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_Wrapper`. 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> dsd <- DSD_Wrapper(DSD_BarsAndGaussians(noise=0.05), n=1500)
R> dsd

Data Frame Stream Wrapper
With 4 clusters in 2 dimensions
Contains 1500 data points - currently at position 1 - loop is FALSE
```

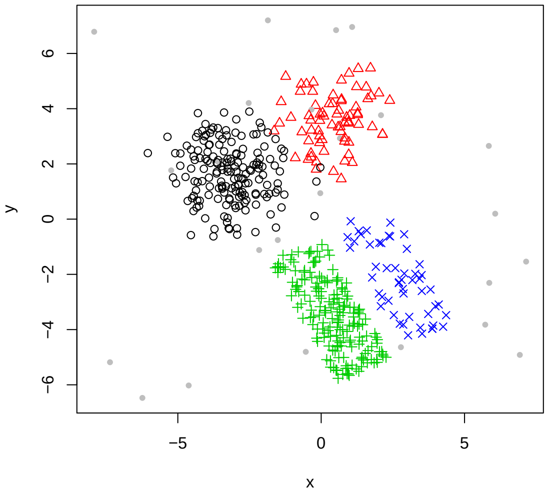


Figure 1: Bar and Gaussians data set.

```
R> plot(dsd)
```

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_Sample(k=100)
R> window <- DSC_Window(horizon=100)
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(dsd)
+   cluster(a, dsd, 1000)
+ }
```

