Outlier Detection using K-Means and Apache Spark

Georgios Arampatzis
School of Informatics
Aristotle University of Thessaloniki
Greece
garampat@csd.auth.gr

Alexia Fytili
School of Informatics
Aristotle University of Thessaloniki
Greece
fkalexia@csd.auth.gr

1 Introduction

There is a vast variety of applications that require being able to detect whether an input of their existing data can be considered as an inlier or an outlier in comparison with the rest of the data. An inlier is usually classified as such, since it follows a similar pattern as the rest of the data. On the other hand, *an outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism* [1].

Outlier detection can be applied in many different fields, ranging from data cleaning to network intrusion and fraud detection. An outlier usually indicates an individual or a group that behaves abnormal in a network. In most applications, outliers are removed from the rest of the dataset since they are classified as noise and make it more difficult to properly analyze a network. But in some cases, like fraud detection, pointing out these outliers can prove extremely valuable.

2 Technologies used & Dataset

In this project, we will tackle the problem of outlier detection, in a dataset consisting of 47001 entries, with two columns pointing out the place of said entries in the XY axis (2-dimensional space). Since all entries are 2-dimensional in the rest of the report they will be referred to as *points.*

We are going to utilize Apache Spark (version 2.4.0) with the Scala (version 2.11.8) programming language, in order to parallelize the work and thus make it run more efficiently.

3 Preprocessing Steps

In order to detect any outliers, we need to apply a clustering algorithm and separate the data into clusters. For this reason, we applied the K-Means algorithm [2]. But before running our data through K-Means some preprocessing steps needed to be taken. The pipeline of text pre-processing consists of a number of stages that aim to improve further the results of the algorithms. The following methods were used for the pre-processing of the data used:

- Split
- toDouble
- Clean
- Normalize

In more detail, after reading the data we checked whether they were read in a 2-column format or not. In case all points were processed as a string (1-column format) we split the input into two columns and after that cast it as double in order to continue properly with the pre-processing steps. After that we check whether there was any incomplete or faulty data in our dataset. If that was the case, the entry was dropped from the dataset. Finally, we apply normalization to the data and call K-Means.

Trying to find the best parameters to run K-Means with, we concluded that we needed a relatively big number of clusters in order to spot the outliers easier. Since detecting outliers is not the only goal, but also doing it in a short time, we chose to split the dataset into clusters based on the total number of points in the dataset and not have a fixed value for all cases.

After numerous experiments we deduced that for small datasets (those that have less than 10.000 points) 100 clusters were enough to properly detect all outliers. But for bigger datasets, like the one that in our case, using 500 clusters was preferable since not only it gave great results, but also gave them in a very short time. Lastly, for datasets with an entry count greater than 50000 we decided to dynamically change the number of clusters generated with K-Means. More specifically we divide the number of points in the dataset by 1000 and then add 500. This is done in order to minimize the computational cost needed for running the K-Means algorithm in massive datasets, while also providing great results with our outlier detection method.

4 Methodology

In order to detect outliers we applied two different approaches. Firstly, we applied a cluster-based approach in order to detect

small clusters consisting of a small number of points in comparison to regular clusters. Then, we applied a distance-based approach aiming at detecting points located in a distance greater than regular cluster's points and tag them as outliers.

4.1 First Check - Cluster-Based Outlier Detection

In order to define outlier clusters, we used the approach developed by He et al. (2003) for detecting large and small clusters. This approach is based on the fact that large and small clusters will have a significant difference in the number of points, so, they take advantage of the number of points that emerged from clustering.

In more detail, as we applied their approach on our project we considered C as a set of clusters $C = \{C_1, C_2, \ldots, C_n\}$where $|C_1| > |C_2| > \ldots > |C_n|$. We defined *parameter b* as the boundary of *small* clusters, thus defining as *small* all clusters that satisfy the following condition:

$$\frac{|C_b|}{|C_{b+1}|} \geq \alpha$$

Thus, the set of *small* clusters is defined as $SC = \{C_i \mid i > b\}$. *Parameter α* defines how smaller a cluster is (regarding the number of points) compared to the rest of the clusters. In our project, we set *parameter α* to 5 meaning that a cluster will be tagged as small if it's size is 5 times smaller than the size of other clusters.

A sequence of steps was applied in order to detect *small* communities. Initially, a dataframe is created containing the number of points belonging to each cluster. Then, this dataframe is sorted in descending order based on point's quantity and for each cluster the ratio presented above is calculated. Thus, we end up with a dataframe in which the first cluster detected having a ratio value greater than or equal to *parameter α* is considered as a *small* cluster and each following cluster is considered as *small* too. It is worth noting that if the first *small* cluster is detected, then all following clusters are automatically considered as *small*, since all clusters were sorted in descending order based on the number of points appointed to them.

