

streamMOA: Interface to Algorithms from MOA for stream

Michael Hahsler
Southern Methodist University

John Forrest
Microsoft

Matthew Bolaños
Microsoft

Abstract

This packages provides an interface for several algorithms from the Massive Online Analysis (MOA) framework to be used in **stream**. This vignette contains some examples.

Keywords: data stream, data mining, clustering, MOA.

1. Introduction

Please refer to the vignette in package **stream** for an introduction to data stream mining in R. In this vignette we give two examples that show how to use the **stream** framework being used from start to finish. The examples encompasses the creation of data streams, preparation of data stream clustering algorithms, the online clustering of data points into micro-clusters, reclustering and finally evaluation. The first example shows how compare a set of data stream clustering algorithms on a static data set. The second example shows how to perform evaluation on a data stream with concept drift (clusters evolve over time).

2. Experimental Comparison on Static Data

First, we set up a static data set. We extract 1500 data points from the Bars and Gaussians data stream generator with 5% noise and put them in a `DSD_Memory`. The wrapper is used to replay the same part of the data stream for each algorithm. We will use the first 1000 points to learn the clustering and the remaining 500 points for evaluation.

```
R> library("stream")  
R> stream <- DSD_Memory(DSD_BarsAndGaussians(noise=0.05), n=5500)  
R> stream
```

Memory Stream Interface

Class: DSD_Memory, DSD_R, DSD_data.frame, DSD

With 4 clusters and 0 outliers in 2 dimensions

Contains 5500 data points - currently at position 1 - loop is FALSE

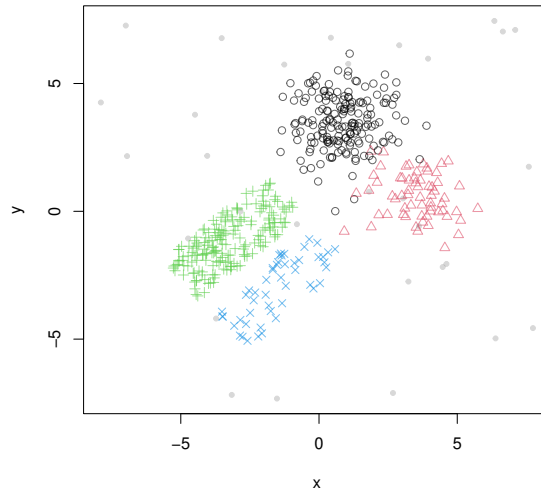


Figure 1: Bar and Gaussians data set.

```
R> plot(stream)
```

Figure 6 shows the structure of the data set. It consists of four clusters, two Gaussians and two uniformly filled rectangular clusters. The Gaussian and the bar to the right have 1/3 the density of the other two clusters.

We initialize four algorithms from **stream**. We choose the parameters experimentally so that the algorithm produce each (approximately) 100 micro-clusters.

```
R> sample <- DSC_TwoStage(micro=DSC_Sample(k=100), macro=DSC_Kmeans(k=4))
R> window <- DSC_TwoStage(micro=DSC_Window(horizon=100), macro=DSC_Kmeans(k=4))
R> dstream <- DSC_DStream(gridsize=.7)
R> dbstream <- DSC_DBSTREAM(r=.45)
```

We will also use two MOA-based algorithms available in package **streamMOA**.

```
R> library("streamMOA")
R> denstream <- DSC_DenStream(epsilon=.5, mu=1)
R> clustream <- DSC_CluStream(m=100, k=4)
```

We store the algorithms in a list for easier handling and then cluster the same 1000 data points with each algorithm. Note that we have to reset the stream each time before we cluster.

```
R> algorithms <- list(Sample=sample, Window=window, 'D-Stream'=dstream,
+ DBSTREAM=dbstream, DenStream_MOA=denstream, CluStream_MOA=clustream)
R> for(a in algorithms) {
+   reset_stream(stream)
+   update(a, stream, 5000)
+ }
```

We use `nclusters()` to inspect the number of micro-clusters.

```
R> sapply(algorithms, nclusters, type="micro")
```

	Sample	Window	D-Stream	DBSTREAM	DenStream_MOA
	100	100	98	161	144
CluStream_MOA	100				

All algorithms except DenStream produce around 100 micro-clusters. We were not able to adjust DenStream to produce more than around 50 micro-clusters for this data set.