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

```
R> sapply(algorithms, nclusters)

   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.

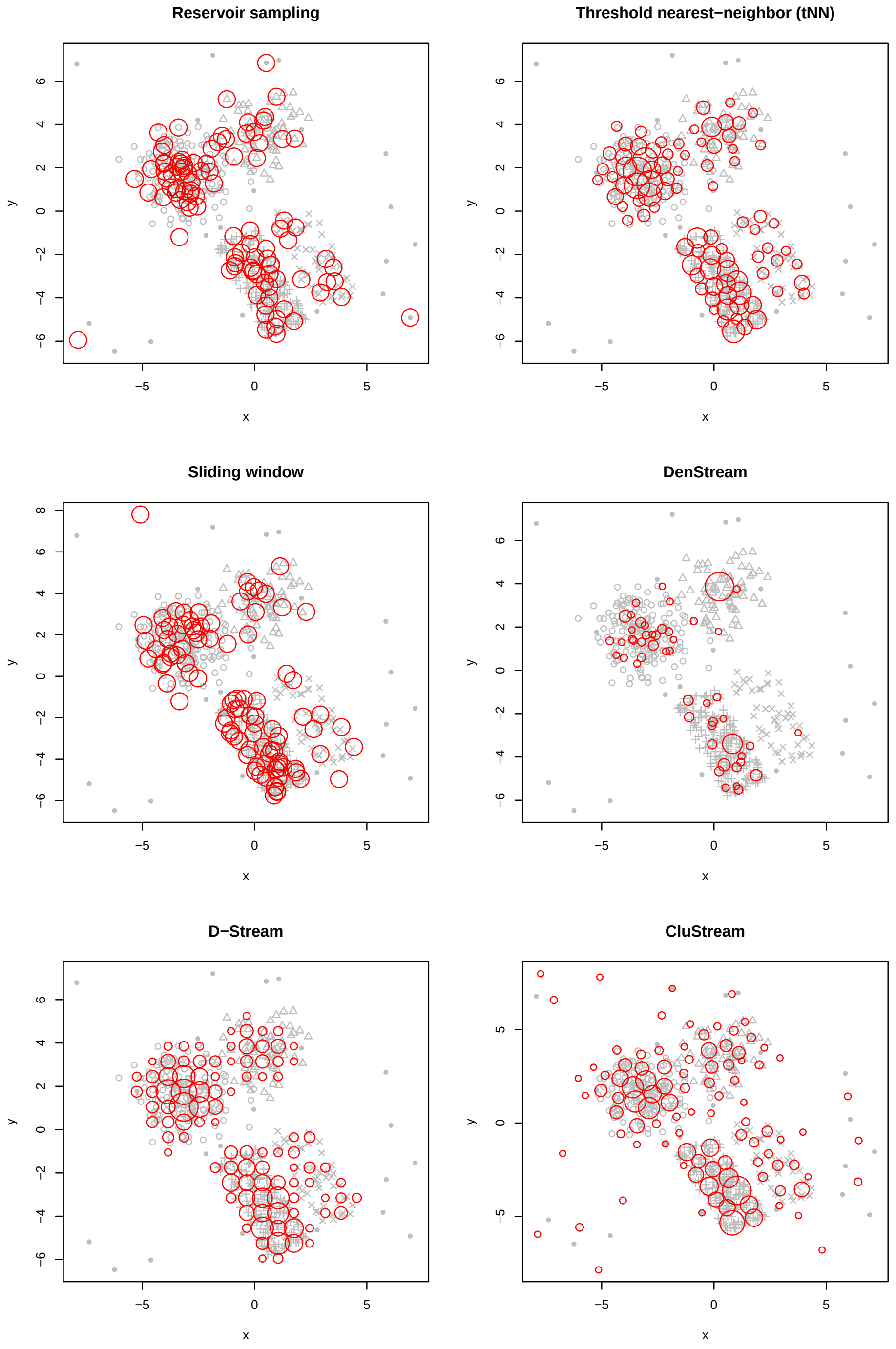
```
R> op <- par(no.readonly = TRUE)
R> layout(mat=matrix(1:6, ncol=2))
R> for(a in algorithms) {
+   reset_stream(dsd)
+   plot(a, dsd, main=a$description)
+ }
R> par(op)
```

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(dsd)
+   plot(a, dsd, main=a$description, assignment=TRUE, weight=FALSE)
+ }
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



Reservoir sampling

Threshold nearest−neighbor (tNN)

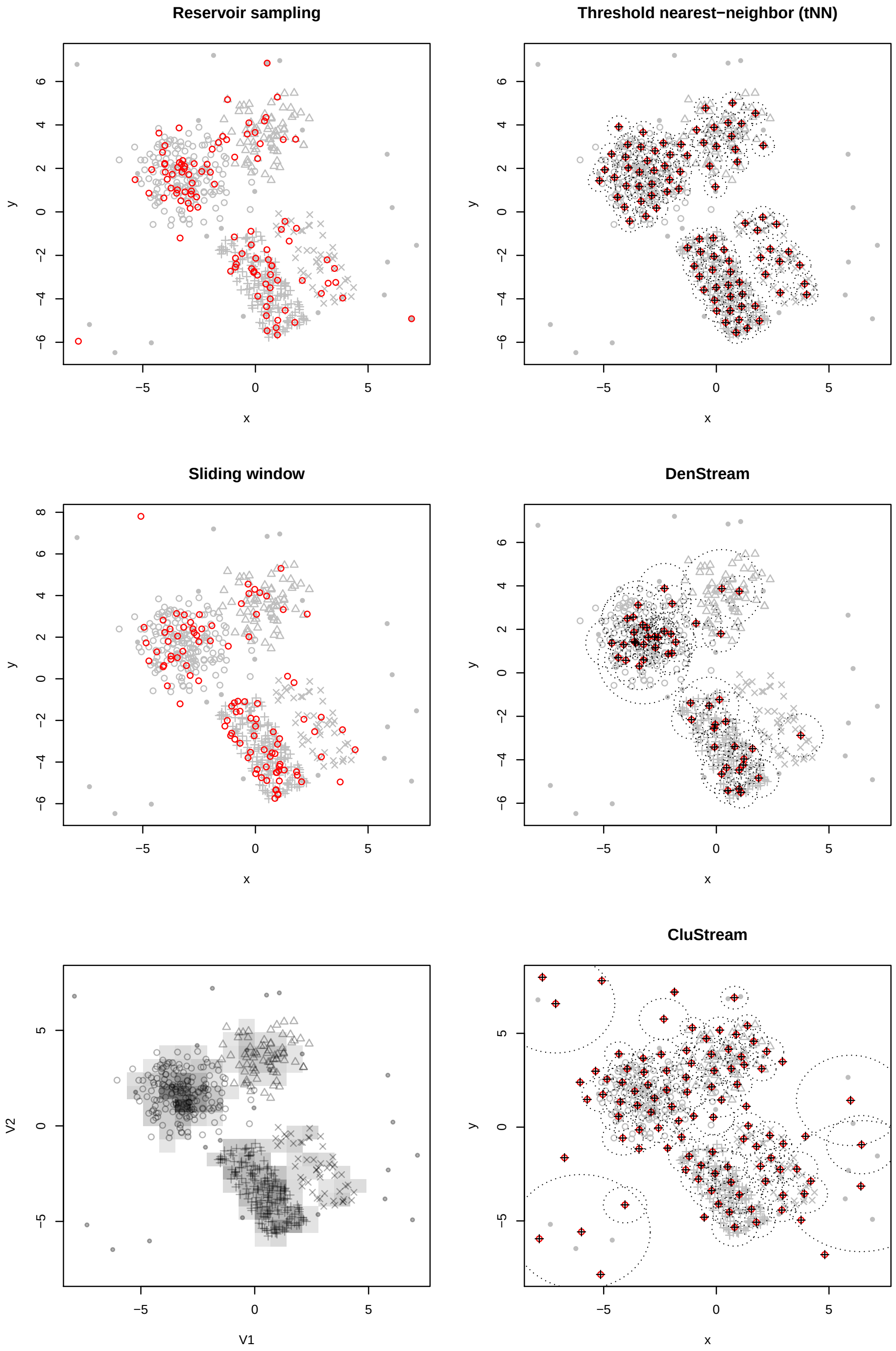
Sliding window

DenStream

D−Stream

CluStream

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


Reservoir sampling

Threshold nearest−neighbor (tNN)

Sliding window

DenStream

CluStream

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(dsd, 1001)
+   evaluate(a, dsd,
+     measure=c("numMicroClusters", "purity", "SSQ", "silhouette"),