Thus, on our approach we consider *small* clusters as outlier clusters and we delete points belonging to them from the initial dataframe.

Figure 1 shows a representative version of the dataframe created for this approach. Cluster with id 284 consists of three points resulting in a ratio value (column *'div'* in Figure 1) equal to 14.666. This value is much smaller compared to the previous cluster's value meaning that this cluster is considered an outlier cluster and each one following cluster would be considered as outlier too.

cluster_id	pointsQTY	lead	div	IsOutlierFirstCheck
233	149	145	1.0275862068965518	false
100	145	142	1.0211267605633803	false
86	142	141	1.0070921985815602	false
31	141	141	1.0	false
314	141	140	1.0071428571428571	false
44	140	139	1.0071942446043165	false
422	139	137	1.0145985401459854	false
274	137	136	1.0073529411764706	false
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
315	51	48	1.0625	false
181	48	48	1.0	false
198	48	45	1.0666666666666667	false
269	45	44	1.0227272727272727	false
279	44	3	14.666666666666666	false
284	3	null	null	true

Figure 1: Dataframe containing each cluster's *'div'* value

4.2 Second Check - Distance-Based Outlier Detection

A number of the total outlier points is being detected by applying the above approach, but in order to detect outlier points that do not belong on *small* clusters we also used a distance-based approach.

In this approach, we calculated each point's distance from it's cluster using Euclidean Distance. Given a point $p = (p_1, p_2)$ and it's cluster center $c = (c_1, c_2)$ we calculated Euclidean Distance as:

$$\boldsymbol{d(p,c) = \sqrt{(p_1 - c_1)^2 + (p_2 - c_2)^2}}$$

Then, we calculated the average distance between all points and their respective centers and then we tagged as outliers the points that verify the following inequality:

$$\frac{d(p,c)}{avg(distance\ of\ cluster's\ points)} \geq \beta$$

With experimentation we concluded that an ideal value for *parameter β* is 8, meaning that an outlier' s point distance is 8 times greater than average points distances.

As depicted in Figure 2 the three outlier points detected from this approach have a value derived from the aforementioned inequality at least three times greater than the rest points (column 'div' in Figure 2, (sorted in descending order based on said column)).

col0	col1	div	IsOutlierSecondCheck
2.5	900.0	28.488030495263985	true
1.0	400.0	16.730990109886008	true
4.0	500.0	15.319038673340994	true
1.723	573.0	5.334672987012136	false
3.856	409.0	4.770315813299098	false
5.078	763.0	4.137815646559286	false
1.142	351.0	3.899398869549037	false
3.841	411.0	3.8916365345407073	false
3.459	558.0	3.8906928050140888	false
1.364	280.0	3.8847091575274635	false
5.048	877.0	3.7735191574615783	false
.	.	.	.
.	.	.	.

Figure 2: Descending ordered dataframe containing each point's *'div'* value

In order to set a proper value for *parameter* β we also experimented with a modified version of our project's dataset containing points either extremely close to a potential cluster or in a position we expected that will challenge our algorithm. Although, despite our efforts to "trick" the algorithm, it continued giving excellent results by returning all outlier points provided. Based on those tests, we concluded that setting *parameter* β equal to 8 is the optimal approach in our algorithm in order to detect all outliers.

5 Results

5.1 Outlier Detection

In this section we present the visualization of the algorithm's output in order to do a proper evaluation. The scripts used to visualize the results were written in python with the addition of the *numpy* [4] and *matplotlib.pyplot* [5] libraries.

First of all we present the dataset in the XY axis in order to get a clear understanding of the issue (Figure 3).

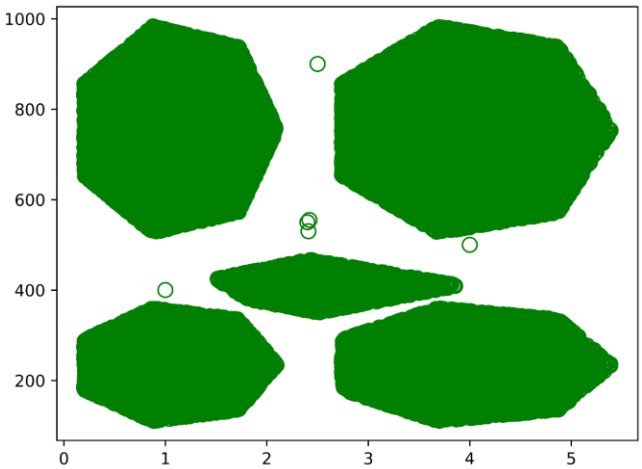

Figure 3: Complete dataset

It is clear that the dataset includes 5 main shapes consisting of a big number of points and that there are also 6 other points in between them.