To inspect micro-cluster placement, we plot the calculated micro-clusters and the original data.

```
R> op <- par(no.readonly = TRUE)
R> layout(mat=matrix(1:6, ncol=2))
R> for(a in algorithms) {
+   reset_stream(stream)
+   plot(a, stream, main=description(a), type="micro")
+ }
R> par(op)
```

Figure 2 shows the micro-cluster placement by the different algorithms. Micro-clusters are shown as red circles and the size is proportional to each cluster's weight. Reservoir sampling and the sliding window randomly place the micro-clusters and also a few noise points (shown as grey dots). Clustream also does not suppress noise and places even more micro-clusters on noise points since it tries to represent all data as faithfully as possible. D-Stream, DenStream and DBSTREAM all suppress noise and concentrate the micro-clusters on the real clusters. D-Stream is grid-based and thus the micro-clusters are regularly spaced. DBSTREAM produces a similar, almost regular pattern. DenStream produces one heavy micro-cluster on one cluster, while using a large number of micro clusters for the others. It also has problems with detecting the rectangular low-density cluster.

It is also interesting to compare the assignment areas for micro-clusters created by different algorithms. The assignment area is the area around the center of a micro-cluster in which points are considered to belong to the micro-cluster. In case that a point is in the assignment area of several micro-clusters, the closer center is chosen. To show the assignment area we add `assignment=TRUE` to `plot`. We also disable showing micro-cluster weights to make the plot clearer.

```
R> op <- par(no.readonly = TRUE)
R> layout(mat=matrix(1:6, ncol=2))
R> for(a in algorithms) {
+   reset_stream(stream)
+   plot(a, stream, main=description(a), assignment=TRUE, weight=FALSE, type="micro")
+ }
R> par(op)
```

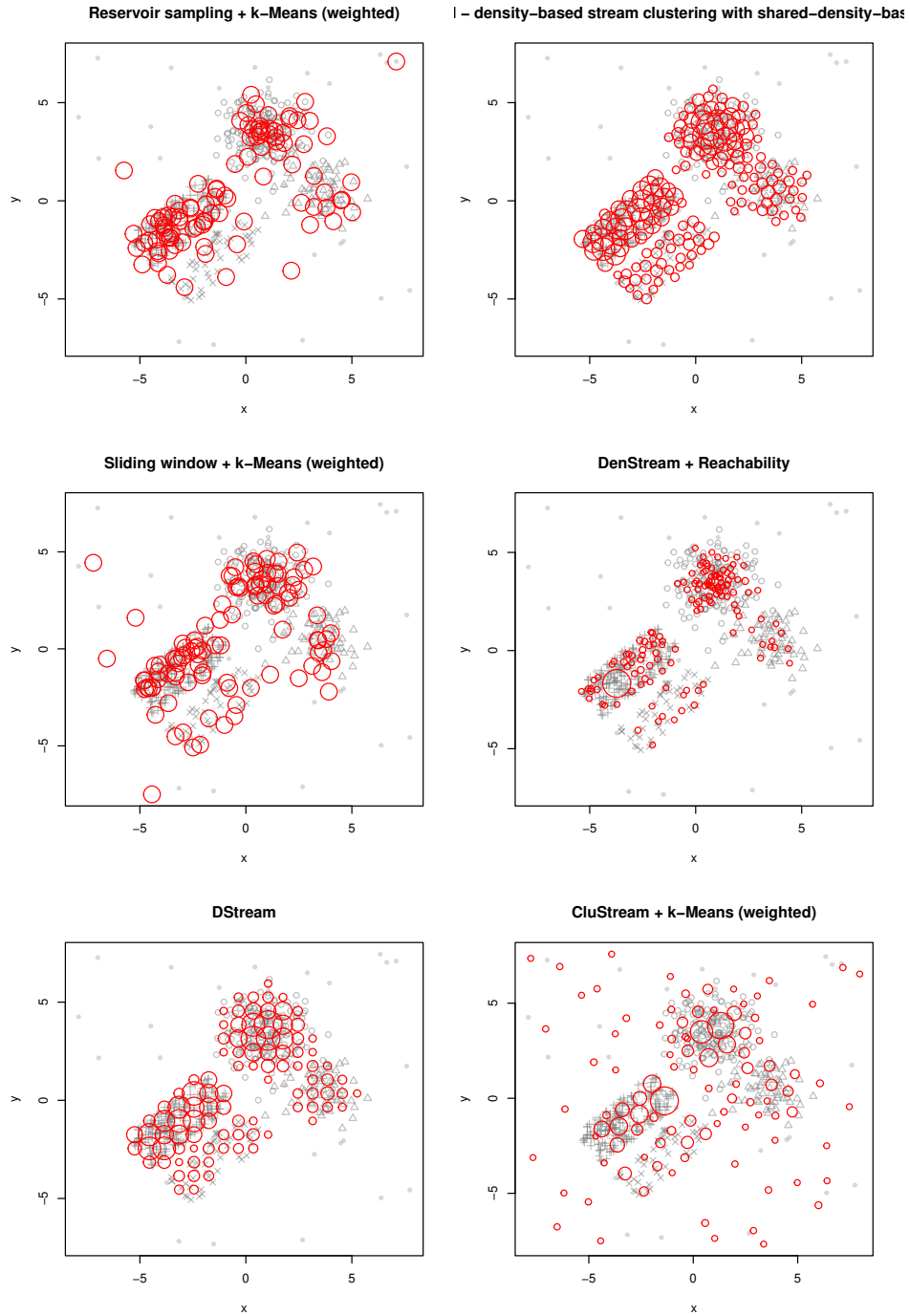


Figure 2: Micro-cluster placement for different data stream clustering algorithms.