+     n=500, assignmentMethod="auto")
+ })

                  Sample  Window D-Stream     tNN DenStream CluStream
numMicroClusters 100.000 100.000   98.000  88.000    50.000   100.000
purity             0.954   0.956    0.900   0.906     0.872     0.930
SSQ              166.254 168.576  148.019 144.392   242.947   162.435
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> sample_km <- DSC_Kmeans(k=4, description="Sample + weighted k-means")
R> recluster(sample_km, algorithms$Sample)
R> algorithms$Sample <- sample_km
R> window_km <- DSC_Kmeans(k=4, description= "Window + weighted k-means")
R> recluster(window_km, algorithms$Window)
R> algorithms$Window <- window_km

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

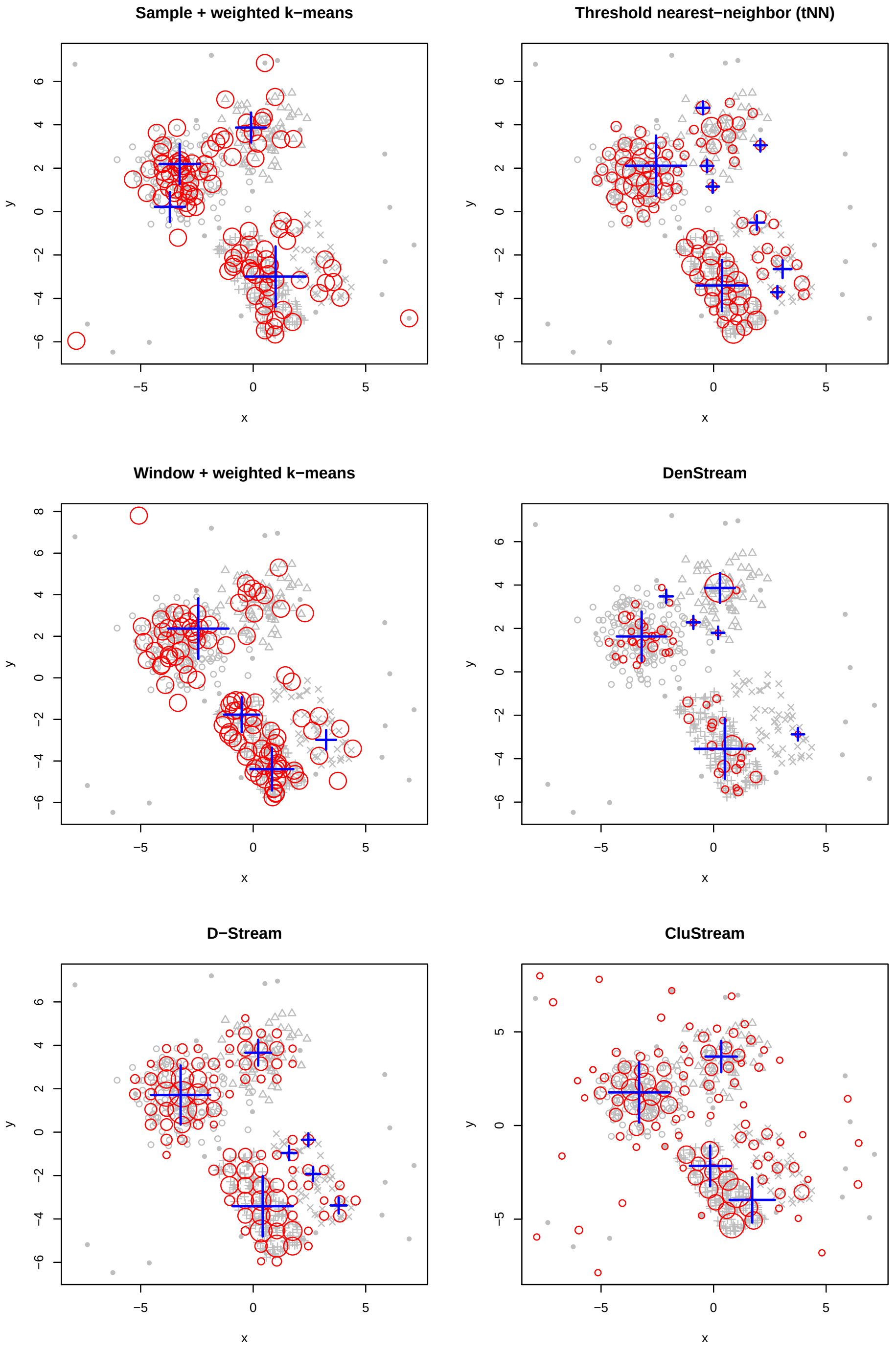


Figure 4: Macro-cluster placement for different data stream clustering algorithms

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.

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

