Discovering Nulls and Outliers

Project Final Report (DS-GA 1004)

Diogo Mesquita
New York University
dam740@nyu.edu

Mihir Rana
New York University
ranamihir@nyu.edu

Kenil Tanna
New York University
kenil@nyu.edu

ACM Reference Format:
Diogo Mesquita, Mihir Rana, and Kenil Tanna. 2018. Discovering Nulls and Outliers: Project Final Report (DS-GA 1004). New York, NY, USA. https://github.com/ranamihir/big_data_project

1 INTRODUCTION

Missing value and outlier detection are two of the most essential, time-intensive, and subjective tasks in any form of data preprocessing. In most unstructured data sets, missing values tend to not have strictly formatted representations, and instead appear as potentially valid data values, which are known as "disguised missing data" [6][7]. This inconsistency arises due to several reasons – non-standardized ways of data collection and storage across various sources, changes in the way an organization stores null values, human error, etc. In the past, several methods have emerged to process missing values; in many cases, especially in a machine learning framework, it is typical to remove rows containing them if the total number of such rows is small, to impute them using several basic (such as mean, median, mode imputation) or other sophisticated techniques, or build entire predictive algorithms to impute them before the main task at hand is tackled [2], if their total count is relatively large.

For outlier detection, typically, the data points are modeled using a statistical distribution [9][10], and the data points that do show anomalous behavior [3] or the distribution of the postulated model by a "substantial" margin are considered outliers. One of the main drawbacks with this approach is that we might not have enough prior information to hypothesize an underlying data distribution, especially in extreme cases when the prior chosen is strongly biased.

Before null values or outliers can be explored, an essential first step is to clean and standardize the data sets by bringing them to a common format across columns and data sets. The first step would be to clean and standardize the data sets by bringing them to a common format across columns and data sets. This is especially becoming of paramount importance in today's day and age of an incessant barrage of data, which is often unstructured, raw, or too disorderly, and requires constant cleaning and processing. Unless it is brought to standardized format, identifying and analyzing what is important, discarding what is not, and strategizing by piecing all this together to make informed decisions, is bound to become a daunting challenge.

In this study, we consider a data set collection of 50 data sets from NYC OpenData. Since the overall objective of this study is only discovery, we will, as such, directly report and remove nulls and outliers.

Big Data (Spring 2018), New York University, New York

https://github.com/ranamihir/big_data_project

2 PROBLEM FORMULATION

As mentioned earlier, we will perform null and outlier detection on 50 sets obtained from NYC OpenData. The problem is broken down into three main components:

2.1 Data Cleaning

In the scope of this study, data cleaning or preprocessing pertains to parsing the data to bring to the correct format. An example of this type, specific type of data being recorded, is the case where income is recorded as "$0.00", leading it to be considered as a string object. Similarly, numeric quantities may also be recorded with commas, like "120,000", which may lead to erroneous analyses, and symbols such as these should be removed at the very first stage. One should also be mindful of zip-codes, which would be typically of numeric type by default, but should be treated as strings, specifically, conforming to only certain valid zip codes.

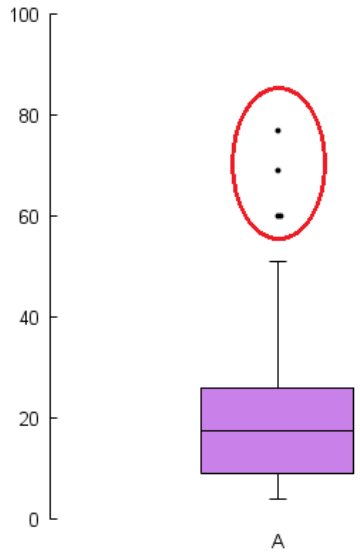
2.2 Missing Values

Missing values, or nulls, present in varying forms for each data set, and in many cases, also do not conform to strict format. For example, if a column is of numeric type, it may encoded as a string within quotes in many cases, such as "100.00". This occurrence becomes more common when null values pertaining to certain rows are encoded as strings, like "None", "N/A", etc., leading to the entire column being parsed as a string object. In many cases, nulls also present as values that are invalid, such as an income being negative, a point being drastically out of the range of the valid values, a probability or a percentage being negative or greater than 1, etc. Values such as "999", "-999", etc. are also typical. These will be reported and then removed for further analysis.