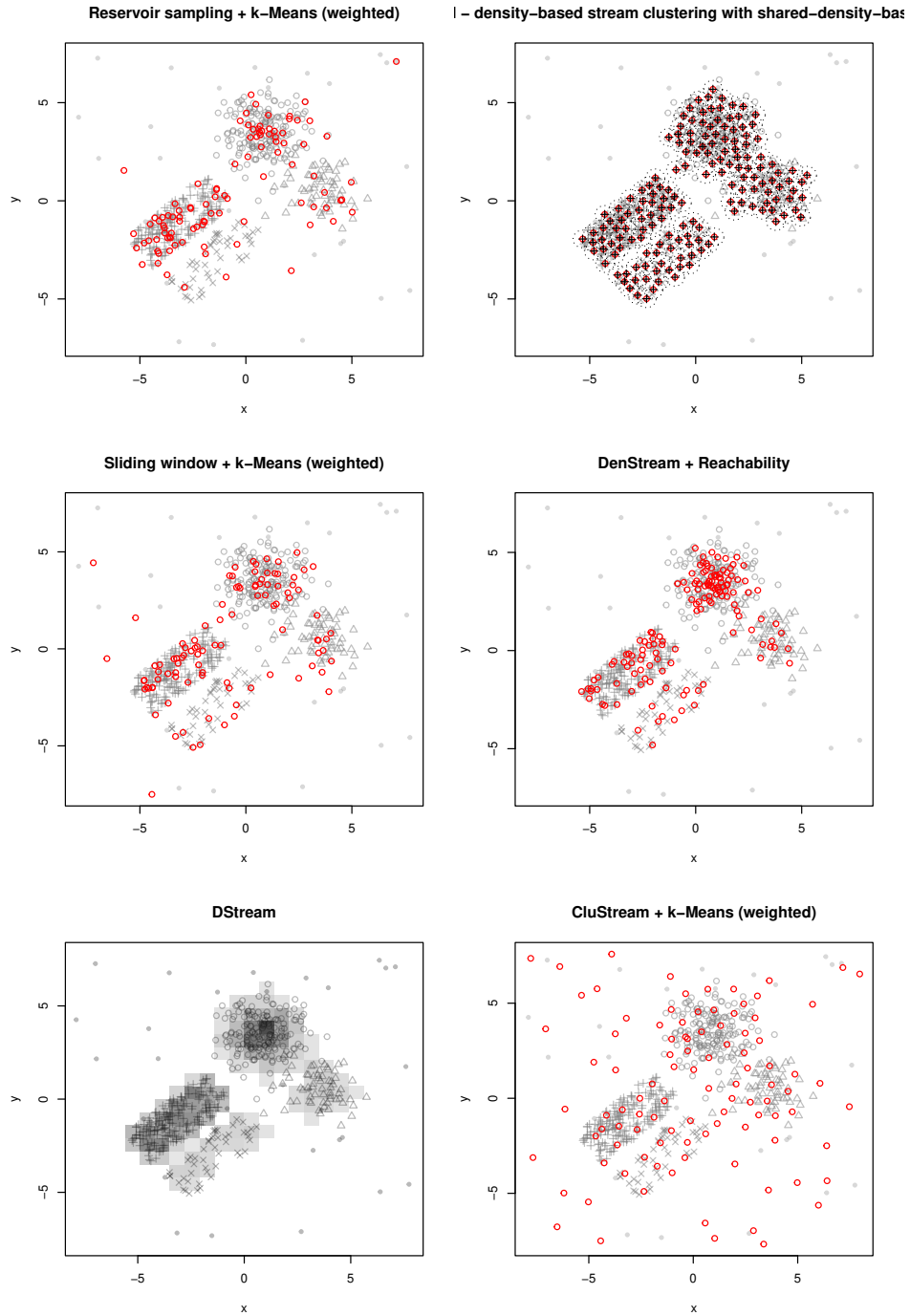


Figure 3: Micro-cluster assignment areas for different data stream clustering algorithms.

Figure 3 shows the assignment areas as dotted circles around micro-clusters. Reservoir sampling and sliding window does not provide assignment areas and data points are always assigned to the nearest micro-cluster. D-Stream is grid-based and shows the assignment area as grey boxes. DBSTREAM uses the same radius for all micro-clusters, while DenStream and CluStream calculate the assignment area for each micro-cluster.

