streamMOA: Interface to Algorithms from MOA for stream

Matthew Bolaños
Southern Methodist University

John Forrest
Microsoft

Michael Hahsler
Southern Methodist University

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=1500)
R> stream

Memory Stream Interface
Class: DSD_Memory, DSD_R, DSD_data.frame, DSD
With 4 clusters in 2 dimensions
Contains 1500 data points - currently at position 1 - loop is FALSE
```
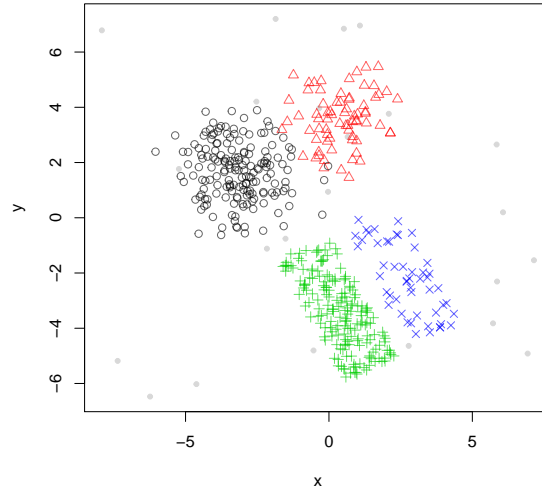
Figure 1: Bar and Gaussians data set.

```
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> tNN <- DSC_tNN(r=.5)
```

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, tNN=tNN,
+   DenStream=denstream, CluStream=clustream)
R> for(a in algorithms) {
+   reset_stream(stream)
+   update(a, stream, 1000)
+ }
```