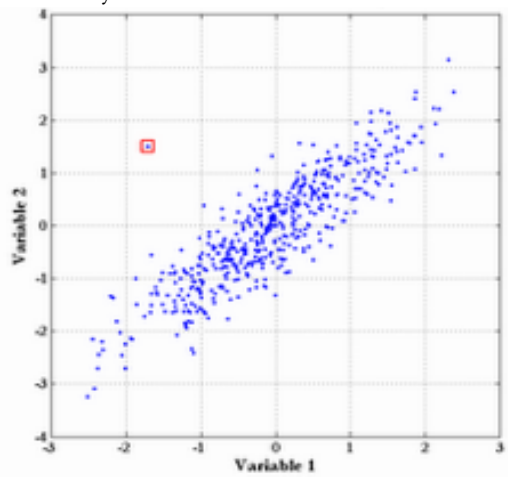
2.3 Outliers

The definition of outliers is subjective to the problem at hand, and has taken many forms over the years. One way is characterize them was described above, of postulating a distribution for the data points and finding points that stick out. Another approach is to use distance-based methods (such k-Means, Quartiles, etc.) and find points that have distances to centers larger than a threshold or clusters that have very few points in them [9].

Univariate outliers, like the ones shown below, are relatively easy to detect:



However, multivariate ones are much harder to deal with. In multiple dimensions, it's harder to visualize or reason out why a data point may be an outlier. For a 2-dimensional distribution, an outlier may look as follows:



In this study, we'll report all such outliers – including ones that may not conform to a distribution spanned across over 100 dimensions.

3 RELATED WORKS AND REFERENCES

[3] describe many parametric and non-parametric methods for the purpose of outlier detection. For parametric methods the common methods are IQR detection, Tukey's method of boxplots, z-score and the standard deviation are some of the common methods used for identifying outliers, since they assume that the data distribution follows a specific target distribution. Although it is easy to calculate errors by this method, one major flaw with this is the assumption that real-world data follows a particular distribution while it may not be the case.

One of the most common non-parametric methods is the nearest neighbor method, which is giving a score to each sample based on its neighbor which is useful in finding global outliers. The notion of global outliers is that they are far away from the rest of the data and the local outliers are generally the ones located in areas of relatively low density. LOF is one such method that finds local outliers as well. It was shown by [11] that the global and local outliers are not strictly different by there are varying degrees of locality.

Other related works have generally focused on efficient implementation of the techniques discussed above, as in rather than improving the outlier detection method they make sure that the current methods are efficient.

Other methods such as ensemble techniques have a potential to improve outlier detection since, each technique captures different things/distributions about the data. Feature bagging is a common procedure which is used for clustering based ensemble methods. The subsampling based method [12] shows that ensemble methods can be applied in an efficient and effective way while without subsampling the data, ensemble methods might be costly in terms of the computation since every method has to be run separately.

Another interesting method was proposed by [8] where they show rather than using a single-view of the data, they use multiple views and although the problem becomes more complex due to the inconsistent behaviour of the samples across multiple views. They propose a multi view low rank analysis framework which contains robust data representation.

4 METHODS, ARCHITECTURE, AND DESIGN

Our code can be found here. For reasons described below, we have created indexed data sets which is made available under:

`/scratch/dam740/GROUP7_rid_gz/`

The main focus in our entire methodology is on making all functions generalizable across columns, data sets, and data set collections. For instance, in outlier detection, we use a variety of similar outlier detection algorithms and finally report the rows which have been identified as outliers by more than 50% of the algorithms.

An overview of the methodology has been shown in Figure 1; the specific ways of doing so are described in the following sections.