To compare the cluster quality, we can check for example the micro-cluster purity, the sum of squares and the average silhouette coefficient. Note that we reset the stream to position 1001 since we have used the first 1000 points for learning and we want to use data points not seen by the algorithms for evaluation.

```
R> sapply(algorithms, FUN=function(a) {
+   reset_stream(stream, 1001)
+   evaluate(a, stream,
+     measure=c("numMicroClusters", "purity", "SSQ", "silhouette"),
+     n=500, assignmentMethod = "auto", type = "micro")
+ })
```

	Sample	Window	D-Stream	DBSTREAM	DenStream_MOA
numMicroClusters	100.000	100.000	98.000	161.000	144.0000
purity	0.972	0.960	0.973	0.985	0.9778
SSQ	96.090	100.483	52.673	32.194	66.3886
silhouette	0.159	0.148	0.168	0.236	0.0626
	CluStream_MOA				
numMicroClusters	100.000				
purity	0.959				
SSQ	78.497				
silhouette	0.162				

We need to be careful with the comparison of these numbers, since they depend heavily on the number of micro-clusters with more clusters leading to a better value. Therefore, a comparison with DenStream is not valid. We can compare the measures of the other algorithms since the number of micro-clusters is close. Sampling and the sliding window produce very good values for purity, CluStream achieves the highest average silhouette coefficient and DBSTREAM produces the lowest sum of squares. For better results more data and cross-validation could be used.

Next, we compare macro-clusters. D-Stream, DenStream, DBSTREAM and CluStream have built-in reclustering strategies. D-Stream joins adjacent dense grid cells to form macro-clusters. DenStream and DBSTREAM use the reachability concept (from DBSCAN). CluStream used weighted k -means clustering (note that we used $k = 4$ when we initialized `DSC_DenStream` above). For sampling and window we apply here weighted k -means reclustering with $k = 4$, the true number of clusters.

```
R> op <- par(no.readonly = TRUE)
R> layout(mat=matrix(1:6, ncol=2))
R> for(a in algorithms) {
+   reset_stream(stream)
```

```
+ plot(a, stream, main=description(a), type="both")
+ }
R> par(op)
```

Figure 4 shows the macro-cluster placement. Sample, window and CluStream use k -means reclustering and therefore produce exactly four clusters. However, the placement is off, splitting a true cluster and missing one of the less dense clusters. DenStream, DBSTREAM and D-Stream identify the two denser clusters correctly, but split the lower density clusters into multiple pieces.

```
R> sapply(algorithms, FUN=function(a) {
+   reset_stream(stream, 1001)
+   evaluate(a, stream, measure = c("numMacroClusters", "purity", "SSQ", "cRand"),
+     n = 500, assign = "micro", type = "macro")
+ })
```

	Sample	Window	D-Stream	DBSTREAM	DenStream_MOA
numMacroClusters	4.000	4.000	7.000	2.000	4.000
purity	0.756	0.823	0.840	0.656	0.778
SSQ	1654.907	1053.646	871.957	2029.569	1851.367
cRand	0.491	0.652	0.855	0.618	0.544
	CluStream_MOA				
numMacroClusters	4.000				
purity	0.739				
SSQ	998.935				
cRand	0.487				

The evaluation measures at the macro-cluster level reflect the findings from the visual analysis of the clustering with D-Stream producing the best results.

3. Experimental Comparison using an Evolving Data Stream

In this section we compare different clustering algorithms on an evolving data stream. We use `DSD_Benchmark(1)` which creates two clusters moving in two-dimensional space. One moves from top left to bottom right and the other one moves from bottom left to top right. Both clusters overlap when they meet exactly in the center of the data space.

```
R> set.seed(0)
R> stream <- DSD_Memory(DSD_Benchmark(1), 5000)
```

Figure 5 illustrates the structure of the data stream. Next, we define the clustering algorithms.

```
R> algorithms <- list(
+   'Sample' = DSC_TwoStage(micro=DSC_Sample(k=100, biased=TRUE),
+     macro=DSC_Kmeans(k=2)),
+   'Window' = DSC_TwoStage(micro=DSC_Window(horizon=100, lambda=.01),
```

