# Anomaly Detection in Data streams using MOA

### G Sandhya Madhuri<sup>a</sup>, Dr. M. Usha Rani<sup>b</sup>

<sup>a</sup> Research Scholar, <sup>b</sup> Research Supervisor

<sup>a, b</sup> Dept. of Computer Science, SPMVV, Tirupati, India

<sup>a</sup> sandhyamadhuri@gmail.com, <sup>b</sup> musha\_rohan@yahoo.com

Article History: Received: 10 November 2020; Revised 12 January 2021 Accepted: 27 January 2021; Published online: 5 April 2021

**Abstract:** Anomaly means anything which deviates from normal. It can be a credit card fraud or sensor alarm or a signal from a condition monitoring device. A problem like anomaly arises when we try to monitor the unusual behaviour of a machine. More number of outliers means the machine needs to be inspected. Anomaly detection in static data can be entirely different from that of streaming data.

We have some issues in anomaly detection in streaming data when compared to static data. If any off - line algorithms attempt to find anomalies in streams, it has to store the entire stream for analysis. So, there is a high probability that it will run out of memory space.

Also, streams can be infinite and evolving over a period of time because of which maintenance of high detection accuracy becomes almost impossible. In this paper we will discuss about anomaly detection in data streams and using MOA (Massive Online Analysis) tool we will analyse which algorithm derives best results.

Keywords: Massive Online Analysis, data streams, outliers, outlier detection algorithms

### 1. Introduction

Data streams are universal, ranging from sensor data to web data and clickstream data. Subsequently, there is a rich and increasing body of work onstream data mining.

Anomaly detection using simulation helps us examine the anomaly examples from big data source. In order to increase the speed of the processing time to handle massive datasets, in this paper we have demonstrated the experiments conducted on advanced distant-based outlier detection algorithms to inspect the most effective algorithms using MOA. The algorithms used in this study are Continuous Outlier Detection (COD), Micro-Cluster based COD or MCOD, Exact STream OutlierR Miner (Exact STORM), Approximate Stream Outlier Miner (Approx STORM), AbstractC algorithms. The results demonstrate MCOD algorithm can surpass other algorithms in terms of processing time and precise anomaly detection.

We have MOA (Massive Online Analysis) an open source frame work for anomaly detection in data streams. The key advantage of MOA is that it provides many recently developed data stream algorithms, with learners for multi-label classification, graph mining and outlier detection.

### 2. Moa Framework

The aim of MOA is a standard framework for running experiments in the data stream mining context by proving

- storable settings for data streams (real and synthetic) for repeatable experiments
- a set of existing algorithms and measures form the literature for comparison and
- an easily extendable framework for new streams, algorithms and evaluation methods.



### Fig 1: MOA workflow

The workflow in MOA follows the simple outline as shown below:

- **First** a data stream (feed, generator) is chosen and configured.
- Second an algorithm (e.g. a classifier) is selected and its parameters are set.
- Third the evaluation method or measure is chosen.
- **Finally** the results are acquired after executing the task.

MOA supports stream classification, stream clustering, and outlier detection[1].

#### 3. Outlier Detection In Moa

The Outliers graphical interface can be seen below. The image shows the environment where we setup any two algorithms to analyse on a data stream that is generated by MOA using RandoRBFGeneratorEvents data stream generator.

		MOA Graph	ical User Interface	- + :
Classificatio	n 🕺 Clustering 🚺 Out	liers		
Setup Vis	ualization			
Outlier Deteo	tion Algorithm Setup			
Stream	RandomRBFGenerators	vents -n	Edit	
Algorithm1	MCOD.MCOD		Edit	
Algorithm2	Angiulli, Exact STORM		Edit Clear	
Start	Stop	Import Ex	wport Weka Exp	Export CSV
		2		

Fig 2: MOA Outliers: Setup

The setup contains of three parameters:

- 1. Data stream
- 2. Algorithm 1
- 3. Algorithm 2

We have to use Edit option to change any of the above mentioned parameters.

Now, let us see the editing options of Algorithms.

Classific	Editing option: Algorithm1 ×	e
Setup	class moa.clusterers.outliers.Angiulli.ExactSTORM 📼	
Outlier D	Purpose MOA Clusteren moa.clusterens.outliens.Angiulli.ExactSTORM	
Stre		
Algorith	radius 0.1 -	
Algorith	k 50-	ear
	queryFreq 1 -	
Start	waitWinFul	кр
	windowSize	
	Help Reset to defaults	
	Cancel	

Fig: 3 Setting the parameters for each algorithm

In the above figure we can observe that to analyse the any two algorithms, the user should first select the algorithm and then change the values of each parameter of the chosen algorithm. If we move the cursor over the names of the parameter, a message box appears describing each parameter[2].