After running our algorithm we tried visualizing the results in different colours to separate the outliers detected from the two different checks.

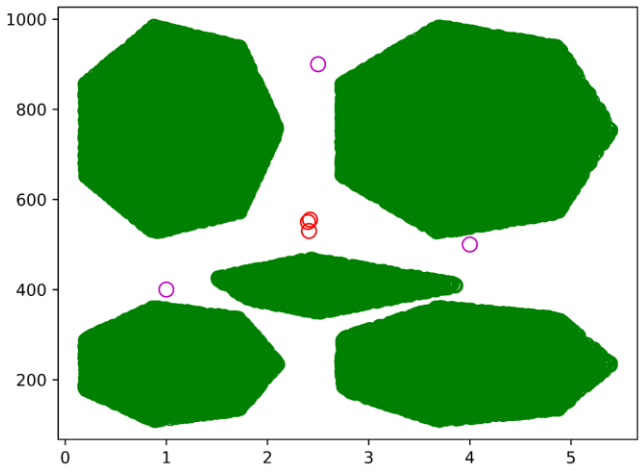

Figure 4: Outlier Detection in our dataset

Outliers found in the cluster-based check (first one) are presented in red, while outliers found in the distance-based check (second one) are presented in purple (Figure 4).

As we can clearly see the algorithm did a great job in detecting all the outliers inside the dataset.

Since we wanted to test our approach even further, we tried altering the dataset presented here. We chose to add an additional small cluster of points and also a new point extremely close to another cluster. As depicted in the following figure (Figure 5), the algorithm managed to detect again correctly all the outliers inside the dataset.

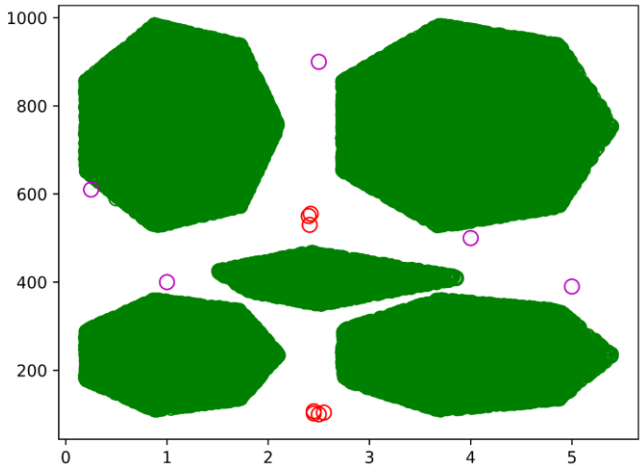

Figure 5: Outlier Detection in an altered dataset

5.2 Execution Time

Having tested the accuracy of the algorithm detecting outliers we also wanted to verify that the execution time of the whole process is equally satisfactory.

The tests were done in a Sony Vaio VPCEH2Q1E laptop with an Intel(R) Core(TM) i5-2430M CPU 2.40 GHz and a 6 GB DDR3-SDRAM, using a Windows 7 operating system.

The execution time of the whole algorithm for the dataset provided was 1min 36sec. In more detail:

Steps	Time
Data Preprocessing	0 min 12 sec
K-Means Execution	0 min 43 sec
Cluster-Based Check	0 min 16 sec
Distance-Based Check	0 min 20 sec

Table 1: Execution Time

6 Conclusion

Outlier detection is an important analytic task. In this project we devised a method to detect those outliers in a timely efficient manner. After finishing our tests, we concluded that outlier detection can be achieved with high precision utilizing K-Means through Apache Spark, while also maintaining a low execution time. Furthermore, combining cluster-based approaches with distance-based ones constitutes an effective outlier detection method capable of locating even the slightest deviation in all cluster's points in comparison to the others.

In the future we plan on implementing a solution using the Local Outlier Factor (LOF), that reflects the degree of the abnormality of any observation and one that utilizes the K-Nearest Neighbours method for another distance-based approach. It would be an interesting approach to develop a scalable algorithm that combines these two methods in such a way that it can be efficiently applied on large datasets.

REFERENCES

[1] Hawkins, D. (1980). Identification of Outliers. Chapaman and Hall. London
[2] https://spark.apache.org/docs/latest/ml-clustering.html
[3] He, Z., Xu, X., & Deng, S. (2003). *Discovering cluster-based local outliers*. *Pattern Recognition Letters*, 24(9-10), 1641-1650.
[4] https://numpy.org/doc/
[5] https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.plot.html