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

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

   Sample    Window  D-Stream        tNN DenStream CluStream
      100       100        98         88        50       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=a$description, 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 tNN 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. tNN 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=a$description, assignment=TRUE, weight=FALSE, type="micro")
+ }
R> par(op)
```

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
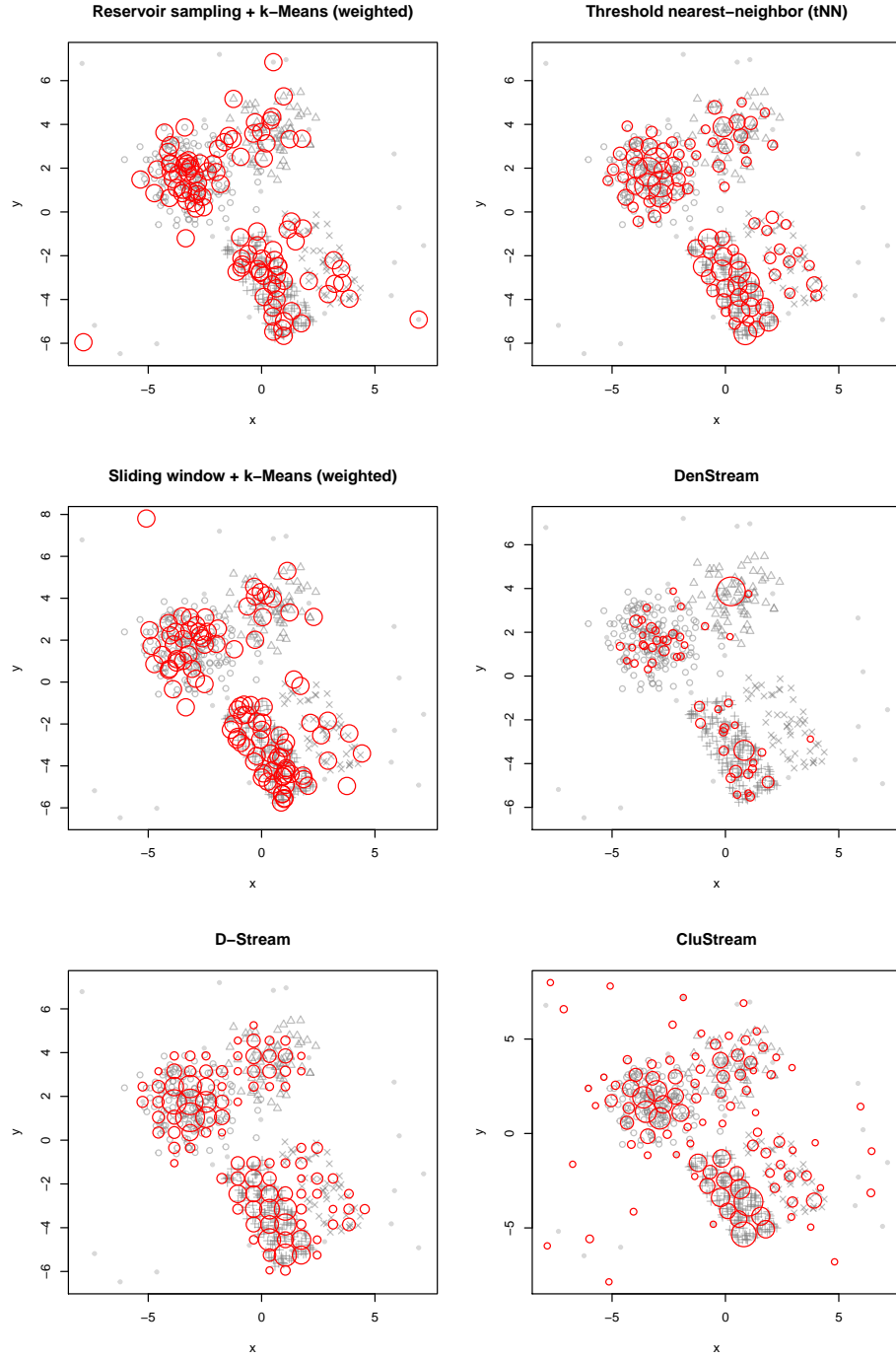
Figure 2: Micro-cluster placement for different data stream clustering algorithms.
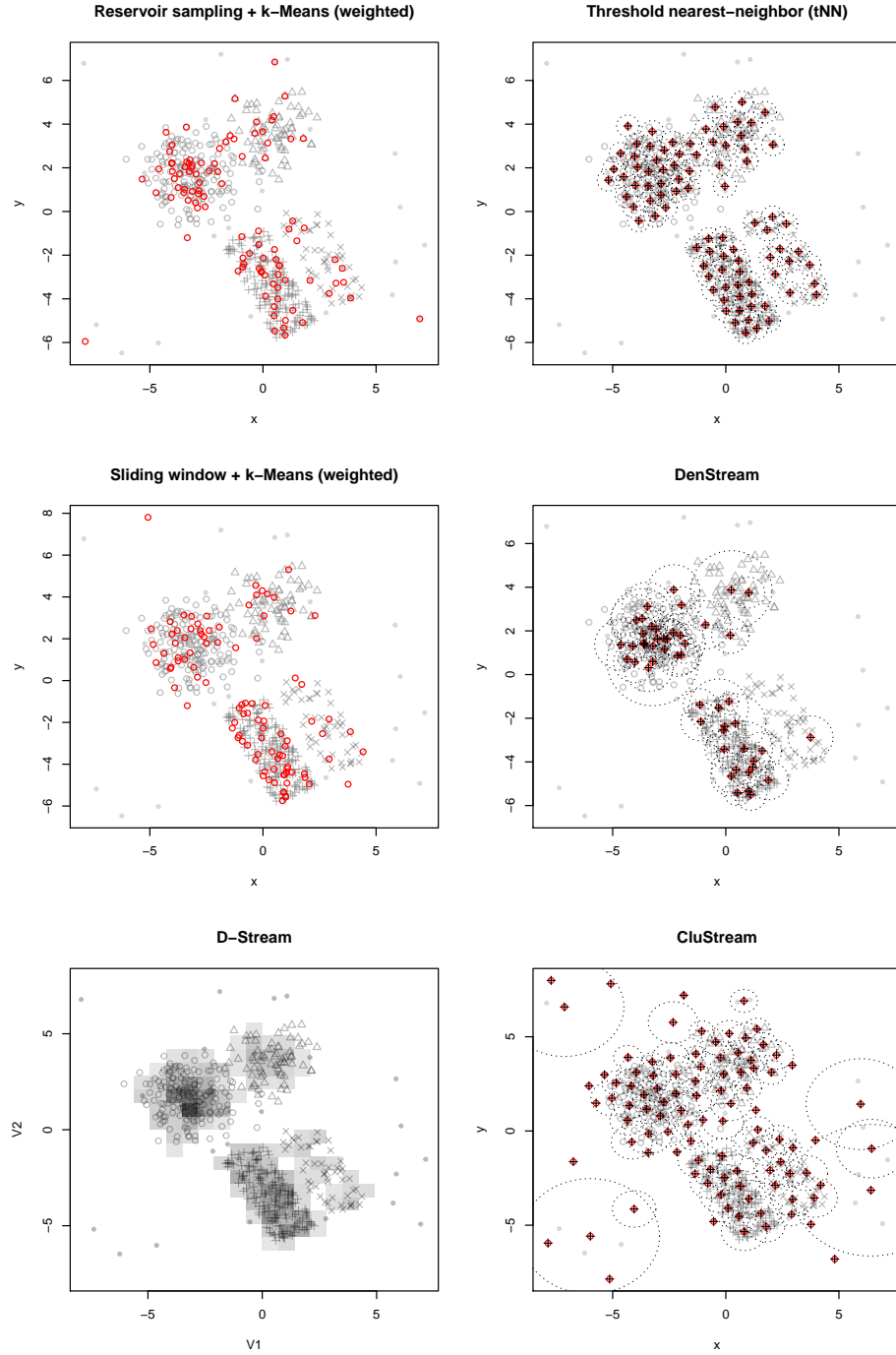
Figure 3: Micro-cluster assignment areas for different data stream clustering algorithms.

grey boxes. tNN 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	tNN	DenStream	CluStream
numMicroClusters	100.000	100.000	98.000	88.000	50.000	100.000