Once the algorithms are selected and parameters are set. The user needs to observe the Visualisation tab, where he can visually see both the algorithms result. The below figure shows the Visualisation tab before the user presses start button.

Start 👌 Streen	X Din 1 🔻 🖌 Poin	rs. 21 mainteachadh	Visualisation Speed	Processed:0
Stop Redraw	Y Din 2 💌 🗹 Out	lem El Waterin ar		Pause In: 100
Process time per object				
Zoom in Y Zoom of	at Y		Zoominb	Zoom out X

Fig: 4 Visualisation tab

In this tab, the user can change many options like display options i.e. either points or outliers. Also, options like wait till the window is full so as to display outliers. The user can also change the visualisation speed, dimensions of the points that are displayed and the value of the pause interval.

Now, let us observe Fig: 5 in which the visualisation of each point in the data stream is displayed in the two areas (screens) allocated for each algorithm. The screen with red colour represents the Algorithm 1 and the screen with blue colour represents Algorithm 2.

The graph at the bottom of the visualisation window shows the time taken by the algorithm to process each data point in the stream.

y-axis represents time per object

x-axis represent eachwindow slide.

* MOA	Graphical User Interfa	ce		- + ×
Classification   Clustering   Outliers				
Setup Visuelization				
Besung Screen K Din t + p	Points	Visualisation 5	peed Pro-	cessed:1000
step Redram y Dig 2 -	Outline Z waitwined		-7	an in the second
and worked a party a				
(b) f = 1 and a second seco		1.1.1.1.1		
州、		21条		
and the second second		20 M	Sec. 1	
. *			- 686 e	
salito.		1.1	Mko.	
		· · · · · · · · · · · · · · · · · · ·	9	lpi -
- 1940 - 1947			6	49.
	8		r: + z .	
18 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1			1.11	
a at the second second		1. se - s	a 3	
-0 <sup>123</sup> 36	- *	111	0°11	5×1
	· · · ·	1.1	- A.	• .*
		•		
Evaluation				
Process time per object		_		
Zoom in Y Zoom out Y			nom in X	Zoon out X
20				
x				
1 A.				
2.02 2000 10002	ED XD	3100	2022	210

Fig: 5 Visualisation tab displaying the data points and outliers

After 1000 window slide, that means the pause interval specified by the user, the visualisation pauses for the user to notice the outcome from each algorithm. We can observe that red coloured points are the outliers and the grey coloured points are inliers or normal points.

The user can also observe a red circle around some outlier points. This indicates that this particular point has been changed from an inlier to an outlier. In the same way we can also observe a black circle around some grey points which indicate that this particular point was an outlier and now has been changed to an inlier. Observe Fig: 6

If we click on any outlier point a popup window is displayed showing information about that point. Such information is specific to algorithm[3].



Fig: 6 Outlier information

We can continue the analysis by pressing resume button, and observe the visualisation of outliers. To stop the visualisation we have to press stop button.

7	MOA Graphical User Interface	- + X
Classificatio	n Clustering Outliers	
Setup Visi	ualization	
Outlier Deter	tion Algorithm Setup	
Stream	Pandom2010 Separator/Pyents-to Edit	
Algorithm1	MCODUMCOD Fait	
Algorithm2	Anglull ExactST0EM Edit Clear	
start	Stop mpart Expart Woka Exp	Export CSV
Algorithm 1. Algorithm 0. Algorithm 1. Algorithm 0. Algorithm 1.	processo Lime per object (mp): 4-510 process time per abject (mp): 8-645 process Lime per abject (mp): 8-565 process Lime per abject (mp): 8-567 processo Lime per abject (mp): 8-568	*
Algorithmi St	atistics:	
Notes alway Notes alway Notes both (San: 1000)	5 uniter: 640 (64 (6) 8 uniter: 67 (6.7%) inlier and outlier: 264 (25.4%)	
Total range Nos memory Total proce	queries: 617 usage: 20 MB sa Lime: C.65 me	
Algorithe2 St	stistics:	
Notes alway Notes alway Notes both (Sum: 1000)	s inlier: 645 (64,54) s cuttion: 87 (8,74) inlier and cutlier: 264 (28,74)	
Total range Nua nunsiry Total proce	quartes, 1000 souge: 20 MB as filey, 2,37 ms	-
	Export as .txt file	12

Fig: 7 Setup tab with statistical information

After this, we can observe some statistical information about each algorithm in the setup tab as in the *Fig*:7. The information includes processing time, Total range queries, Maximum memory usage, information about nodes that are inlier and outlier.