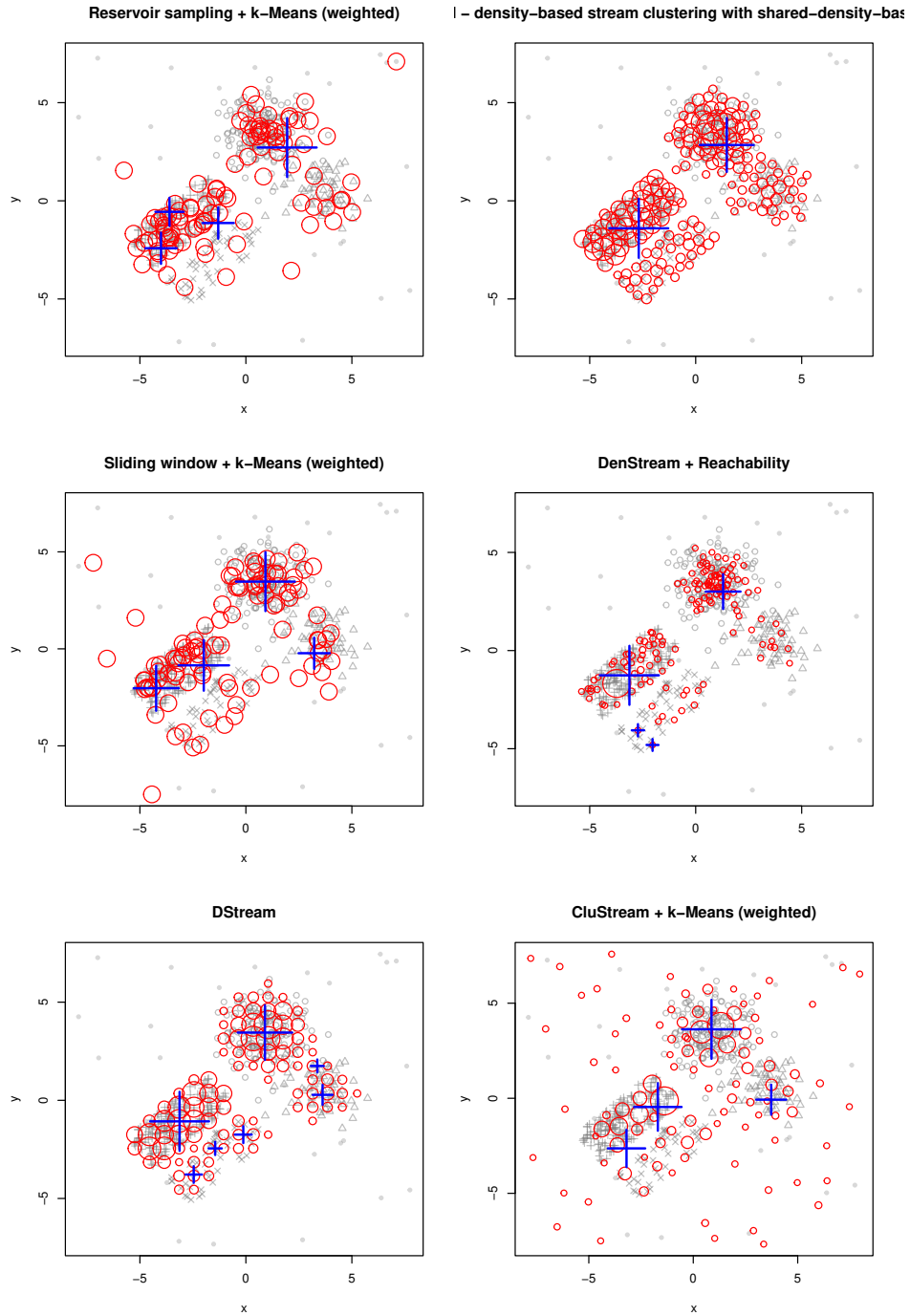


Figure 4: Macro-cluster placement for different data stream clustering algorithms

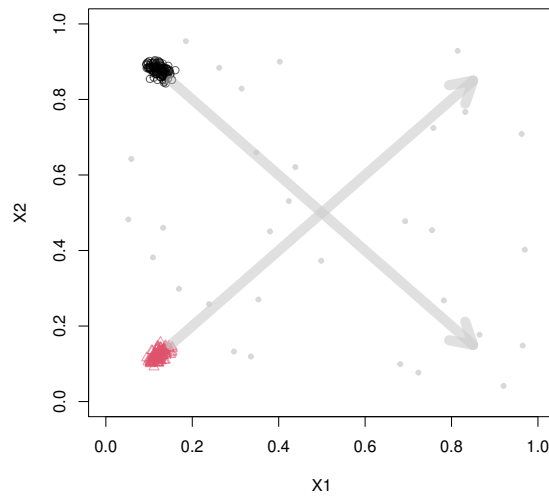


Figure 5: Data points from `DSD_Benchmark(1)` at the beginning of the stream. The two arrows are added to highlight the direction of movement.

```
+   macro=DSC_Kmeans(k=2),
+   'DBSTREAM' = DSC_DBSTREAM(r=.05, lambda=.01),
+   'D-Stream' = DSC_DStream(gridsize=.1, lambda=.01),
+
+   'DenStream' = DSC_DenStream(epsilon=.1, lambda=.01),
+   'CluStream' = DSC_CluStream(m=100, k=2)
+ )
```