purity	0.956	0.947	0.968	0.966	0.963	0.955
SSQ	216.769	245.412	739.957	676.979	1092.193	514.401
silhouette	0.151	0.147	0.128	0.174	0.111	0.225

We need to be careful with the comparison of these numbers, since the 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 tNN 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, tNN and CluStream have built-in reclustering strategies. D-Stream joins adjacent dense grid cells for form macro-clusters. DenStream and tNN 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=a$description, 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, tNN and D-Stream identify the two denser clusters correctly, but split the lower density clusters into multiple pieces.
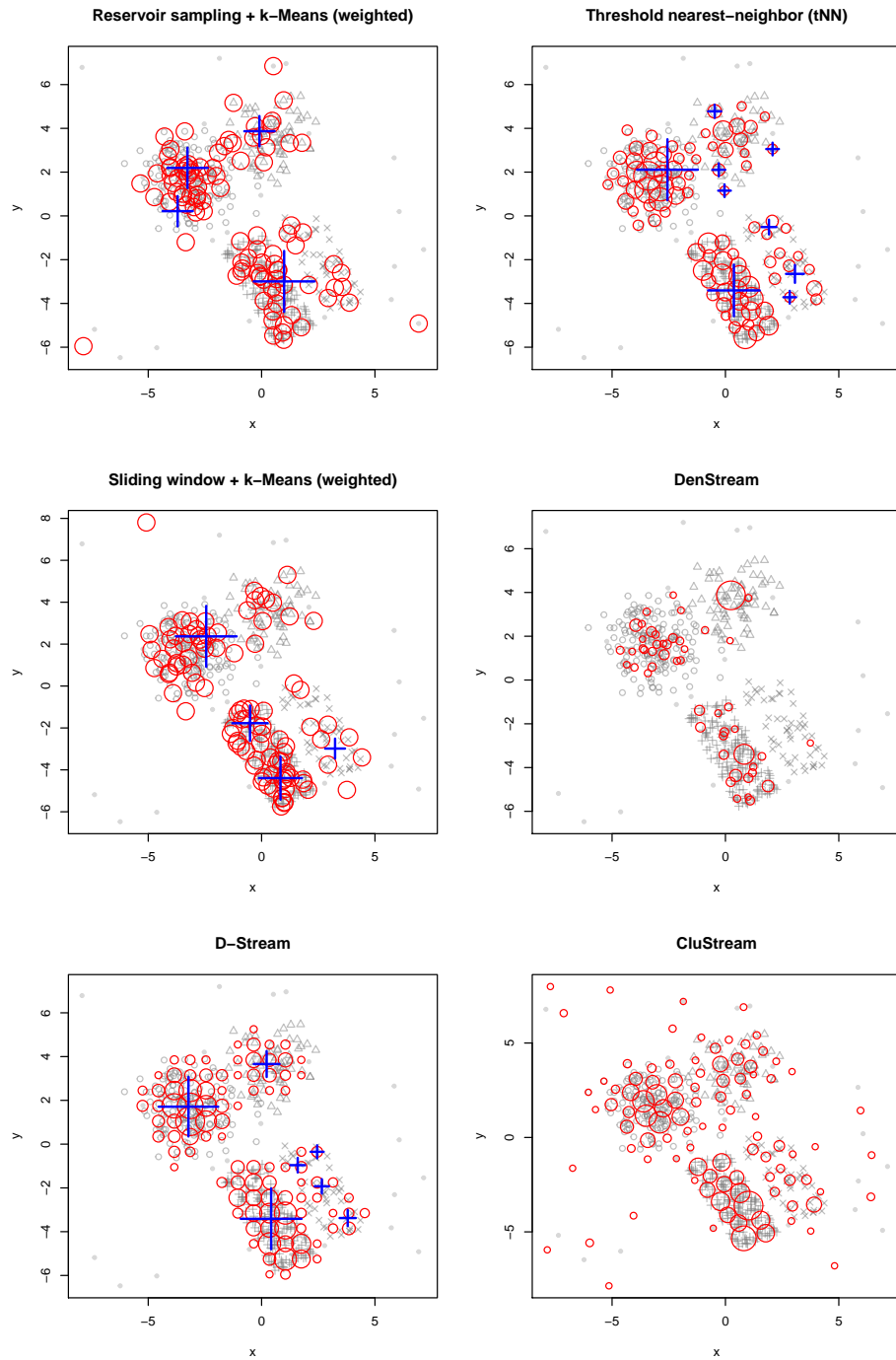
Figure 4: Macro-cluster placement for different data stream clustering algorithms
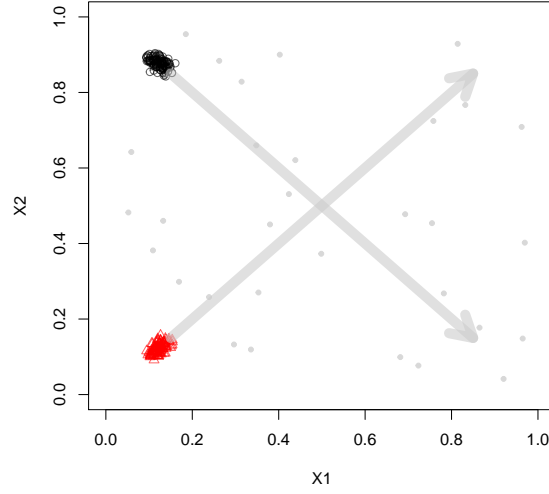
Figure 5: Data points from `DSD_Benchmark(1)` at the beginning of the stream. The two arrows are added to highlight the direction of movement.