### 4. Analysis Of Data Stream In Moa

We have taken the stream generator by RandomRBFGeneratorEvents in MOA tool and tested MCOD algorithm with all the other algorithms[iii]. Given below is the screenshot of the parameters set for the data stream generator.

0.494	Participal Generations a mendiore routial	baars fainting stream.		
	modelkandombeed		1 =	111
Argen	instanceRandor/Geed		5.2	
Algue	numCluster			1 A A
514	numClusterRange		1.2	
igers.	kernelRadius	0.071 1	-	
lgori lgori	kenelfadusfange	0.13 1	-	
Lgori Lgori Lgori L	densityflange	0.152 (		
Lgorii Lgorii Lgorii	spend	1.5	00 I	
lasrs	thelp.	Reset to defaults		
Dutt True Fala		Cancel	œ	1
Nun on	process Time: 8,15 m			

Fig: 8 Parameters in Stream Generator

In the above figure Fig: 8 we have set the parameters such as number of clusters that can be formed are 6, the Kernel radius as 0.071, Range of the radius as 0.13 etc. Using these parameters, a data stream is generated with sliding window size as 1000[4].

Now let us examine the results of each algorithm we have tested with MCOD. Observe the figures below.

### 1. MCOD and Abstract C algorithms

WILD STOLEN	
Deame Sneehrt X Dz I B C horts Sage Bolan y Dz 2 C Dalay C Anthenio Pacesso 2000	
fvilution	
Process for e-per object Zerem on T Zerem on T	Zaan in X Zoom ost X
Fig: 9 Visualisation of MCO	D and Abstract C
Fig: 9 Visualisation of MCO	D and Abstract C
Fig: 9 Visualisation of MCO Outlier Detection Algorithm Setup Stream 115 - K 6 - R 0.071 - r 0.13043 - d 0.152	D and Abstract C
Fig: 9 Visualisation of MCO Outlier Detection Algorithm Setup Stream 115 - K 6 - R 0.071 - r 0.13043 - d 0.152 Algorithm1 MCOD.MCOD	D and Abstract C
Fig: 9 Visualisation of MCO outlier Detection Algorithm Setup Stream 115 - K 6 - R 0.071 - r 0.13043 - d 0.152 Algorithm1 MCOD.MCOD Algorithm2 Abstract CabstractC	D and Abstract C
Fig: 9 Visualisation of MCO Outlier Detection Algorithm Setup Stream 115 - K 6 - R 0.071 - r 0.13043 - d 0.152 Algorithm1 MCOD.MCOD Algorithm2 AbstractC.AbstractC	D and Abstract C
Fig: 9 Visualisation of MCO outlier Detection Algorithm Setup Stream 1ts - K 6 - R 0.071 - r 0.13043 - d 0.152 Algorithm1 MCOD.MCOD Algorithm2 AbstractC.AbstractC Start Stop Import	D and Abstract C
Fig: 9 Visualisation of MCO	D and Abstract C
Fig: 9 Visualisation of MCO	D and Abstract C
Fig: 9 Visualisation of MCO	D and Abstract C

We can clearly observe that in the above two algorithms, almost all the parameters of the two algorithms are same except the processing time per object. Clearly MCOD has less processing time compared to AbstractC.

ys inlier: 1403 (70.2%) ys outlier: 233 (11.7%) inlier and outlier: 364 (18.2%)