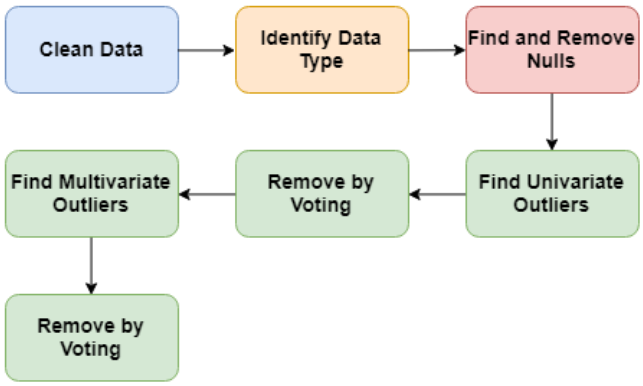


Figure 1: Flowchart of Methodology

4.1 Data Cleaning and Null Value Removal

We have implemented several wrapper functions that automatically identify the true data types for a column. For this, we randomly sample 500 rows per column, apply regular expressions that remove commas (in the middle of a string) and all (non-alphanumeric) symbols like "$", "@", etc. from all them, and finally check if at least 70% of the points are compatible with a numeric type. Also, to make data handling easier by reducing the overhead amount of data stored and communicated at each step, we added a new column of indices (starting with 0) for all data sets. Please note, this is a necessity for running our code, which is why we have made the new indexed data available (see above for path). Additionally, if a new data set is used which does not have the indices (under the column name `rid`), our code will automatically create them and then perform the analysis.

As for null values, we wrote several regular expressions that handle the many forms in which null values usually present themselves, like "None", "N/A", "Unknown", "-999", " ", "-", etc. Finally, before performing outlier detection, we drop all null values in the data sets.

4.2 Outliers

For outlier detection, we follow the basic methodology presented in [3]. For robustness, we first apply all univariate outlier detection methods, and only report those rows as outliers which have been identified by more than 50% of the methods, and then do the same for multivariate techniques. To gain a (small) computation speedup, we also drop the univariate outliers before proceeding with the multivariate ones.

The methods have been divided into subgroups based on the type of algorithms they employ, and the type of column data type – numeric and strings.

It is worth noting that we used [3] as a primary reference for this study, and for robustness, we implemented at least 1 technique from each category of outlier detection methods out of the ones which are applicable (i.e., unsupervised) – with our own modifications and improvements – and report only those rows as outliers which have been identified by more than 50% of the *similar* algorithms.

- **Statistical Anomaly based**
  - *Box Plot Rule.* For numeric columns, we use the approach used in box-plots or whisker-plots, where an outlier is is found as follows:
    Let Q_1 = First Quartile, Q_2 = Second Quartile, Q_3 = Third Quartile.
    And, inter-quartile range, $IQR = Q_3 - Q_1$
    Then, a given value V is an outlier if:
    (1) $V < Q_1 - 3 \times IQR$, or
    (2) $V > Q_3 + 3 \times IQR$
  - *Gaussian model based (z-score).* One standard approach in identifying outliers is by fitting a Gaussian to the data, and identifying rows having:

    $$z = \frac{x - \mu}{\sigma} > 3 \text{ or } z = \frac{x - \mu}{\sigma} < -3$$

    where z = z-score, x = row value, μ = mean of Gaussian distribution, and σ = standard deviation.
  - *Other Probabilistic Models (Beta, Gamma, etc.).* In standard statistical anomaly detection techniques, it not uncommon to assume that the data is normally distributed; however, in many cases, it is possible that the data actually follows a different distribution. To account for this, we also fit other common probabilistic distributions like the *Beta* distribution and the *Gamma* distribution.
- **Clustering based**
  - *k-Means.* It is used as both a univariate and a multivariate technique on numeric columns with k-Means++ [1] initialization for both and with $k = 3$. Once the distance of each point to its cluster is found, we apply the Box Plot Rule to this set, with the intent of finding points which belong to a particular cluster, but are actually far away from its center and not conform to its general distribution.
- **Mixture of Parametric Distributions**
  - *Gaussian Mixture Models.* This is used as both a univariate and a multivariate technique on numeric columns. Since this is an unsupervised task, it is not easy to evaluate the quality of clusters; nevertheless, as is common practice in outlier detection tasks, we used 2 components – one for the normal points and one for the outliers. Once the likelihood of each point to its component center (mean) is found, we apply the Box Plot Rule to this set, just like in the case of k-Means.
- **Non-Parametric**
  - *Histogram / Frequency based.* One possible approach as described in [5] is to compute the counts per category of a string class, and report the ones that satisfy condition (1) of the Box Plot rule. For instance, this may be useful for zip-codes, colors, days, etc. where some categories appear extremely rarely. Similarly, this can also be used for numeric columns, by binning the range of a column into buckets and applying the Box Plot Rule on the counts of each bucket.
  - *Length based.* Similar to the Histogram - based technique, for strings we applied the Box Rule on the lengths of string columns to find out rows with abnormally long lengths, which be useful in cases such as identifying exceptionally long (and out-of-distribution) zip-codes, addresses, etc.
- **Nearest Neighbor based**
  - *DBSCAN [4].* As the authors recommend, this technique should be used for similar columns, and hence, we only use it for geo-location data (i.e., latitudes and longitudes) on the Taxi data sets which comprise the pickup and drop off locations.

5 RESULTS

All our results are reproducible; to obtain them, the following commands may be used (back to back):

```
python3 clean_data.py GROUP7/bss9-579f.tsv
python3 get_outliers.py GROUP7/bss9-579f_clean.tsv
```

As mentioned earlier, DBSCAN is used on specific columns on specific data sets. One such example is given below for the `ghpb-fpea.tsv` data set (after cleaning it):

```
python3 src/dbscan_outliers.py
GROUP7/ghpb-fpea_clean.tsv pickup_latitude
pickup_longitude ghpb-fpea_dropoff
```

The detailed instructions for usage and locating the output files are given the GitHub repository.

For obvious reasons, it is not possible to list out all results here. The reader is encouraged to download the code and run the above command to obtain the full list of outliers.
A short list of results is shown below.

5.1 Data Cleaning

For the cleaning data part, we show below an example of a column that is originally identified with a string data type. However, in reality it has an integer data type. The issue is that the values are separated by commas. From the figure below one can see that our method is able to detect this and correct it. Finally, our method removes all the null values, as seen below.

```
df.select('comparable_rental_3_full_market_value').head(11)
```

```
[Row(comparable_rental_3_full_market_value='2,405,000'),
 Row(comparable_rental_3_full_market_value='4,497,000'),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value='2,142,000'),
 Row(comparable_rental_3_full_market_value='9,226,000'),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value='9,226,000'),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value='5,357,000')]
```

```
replaced_df.select('comparable_rental_3_full_market_value').head(11)
```

```
[Row(comparable_rental_3_full_market_value=2405000.0),
 Row(comparable_rental_3_full_market_value=4497000.0),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value=2142000.0),
 Row(comparable_rental_3_full_market_value=9226000.0),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value=9226000.0),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value=None),
 Row(comparable_rental_3_full_market_value=5357000.0)]
```

```
clean_df.select('comparable_rental_3_full_market_value').head(6)
```

```
[Row(comparable_rental_3_full_market_value=2405000.0),
 Row(comparable_rental_3_full_market_value=4497000.0),
 Row(comparable_rental_3_full_market_value=2142000.0),
 Row(comparable_rental_3_full_market_value=9226000.0),
 Row(comparable_rental_3_full_market_value=9226000.0),
 Row(comparable_rental_3_full_market_value=5357000.0)]
```

Above, we see that various type of non-alphanumeric symbols that are present in the data, which must be cleaned before we can proceed with any form of analysis, since these columns need to be treated as numeric columns, instead of strings.