```
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	tNN	DenStream	CluStream
numMacroClusters	4.000	4.000	7.000	9.000	NA	NA
purity	0.870	0.833	0.900	0.853	NA	NA
SSQ	1655.925	2094.181	1475.324	2035.116	NA	NA
cRand	0.581	0.550	0.846	0.726	NA	NA

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),
+     macro=DSC_Kmeans(k=2)),
+
+   'D-Stream' = DSC_DStream(gridsize=.05, lambda=.01),
+   tNN = DSC_tNN(r=.02, lambda=.01),
+   DenStream = DSC_DenStream(epsilon=.05, 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> reset_stream(stream)
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	D-Stream	tNN	DenStream	CluStream
[1,]	0.7590	0.7590	0.990	0.977	NA	NA
[2,]	0.8070	0.8098	0.982	0.998	NA	NA
[3,]	0.8280	0.8280	0.976	0.997	NA	NA
[4,]	0.7876	0.7884	0.990	0.972	NA	NA
[5,]	0.7745	0.1639	0.981	0.988	NA	NA
[6,]	0.8371	0.8371	0.982	0.988	NA	NA
[7,]	0.8833	0.8833	1.000	0.992	NA	NA
[8,]	0.8166	0.1178	1.000	0.976	NA	NA
[9,]	0.8365	0.8359	0.991	0.978	NA	NA
[10,]	0.7943	0.0605	0.249	0.936	NA	NA
[11,]	0.0689	0.0384	0.220	0.227	NA	NA

```
[12,] 0.0991 0.0759    0.199 0.209         NA        NA
[13,] 0.7489 0.7507    0.334 0.314         NA        NA
[14,] 0.8903 0.8903    1.000 1.000         NA        NA
[15,] 0.8669 0.8646    0.983 0.988         NA        NA
[16,] 0.7629 0.7629    0.977 0.977         NA        NA
[17,] 0.8448 0.8451    1.000 0.991         NA        NA
[18,] 0.8220 0.8147    0.992 0.996         NA        NA
[19,] 0.8719 0.8688    0.983 0.983         NA        NA
[20,] 0.8019 0.8021    0.985 0.982         NA        NA


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)
```

And then we compare the sum of squares.