We perform the evaluation using `evaluate_cluster` which performs clustering and evaluates clustering quality every `horizon=250` data points. For sampling and window we have to specify a macro-clustering algorithm. We use k -means with the true number of clusters $k = 2$.

```
R> n <- 5000
R> horizon <- 250
R> evaluation <- lapply(algorithms, FUN=function(a) {
+   reset_stream(stream)
+   evaluate_cluster(a, stream,
+     type="macro", assign="micro",
+     measure=c("numMicro", "numMacro", "SSQ", "cRand"),
+     n=n, horizon=horizon)
+ })
```

First, we look at the development of the corrected Rand index over time.

```
R> Position <- evaluation[[1]][,"points"]
```

```
R> cRand <- sapply(evaluation, FUN=function(x) x[, "cRand"])
R> cRand
```

	Sample	Window	DBSTREAM	D-Stream	DenStream	CluStream
[1,]	NA	NA	NA	NA	NA	NA
[2,]	0.8078	0.8098	0.621	0.97376	0.7050	0.0397
[3,]	0.8355	0.8339	0.858	0.10320	0.8501	0.0576
[4,]	0.7860	0.7856	0.990	0.99042	0.9364	0.1714
[5,]	0.7745	0.7735	0.863	0.92194	0.8943	0.1518
[6,]	0.0820	0.8371	0.928	0.16580	0.8398	0.0396
[7,]	0.8833	0.8833	0.992	1.00000	0.7061	0.0957
[8,]	0.8164	0.8166	0.887	0.73767	0.1149	0.0756
[9,]	0.8365	0.8365	0.818	0.88573	0.5178	0.1544
[10,]	0.7963	0.8006	0.907	0.26502	0.1956	0.0454
[11,]	0.3614	0.2826	0.204	0.11440	0.1157	0.1075
[12,]	0.0874	0.0759	0.178	0.19888	0.0258	0.0141
[13,]	0.0769	0.7489	0.334	0.33434	0.4843	-0.0214
[14,]	0.8903	0.8903	0.154	-0.00121	0.2381	0.1541
[15,]	0.8625	0.8646	0.163	0.19710	0.0745	0.0125
[16,]	0.7710	0.7710	0.861	0.94826	0.0524	0.1315
[17,]	0.8451	0.8451	0.801	0.13243	0.1989	0.0452
[18,]	0.8194	0.8170	0.882	0.96493	0.0502	0.1256
[19,]	0.8703	0.8703	0.799	0.76705	0.6753	0.1190
[20,]	0.8019	0.8021	0.882	0.69115	0.7122	0.0313

```
R> matplot(Position, cRand, type="l", lwd=2)
R> legend("bottomleft", legend=names(evaluation),
+ col=1:6, lty=1:6, bty="n", lwd=2)
```

```
R> boxplot(cRand, las=2, cex.axis=.8)
```

And then we compare the sum of squares.

```
R> SSQ <- sapply(evaluation, FUN=function(x) x[, "SSQ"])
R> SSQ
```

	Sample	Window	DBSTREAM	D-Stream	DenStream	CluStream
[1,]	NA	NA	NA	NA	NA	NA
[2,]	0.243	0.225	0.332	0.154	0.510	0.266
[3,]	0.334	0.356	0.291	1.092	0.825	0.560
[4,]	0.274	0.263	0.374	0.671	1.108	1.271
[5,]	0.317	0.200	0.563	0.262	1.441	2.272
[6,]	9.193	0.390	0.452	1.433	2.749	2.337
[7,]	0.620	0.507	0.529	0.452	3.656	2.895
[8,]	0.662	0.373	0.696	0.487	4.758	3.278
[9,]	0.816	0.875	0.499	1.416	3.572	3.387