Total range queries: 2000 Max memory usage: 583 MB Total process time: 6.31 ms

### 2. MCOD and Approx STORM algorithms

		CIRCUDCIDID REGISCUOT	Capering	Land Public Lotes			
		Carrieran separat		Conception			
		Municetian Fra					
Resume Screenshot 3	Dim 1 C Points	visualisation spe	NO Processed 20	100			
Stop Redraw y	Dim 2 📴 🖬 Outliers		Pause in:	1000			
8			29 CC				
- N 🦷 (N	. •		1 N.		·		
· · · ·	14		1.1	11 A.	21 A.		
				* .ii	<u>k</u> . •		
- 施				40			
duation ocess time per object							
Zoom in Y Zoom out 1						Zoom in X Z	porn out X
3							
00 E 1276	1500 3750 5800	6250 77	10 8750	1800 1	1210 11900	13750 15080	26295
Outlier Detection	on Algorithm Setup						Setup
Outlier Detection	on Algorithm Setup hts –K 6 –R 0.071 –	r 0.13043 –d (	).15217 –n	Edit		]	Setup
Outlier Detection	on Algorithm Setup hts –K 6 –R 0.071 – MCOD.MCOD	-r 0.13043 -d (	).15217 – n	Edit Edit		]	Setup
Outlier Detection Stream Algorithm1 Algorithm2	n Algorithm Setup hts -K 6 -R 0.071 - MCOD.MCOD Angiulli.ApproxSTC	-r 0.13043 -d ( DRM	).15217 – n	Edit Edit Edit	Clear	]	Setup
Outlier Detection Stream Algorithm1 Algorithm2 Start	n Algorithm Setup 1ts -K 6 -R 0.071 - MCOD.MCOD Angiulli.ApproxSTC Stop	r 0.13043 -d ( DRM	).15217 -n Exp	Edit Edit Edit	<b>Clear</b> eka Explorer		Setup
Outlier Detection Stream Algorithm1 Algorithm2 Start Algorithm 0, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1,	Algorithm Setup Its -K 6 -R 0.071 - MCOD.MCOD Angiulii ApproxSTC Stop process time per o process time per o	r 0.13043 -d ( DRM bject (ms): 1 bject (ms): 2 bject (ms): 2 bject (ms): 2 bject (ms): 2 bject (ms): 2	0.15217 - n .032 .032 .037 .062 .777 .785 .152 .889 .543	Edit Edit Edit ort W	Clear eka Explorer		Setup
Outlier Detection Stream Algorithm1 Algorithm2 Start Algorithm6, Algorithm6, Algorithm6, Algorithm6, Algorithm6, Algorithm6, Algorithm6, Algorithm6, Algorithm7, A	Algorithm Setup 115 - K 6 - R 0.071 - MCOD.MCOD Anglulli.ApproxSTC Stop process Line per process Line per	rr 0.13043 -d ( DRM bject (ms): bject (ms): bject (ms): bject (ms): bject (ms): bject (ms): bject (ms):	0.15217 - n 0.15217 - n 0.032 0.037 0.052 0.777 7.785 0.889 0.543	Edit Edit Oort W	Clear eka Explorer		Setup
Outlier Detection Stream Algorithm1 Algorithm2 Start Algorithm 1, Algorithm 4, Algorithm 4, Algorithm 4, Algorithm 4, Algorithm 4, Algorithm 4, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 5, Algorithm 5, Algo	Its -K 6 - R 0.071 - MCOD.MCOD Angluill Approx5TG Stop process Like per c process Like per c sources Like pe	r 0.13043 -d ( DRM limport bject (ms): 1 bject (ms): 1 bject (ms): 6 bject (ms): 2 bject	0.15217 - n .032 .032 .037 .032 .052	Edit Edit Ort W	Clear eka Explorer		Setup
Outlier Detecti Stream Algorithm1 Algorithm2 Start Algorithm 1, Algorithm 2, Algorithm 3, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 1, Algorithm 2, Algorithm 3, Algorithm 3, Algorithm 3, Algorithm 3, Algorithm 3, Algorithm 3, Algorithm 3, Algorithm 3, Algorithm 3, Algorithm 1, Algorithm 3, Algorithm 3, Algori	Algorithm Setup 115 - K 6 - R 0.071 - 115 - K 6 - R 0.071 - MCOD.MCOD Anglulli.Approx510 Stop process Line per process Line per sources - Line per process Line per s outlier: 233 (11 inilier and outlier guera 507 MB s Line: 455 mS	r 0.13043 - d ( DRM import ubject (ms) : ubject (ms): ubject (mbject (mbjec	0.15217 - n Exp .032 .037 .052 .777 .785 .152 .889 .543	Edit Edit Ort W	Clear eka Explorer		Setup
Outlier Detecti Stream Algorithm1 Algorithm2 Start Start Algorithm 0, Algorithm 0, Algorithm 1, Algorithm 1,	Algorithm Setup Its - K 6 - R 0.71 - MCOD.MCOD Anglulli ApproxStG Stop process time per c process time per c process time per c process time per c process time per c atistics: s initer: 4483 (7 s outlier: 233 (11) initer and outlies queries: 1315 s usige:: 507 MB ss time: 0.55 mS	r 0.13043 - d ( DRM Import Diplect (ms): 1 Diplect (ms): 2 Di	0.15217 - n .032 .0337 .0337 .032 .152 .152 .152 .543	Edit Edit Edit	Clear eka Explorer		Setup
Outlier Detecti Stream Algorithm 1 Algorithm 2 Algorithm 4, Algorithm 5, Algorithm 6, Algorithm 5, Algorithm 6, Algorithm 7, Algorithm	Algorithm Setup Its K 6 - R 0.071 MCOD.MCOD Anglulli ApproxSto Stop process line per o process line per o s outlier: 233 (III) s outlier: 233 (III) s of the soutlier s of the soutlier s of the soutlier of the soutlier s of the soutlier: 239 (III) s of the soutlier: 239 (III) s of the soutlier: 239 (III) s outlier: 239 (III)	r 0.13043 - d ( DRM import bbject (ms): i bbject (	0.15217 - n .032 .037 .037 .057 .785 .152 .889 .543	Edit Edit Edit	Clear eka Explorer		Setup
Outlier Detection Stream Algorithm 1 Algorithm 2 Start Start Start Algorithm 2 Algorithm 3 Algorithm 3	an Algorithm Setup 115 - K 6 - R 0.071 - 115 - K 6 - R 0.071 - MCOD.MCOD Anglulli.ApproxSTC Stop process time per c process time per c process time per c process time per c stistics: s initer 1493 (Fr gueries; 1783 (Here s time: - 235 (Here) s time: - 235 (Here) gueries; 1783 (Here) s time: - 235 (Here) s time:	r 0.13043 -d ( DRM b)ect (ms): 1 b)ect (ms): 2 b)ect	0.15217 - n .032 .037	Edit Edit Oort W	Clear		Setup