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


      Sample Window D-Stream    tNN DenStream CluStream
 [1,]   9.02   8.37   0.0915 0.0954        NA        NA
 [2,]   6.97   7.27   0.1072 0.1004        NA        NA
 [3,]   4.05   4.05  94.8014 0.0988        NA        NA
 [4,]   4.92   5.01   0.1093 0.0992        NA        NA
 [5,]   6.03  15.30  48.2478 0.0988        NA        NA
 [6,]   3.22   3.27  23.4405 0.0973        NA        NA
 [7,]   2.25   2.30   0.0985 0.0927        NA        NA
 [8,]   3.72   6.21   0.0937 0.0866        NA        NA
 [9,]   3.06   3.11   0.1069 0.1061        NA        NA
[10,]   3.91   4.12   0.6858 0.1127        NA        NA
[11,]   3.13   3.08   0.1832 0.1721        NA        NA
[12,]   1.67   1.48   0.2290 0.2109        NA        NA
[13,]   5.28   5.35   0.8783 0.8784        NA        NA
[14,]   2.05   2.05   0.1175 0.1000        NA        NA
[15,]   2.77   2.60  15.5424 0.1051        NA        NA
[16,]   4.51   4.58   0.1024 0.0908        NA        NA
[17,]   3.03   3.02   0.1003 0.0945        NA        NA
[18,]   3.25   3.34   0.1292 0.1036        NA        NA
[19,]   3.55   3.82   0.1128 0.1048        NA        NA
[20,]   6.57   6.63   0.0903 0.0849        NA        NA


R> matplot(Position, SSQ, type="l", lwd=2)
R> legend("topright", legend=names(evaluation),
+   col=1:6, lty=1:6, bty="n", lwd=2)
```
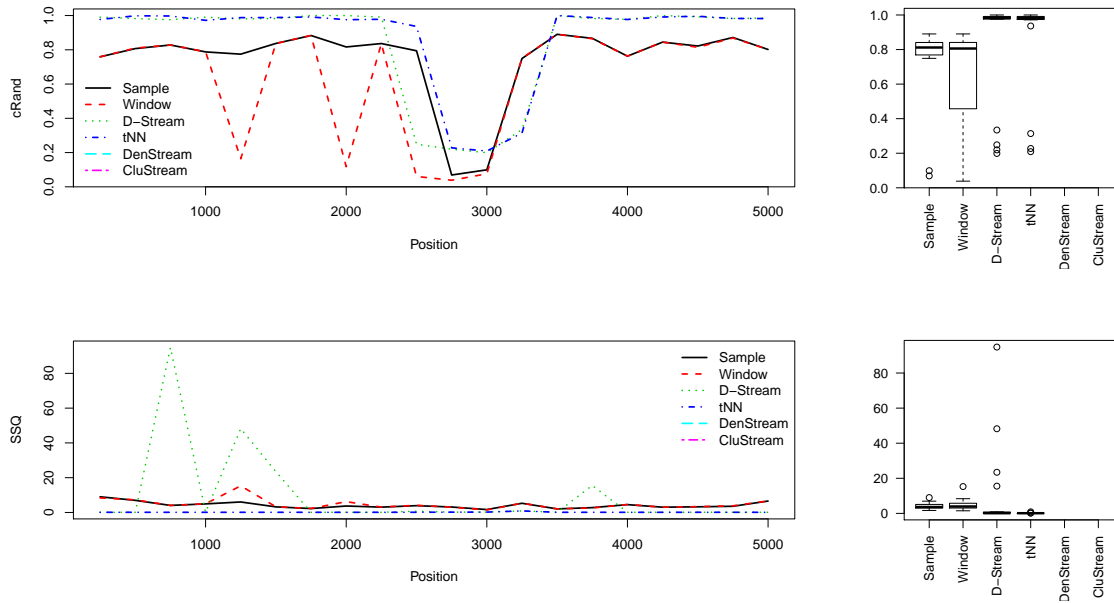
Figure 6: Evaluation of data stream clustering of an evolving stream.

```
R> boxplot(SSQ, las=2)
```

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 box-plots to the right indicate that D-Stream and tNN perform overall better than the other algorithms.

Acknowledgments

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://lyle.smu.edu/~mhahsler

Matthew Bolaños
Computer Science and Engineering
Lyle School of Engineering
Southern Methodist University
E-mail: mbolanos@smu.edu

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