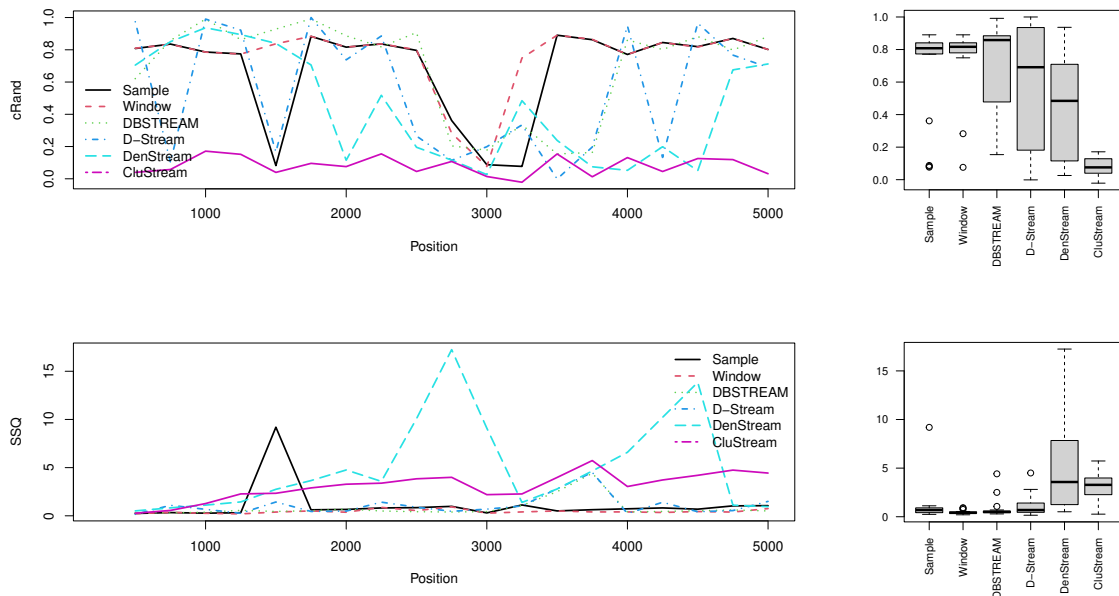


Figure 6: Evaluation of data stream clustering of an evolving stream.

[10,]	0.847	0.541	0.424	0.895	10.075	3.840
[11,]	0.979	0.928	0.394	0.466	17.246	3.991
[12,]	0.301	0.274	0.437	0.692	9.095	2.195
[13,]	1.124	0.402	1.065	1.022	1.387	2.269
[14,]	0.507	0.506	2.505	2.811	2.783	3.979
[15,]	0.633	0.400	4.412	4.501	4.651	5.734
[16,]	0.724	0.405	0.398	0.241	6.589	3.038
[17,]	0.818	0.345	0.455	1.404	10.218	3.719
[18,]	0.687	0.442	0.524	0.422	13.850	4.201
[19,]	1.030	0.388	0.605	0.542	1.104	4.742
[20,]	1.054	0.751	0.503	1.495	0.952	4.436

```
R> matplot(Position, SSQ, type="l", lwd=2)
R> legend("topright", legend=names(evaluation),
+ col=1:6, lty=1:6, bty="n", lwd=2)
```

```
R> boxplot(SSQ, las=2, cex.axis=.8)
```

Figure 6 shows how the different clustering algorithms compare in terms of the corrected Rand index and the sum of squares. For all algorithms the performance degrades around position 3000 since both clusters overlap completely at that point in the stream. The boxplots to the right indicate that D-Stream and DBSTREAM perform overall better than the other algorithms.

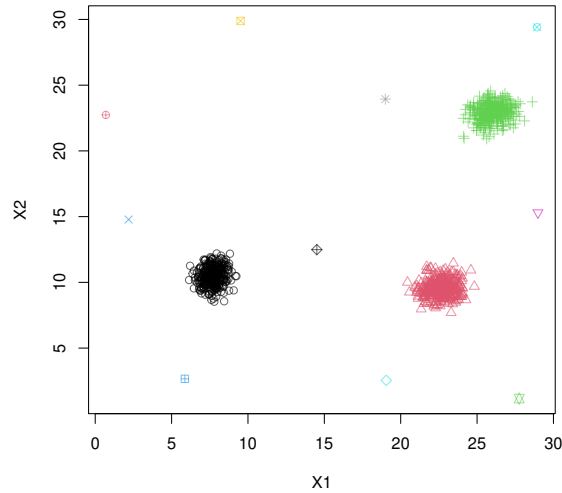


Figure 7: Data points from DSD_Gaussians having 3 clusters and 10 outliers.

4. Outlier detection

To support the outlier detection area, **streamMOA** contains a wrapper to the MOA implementation of the Micro-cluster Continuous Outlier Detector (MCOOD). To demonstrate a synergy of outlier detection capabilities between **stream** and **streamMOA** packages, we bring two basic examples. First, we create a data stream by using an outlier generating data stream, such as

```
R> library(stream)
R> set.seed(1000)
R> stream <- DSD_Gaussians(k = 3, d = 2, outliers = 10,
+                         separation_type = "Mahalanobis", separation = 4,
+                         space_limit = c(0, 30), variance_limit = 0.8,
+                         outlier_options = list(outlier_horizon = 1000))

R> plot(stream, n=1000)
```

The generated stream can be seen in Figure 7.

Then we define a DSC_MCOOD clusterer. Since this is a single-pass clusterer DSC_SinglePass, we do not need to update the model first, we can immediately call evaluation.

```
R> reset_stream(stream)
R> mic_c <- DSOutlier_MCOOD(r = 2.5, w = 1000)
R> evaluate(mic_c, stream, n = 1000, type = "micro",
+          measure = c("crand", "outlierjaccard"))
```

Evaluation results for micro-clusters.

Points were assigned to micro-clusters.