Fig: 12 Comparison of MCOD and Approx STORM

If we observe the above two figures we can clearly understand that the Memory usage of both the algorithms it is same. Also, we can see that the number of Outliers found is also approximately same for both the algorithms. But, if we observe the processing time per object MCOD has outdone Approx STORM algorithm[5].

### 3. MCOD and Exact STORM algorithms

•	MOA Grap	acal User Interface	
	Classification Regression	Clustering Outlines Concept Drift	
	Setup	Visualization	
Resume Screenshot X Dim 1 C Stop Redraw Y Dim 2 C	Points     WaitWinFull     Outliers	Processed 2000 Pause In: 1000	
valuation Process time per object Zoom in Y Zoom out Y		•	Zoom in X Zoom out X
230 145 	170 900 610 700	อร์ส เช่นอ เปล่อ	1790 1994 196





Fig: 14 Comparison of MCOD and Exact STORM

By comparison of these two algorithms we have observed that the outlier detection, memory usage etc are all similar except processing time. MCOD is the faster in processing when compared to Exact STORM[6].

## 4. MCOD and Simple COD algorithms







Fig: 16 Comparison of MCOD and Simple COD

If we observe the comparison of these two algorithms we see that all other aspects like memory usage and outlier nodes are similar except the processing speed. MCOD is faster than Simple COD. Thus, we conclude that MCOD is faster in its processing speed when compared to all other algorithms, even though all other parameters are similar.

### 5. Conclusion

During our analysis in finding the best algorithm in MOA in case of processing time, outliers, inliers, memory usage etc., we have concluded that MCOD is the best possible algorithm.

### References

- 1. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive Online Analysis," J. Mach. Learn. Res., 2010.
- 2. S. V. V. Reddy, T. Harshita, S. A. Koneru, and K. Ashesh, "Outlier detection in data streams using MCOD algorithm," 2018, doi: 10.1109/ICATCCT.2017.8389156.
- 3. P. T. Darshana Parikh, "Data Mining & Data Stream Mining Open SOURCE Tools.pdf," Int. J. Innov. Res. Sci. Eng. Technol., 2013.
- 4. Bifet, G. Holmes, and B. Pfahringer, "MOA-TweetReader: Real-time analysis in twitter streaming data," 2011, doi: 10.1007/978-3-642-24477-3\_7.
- P. M. Arunkumar and S. Kannimuthu, "Mining big data streams using business analytics tools: A bird's eye view on MOA and SAMOA," Int. J. Bus. Intell. Data Min., 2020, doi: 10.1504/IJBIDM.2020.108761.
- M. J. Bah, H. Wang, M. Hammad, F. Zeshan, and H. Aljuaid, "An effective minimal probing approach with micro-cluster for distance-based outlier detection in data streams," IEEE Access, 2019, doi: 10.1109/ACCESS.2019.2946966