spark, which directly uses the pseudocode from Koufakou's paper as shown in Figure 1. Similar to clustering algorithm we identify those rows with blank or null values as null points and remove them from out data set. Then we calculated AVF score for each remaining points (rows) as described in section 2, sort them by AVF score and select those with least AVF scores as outliers.

```
Input:   Dataset – D (n points, m attributes)
         Target number of outliers – k
Output:  k detected outliers

Label all data points as non-outliers;
Calculate frequency of each attribute value, $f(x_{il})$;
foreach point $\mathbf{x}_i$ ( $i = 1..n$ )
    foreach attribute $l$ ( $l = 1..m$ )
        AVF Score ($\mathbf{x}_i$) += $f(x_{il})$ ;
    end
    $Average_l$ ( AVF Score ($\mathbf{x}_i$) );
end
Return top k outliers with minimum AVF Score
```

Figure 1: AVF Algorithm Pseudocode

```
Input:   Dataset – D (n points, m attributes)
         Target number of outliers – k
Output:  k detected outliers

HashTable H;
map($k1 = i$, $v1 = D_i = \mathbf{x}_i$, $i = 1..n$) begin
    foreach $l$ in $\mathbf{x}_i$ ($l = 1..m$)
        collect($x_{il}$, 1);
end
reduce($k2 = x_{il}$, v2) begin
    $H(x_{il}) += \sum$ v2;
end
map($k1 = i$, $v1 = D_i = \mathbf{x}_i$) begin
    AVF = $\sum_{l=1}^{m} H(x_{il})$;
    collect(k1, AVF);
end
reduce($k2 = AVF_i$, $v1 = i$ );
```

Figure 2: MR-AVF

4.4 MR-AVF

One serious concern of our Spark implementation of AVF is that we seem to only 'accelerate a sequential algorithm with the help of parallelized tool', instead of improving the implementation from algorithmic and design level. MR-AVF, the MapReduce version of AVF, is covered by some previous researchers [3]. As Figure 2 shows, MR-AVF consists of two MapReduce phases. However, the direct implementation from pseudocode is actually hard, since passing data structures like HashMap between mapper and reducer is not an easy process. Instead, we develop our own implementation that's slightly different from original pseudocode, which we'll illustrate below.

Let's use a simple data set for better illustration (see Table 1). Note that index is not part of the initial data set. We use unique index to mark individual data points. The whole process consists of three sequential MapReduce instead of two as shown in the pseudocode. We use || to separate key and value.

- Mapper 1: For each line we read in, take the value in each entry and the column number they reside in as key. We take

index	col1	col2
1	dog	A
2	dog	B
3	cat	B
4	dog	A

Table 1: Sample Data Set

column number because a value may appear in different columns. We set the index number of this row as value.

Mapper 1 output: (dog, 1) || 1, (A, 2) || 1, (dog, 1) || 2, (B, 2) || 2, (cat, 1) || 3......

- Reducer 1: For each key-value pair with the same key, maintain key, and change value to an array of all indices that have this value at this column, as well as the amount of such indices.

  Reducer 1 output: (dog, 1) || [3, (1, 2, 4)], (cat, 1) || [1, (3)], (A, 2) || [2, (1, 4)], (B, 2) || [2, (2, 3)]
- Mapper 2: For each key-value pair, generate new key-value pairs such that all indices in the value serve as key, while column number and the count in value as new value. Note that here we use count to replace actual categorical values (like 3 replaces dog).

  Mapper 2 output: 1 || (1, 3), 2 || [1, 3], 4 || [1, 3], 3 || [1, 1]......
- Reducer 2: Recall that index is unique, so for each index (as key), we subtract all counts from values, store them into a new array, and use that array as new value. We abandon column number here.

  Reducer 2 output: 1 || [3, 2], 2 || [3, 2], 3 || [1, 2], 4 || [3, 2]
- Mapper 3 and Reducer 3 are trivial, in which we simply sum all counts in value up as the AVF score for this index (point), and sort them in Reducer by switching key and value. In actual code we use Spark to execute this process for convenience concern.