5.2 Null Values

We implemented several regular expressions to weed out null values. A non-exhaustive list of the null values detected by our techniques has been attached below:

- "None"
- "N/A"
- "-"
- " "
- -999
- "Unknown"
- 0.00 (latitude/longitude)

For obvious reasons, it not possible to write regular expressions for each possible case. One interesting result we found was that our outlier detection algorithms were also sensitive to effective in finding null values. One such example is the value "5 UNKNOWN" – an address which presumably was written down only partially. The Histogram - based technique that we used for finding outliers in strings was able to detect it, owing the low counts of this particular value in that address column.

5.3 Outliers

For numeric columns, we show the outliers for the column `num_level_3` from the `usap-qc7e_clean.tsv` data set:

Summary	num_level_3
count	5302
Mean	239.776
stddev	267.144
min	0.0
max	2020.0

Table 1: Summary Statistics for `num_level_3`

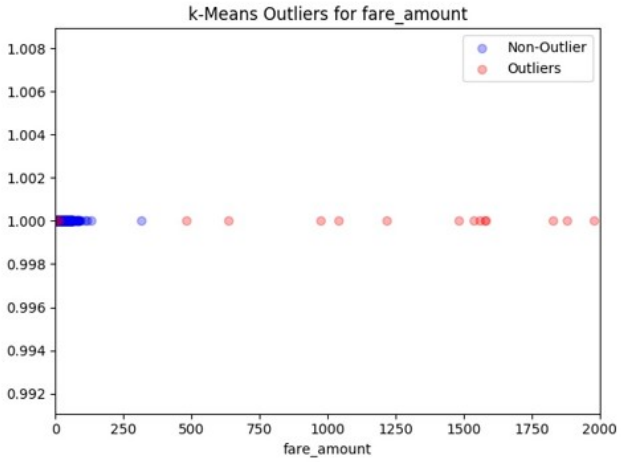
And some of the outlier rows (with their respective `rid`'s and values for the `num_level_3` column are shown below:

rid	num_level_3
847	1386.0
851	1434.0
963	1337.0
990	1344.0
1493	1522.0

Table 2: Outliers for `num_level_3`

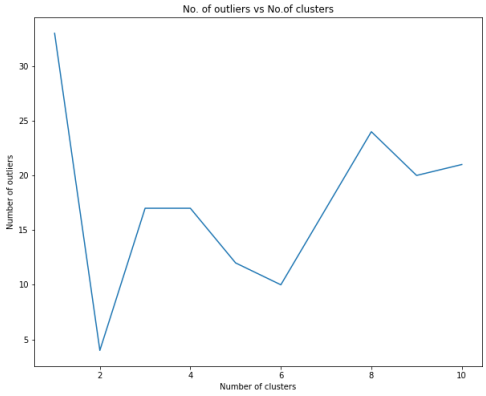
On the left in the summary statistics for the column, we can see that the mean and the standard deviation of the column are ~ 240, while the outlier rows have values ~ 1500. It is clear that these are potential outliers for that column, since they fall very clearly out of the normal range of values.

Next, we show outliers detected by our k-Means implementation follows by the Box Plot rule on the distances. As it is not possible to easily show outliers for the multivariate case, we have only shown the ones for the univariate one on the `fare_amount` column in the `ghpb-fpea.tsv` data set:

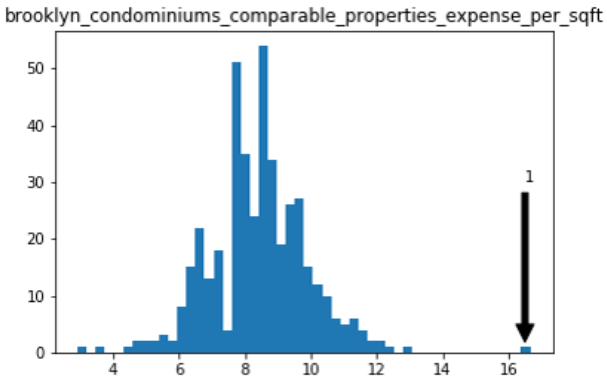


In the figure shown above, the regular values are clustered in the range ~ 0-200. This is where most of the density of the data lies; however, the points shown in red lie completely out of the range, with their values going as high as 2000 for a few cases. It is a safe bet to say that these are correctly identified outliers.