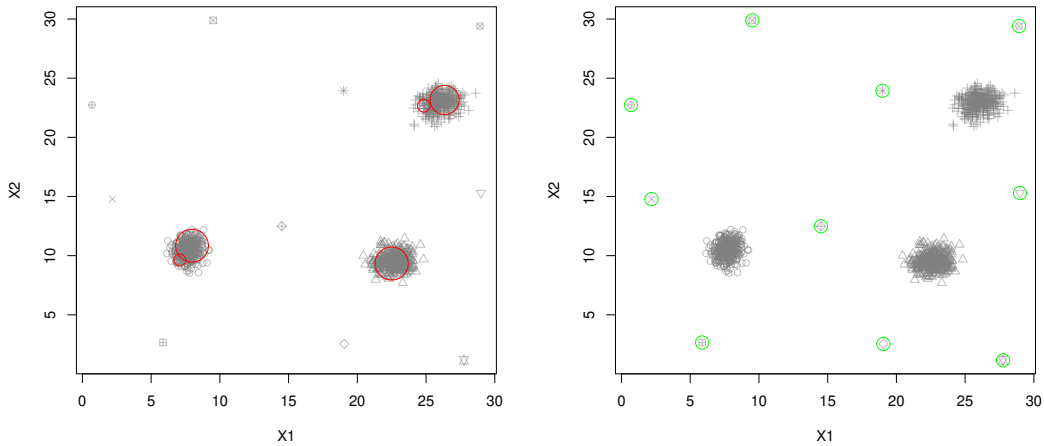


Figure 8: MCODE outlier detection.

```

cRand OutlierJaccard
0.668           1.000

```

```

R> reset_stream(stream)
R> plot(mic_c, stream, n=1000, type="micro")

R> reset_stream(stream)
R> plot(mic_c, stream, n=1000, type="outlier")

```

In Figure 8 we can see micro-cluster and outlier assignments for the generated data stream. Additionally, we can further improve the clustering result by adding the macro-clustering phase as the offline part. This time we will lower the number of neighbor data points, to get more micro-clusters. However, this might affect the outlier detection capability.

```

R> reset_stream(stream)
R> mic_c <- DSOutlier_MCOD(r = 1.5, w = 1000, t = 10)
R> mac_c <- DSC_Kmeans(3)
R> tp_c <- DSC_TwoStage(mic_c, mac_c)
R> evaluate(tp_c, stream, n = 1000, type = "macro",
+          measure = c("crand", "outlierjaccard"))

```

Evaluation results for macro-clusters.
Points were assigned to micro-clusters.

```

cRand OutlierJaccard
0.892           0.185

```

```

R> reset_stream(stream)
R> plot(tp_c, stream, n=1000, type="all")

```

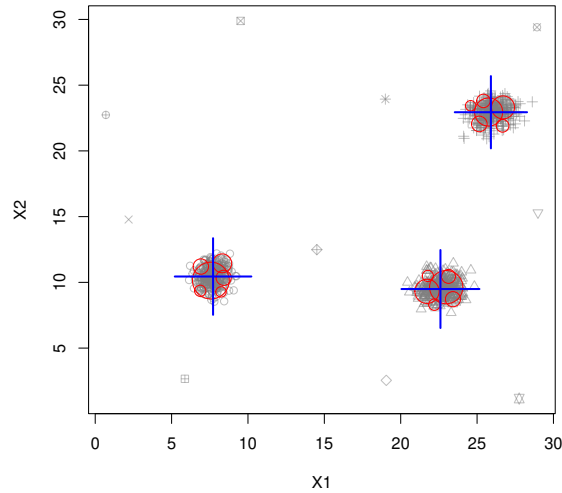


Figure 9: Two stages MCOD + k-means (weighted).

In Figure 9 we can see marked all objects detected by both algorithms in both stages. We can observe more micro-clusters than in Figure 9a, which is due to the lower neighborhood parameter τ . Macro-clusters are correctly identified by the k-means algorithm. We got one false positive outlier, which affected the `OutlierJaccard` index. Overall corrected rand (`cRand`) improved in comparison to using MCOD algorithm only. Such approach can be used to improve the final clustering results.

Acknowledgments

This work is supported in part by the U.S. National Science Foundation as a research experience for undergraduates (REU) under contract number IIS-0948893 and by the National Human Genome Research Institute under contract number R21HG005912.

Affiliation:

Michael Hahsler
 Engineering Management, Information, and Systems
 Lyle School of Engineering
 Southern Methodist University
 P.O. Box 750122
 Dallas, TX 75275-0122
 E-mail: mhahsler@lyle.smu.edu
 URL: <http://michael.hahsler.net>

John Forrest
Microsoft Corporation
E-mail: jforrest@microsoft.com

Matthew Bolaños
Microsoft Corporation
E-mail: mbolanos@curiouscrane.com