We expect this implementation to be efficient and available for scaling up because of two main reasons: 1, it takes only $O(1)$ memory, which avoids potential risks like OOM Exception in Spark; 2, we can utilize an arbitrary number of workers in this task, which is important for scalability.

5 RESULT AND DISCUSSION

5.1 Cleaner

Our cleaner determines that 37 out of 50 data sets are suitable for the following outlier detection. Data sets that are dropped by cleaner include: data set that contains almost only timestamps; data set that contains only 'messy data', for example address, email etc; data set that contains too many blank entries in every single column. Indeed we should be able to clean those data sets either manually or given more domain knowledge, but those methods are unavailable/undesirable in our project. Among the 37 data sets that we pass to outlier detection, 8 are identified as mainly numerical and passed to clustering, while the rest, either mainly categorical or hybrid, are passed to AVF.

5.2 Clustering and Visualization

Our clustering algorithm processes those 8 numerical datasets. All rows containing null or blank are identified as null, and we select the top 5% points sorted by the distance to centroid in descending order. Here we use ny8v-zzzb.tsv as sample for illustration. Optimal k for this data set is 3, no points are identified as null points in this data set, and 21 points with highest distance to their corresponding centroids are classified as outliers.

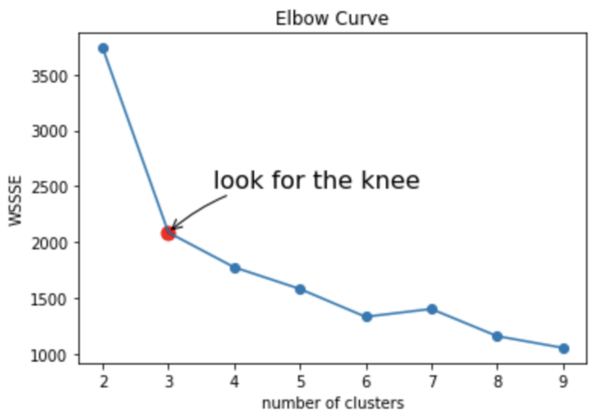

Figure 3: WSSSE curve for different k

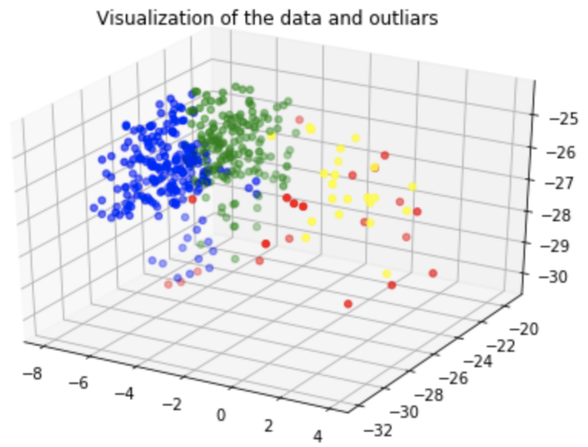

Figure 4: Visualization of clustering

In the visualization above, we clearly see the 'sudden drop of WSSSE' at k=3. If we visualize the clustering results with PCA, we see that outliers (marked with red) seem to be reasonable in this 'triple cluster' data set (three clusters are marked with blue, green and yellow).

5.3 AVF and MR-AVF

Our AVF algorithm processes the other 29 categorical and hybrid data sets. Again, we identified rows with null or blank as null, and select 5% with lowest AVF score as outliers.