For k-Means, we also show the variation of the number of outliers detected with the number of clusters we set in the k-Means algorithm for the column `comparable_rental_2_gross_sqft` of the data set `bss9-579f.tsv`.



Lastly, we have attached one result from the Histogram-based method we applied for the `brooklyn_condominiums_comparable_properties_expense_per_sqft` in the `bss9-579f.tsv` data set:



Here, we performed binning of columns into 50 buckets, and applied the Box Plot Rule for each of the counts. As is evident, this algorithm detects points which are far away from the normal distribution of the the column, and that have very low counts for that bin.

For a full list of results, the reader is encouraged to run the code (instruction given at the beginning of the Results section).

6 DISCUSSION AND KEY STRENGTHS

The main focus of this project was on outlier detection. As was mentioned before, this study aims at performing this task in a fully automated manner, independent of any parameter inputs from the user. We followed and implemented at least one algorithm from each class of outlier detection techniques mentioned in [3].

To automate everything while retaining performance, we applied to Box Plot Rule – which is known to be extremely robust – at the end of the results given by all other (more sophisticated) algorithms like k-Means and Gaussian Mixture Models.

Also, to gain a small computation speedup, we drop all rows which have been detected as outliers by our univariate algorithms before we proceed with the multivariate analysis. This may prove to be important in kernelized algorithms and ones based on nearest neighbours, where the computation complexity increases significantly with the total number of rows.

Finally, to ensure that we do not give out increase the number of false positives while implementing such a variety of algorithms, we presented a novel approach to already existing techniques by doing a voting amongst similar techniques and only reporting rows which have been detected as outliers for more than half of the algorithms.

REFERENCES

[1] David Arthur and Sergei Vassilvitskii. 2007. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1027–1035. http://dl.acm.org/citation.cfm?id=1283383.1283494

[2] Jason Brownlee. 2016. How To Handle Missing Values In Machine Learning Data With Weka. (Jun 2016). https://machinelearningmastery.com/how-to-handle-missing-values-in-machine-learning-data-with-weka/

[3] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2007. Anomaly Detection: A Survey. (2007).

[4] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 226–231. http://dl.acm.org/citation.cfm?id=3001460.3001507

[5] Google. 2018. Locate Outliers, Google Cloud Dataprep Documentation, Google Cloud. (2018). https://cloud.google.com/dataprep/docs/html/Locate-Outliers_57344572

[6] Ming Hua and Jian Pei. 2007. Cleaning Disguised Missing Data: A Heuristic Approach. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)*. ACM, New York, NY, USA, 950–958. https://doi.org/10.1145/1281192.1281294

[7] Ming Hua and Jian Pei. 2008. DiMaC: a disguised missing data cleaning tool. In *KDD*.

[8] Sheng Li, Ming Shao, and Yun Fu. 2015. Multi-view low-rank analysis for outlier detection. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 748–756.

[9] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient Algorithms for Mining Outliers from Large Data Sets. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*. ACM, New York, NY, USA, 427–438. https://doi.org/10.1145/342009.335437

[10] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient Algorithms for Mining Outliers from Large Data Sets. *SIGMOD Rec.* 29, 2 (May 2000), 427–438. https://doi.org/10.1145/335191.335437

[11] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. 2014. Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection. *Data Mining and Knowledge Discovery* 28, 1 (01 Jan 2014), 190–237. https://doi.org/10.1007/s10618-012-0300-z

[12] Arthur Zimek, Matthew Gaudet, Ricardo J.G.B. Campello, and Jörg Sander. 2013. Subsampling for Efficient and Effective Unsupervised Outlier Detection Ensembles. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. ACM, New York, NY, USA, 428–436. https://doi.org/10.1145/2487575.2487676