Compared to the visualization of AVF, the more interesting topic is the comparison of Spark AVF and MR-AVF. We test on three data sets with different sizes. Both Spark AVF and MR-AVF produce exactly the same result. Then, we record the wall time of both methods because Hadoop and Spark have different ways of time recording. The result is shown in Table 2:

Table 2: Time Comparison Between Spark AVF and MR-AVF

Name	Size	MR-AVF	Spark AVF
pvqr-7yc4	6.4*1e8	356	1400
tm6d-hbzd	2.6*1e8	123	200
r4s5-tb2g	360725	55	20

According to this result, MR-AVF seems to have better scalability on large data sets, while Spark AVF does better on small data sets. Indeed, this result may influenced by different I/O mechanisms, different system processing costs or even our method of implementation. Also, Spark AVF surpasses MR-AVF a lot on convenience. For example, MR-AVF need extra steps for null point detection, while the same task can be easily integrated in Spark AVF.

5.4 Discussion and Potential Improvements

In practice, we expect the more frequent use of AVF algorithm compared to clustering because we should assume that most data sets in real world are not purely, or dominantly, numerical, and AVF algorithm provides a relatively reasonable outlier detection mechanism for hybrid data sets. Also, although we can't fully assert the inefficiency of Spark AVF on large data sets due to the potential inaccuracy in time comparison, MR-AVF does seem to scale up better for large data sets. However, Spark AVF outperforms MR-AVF on small data sets, and the convenience of Spark compared to Hadoop should also be taken into consideration. Spark AVF is still a good choice given the volume of data is not too enormous.

Obviously, there are still plenty of spaces for potential improvements. First, we can develop more complicated cleaner that can extract more information and therefore identify less column as 'intractable' (and drop them). For example, an ideal cleaner should be able to parse time stamps (in different types), number-like entries (percent or scientific notation), list-like objects (like "[apple, pear, orange]") or others. Even if we can't guarantee the extraction of all information, an 'almost omnipotent' cleaner can definitely enhance the accuracy of outlier detection and avoid the loss of crucial information. Second, we may apply more 'smarter' mechanisms for setting outlier thresholds. Currently user should manually input the threshold (5% points are outliers) but in real world it's very likely that user has no idea what threshold he/she should input. Finally, we should explore more efficient outlier detection algorithms, especially for hybrid data, since the algorithms we're applying currently may not be optimal. For example, clustering, although useful, is not designed for outlier detection at first, and the time complexity is generally NP-Hard. To conclude, our project sets a good starting point for automated and generalized outlier detection, yet we accept the drawbacks in our design and look forward to the future improvement.

6 RECAP AND MISCELLANY

In our project, we design and implement an automated data cleaning and outlier detection process. Our cleaner cleans data without assuming domain knowledge and identifies data that suitable for automatic outlier detection. Mainly numerical data sets are sent to k-means clustering while the rest are sent to AVF. Our clustering algorithm identifies null points and finds the optimal k, which is then used for outlier detection. We implement two versions of AVF, one in Spark and anther one, with our original way of implementation, in MapReduce. Both implementation bring the same result and we also compare the performance with respect to time for both versions.

See code, command line instruction and other notes at https://github.com/szyyn95/DSGA1004

REFERENCES

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41 (7 2009). http://doi.acm.org/10.1145/1541880.1541882

[2] Sanjay Chawla and Aristides Gionis. 2013. *<italic>k</italic>-meansâĂŞ: A unified approach to clustering and outlier detection.* SIAMPub, 600 Market Street, 6th Floor, Philadelphia, PA, 189–197. https://doi.org/10.1137/1.9781611972832.21 arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611972832.21

[3] A. Koufakou, J. Secretan, J. Reeder, K. Cardona, and M. Georgiopoulos. 2008. Fast parallel outlier detection for categorical datasets using MapReduce. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence).* IEEE Computer Society, Washington, DC, USA, 3298–3304. https://doi.org/10.1109/IJCNN.2008.4634266

[4] R Smith, A Bivens, M Embrechts, C Palagiri, and Boleslaw Szymanski. 2002. Clustering approaches for anomaly based intrusion detection. *Proceedings of Intelligent Engineering Systems Through Artificial Neural Networks* 12 (01 2002), 579–584.