```
                   Sample Window D-Stream     tNN DenStream CluStream
numMacroClusters    4.000   4.00    7.000   9.000     7.000     4.000
purity              0.826   0.79    0.894   0.818     0.870     0.818
SSQ               673.278 726.35  568.623 612.547   588.393   588.336
cRand               0.581   0.55    0.846   0.726     0.773     0.652
```

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> dsd <- DSD_Wrapper(DSD_Benchmark(1), 5000)
```

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

```
R> sample <- DSC_Sample(k=100, biased=TRUE)
R> window <- DSC_Window(horizon=100, lambda=.01)
R> dstream <- DSC_DStream(gridsize=.05, lambda=.01)
R> tNN <- DSC_tNN(r=.02, lambda=.01)
R> denstream <- DSC_DenStream(epsilon=.05, lambda=.01)
R> 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 $k = 2$.

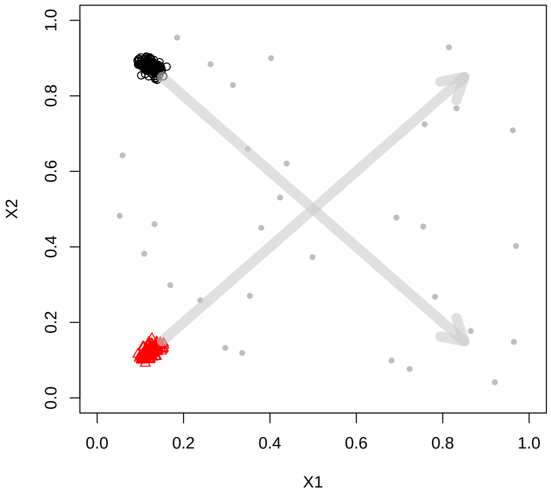


Figure 5: Data points from `DSD_Benchmark(1)` at the beginning of the stream. The two arrows are added to highlight the direction of movement.

```
R> evaluation <- list()
R> n <- 5000
R> horizon <- 250
R> reset_stream(dsd)
R> evaluation[["D-Stream"]] <- evaluate_cluster(dstream, dsd,
+   type="macro", assign="micro",
+   measure=c("numMicro","numMacro","SSQ", "crand"),
+   n=n, horizon=horizon)
R> reset_stream(dsd)
R> evaluation[["tNN"]] <- evaluate_cluster(tNN, dsd,
+   type="macro", assign="micro",
+   measure=c("numMicro","numMacro","SSQ","crand"),
+   n=n, horizon=horizon)
R> reset_stream(dsd)
R> evaluation[["DenStream"]] <- evaluate_cluster(denstream, dsd,
+   type="macro", assign="micro",
+   measure=c("numMicro","numMacro","SSQ", "crand"),
+   n=n, horizon=horizon)
R> reset_stream(dsd)
R> evaluation[["CluStream"]] <- evaluate_cluster(clustream, dsd,
+   type="macro", assign="micro",
+   measure=c("numMicro","numMacro","SSQ", "crand"),
+   n=n, horizon=horizon)
R> reset_stream(dsd)
R> evaluation[["Sample"]] <- evaluate_cluster(sample, dsd,
+   macro=DSC_Kmeans(k=2),
+   type="macro", assign="micro",
+   measure=c("numMicro","numMacro","SSQ", "crand"),
+   n=n, horizon=horizon)
R> reset_stream(dsd)
R> evaluation[["Window"]] <- evaluate_cluster(window, dsd,
+   macro=DSC_Kmeans(k=2),
+   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

      D-Stream   tNN DenStream CluStream Sample Window
 [1,]    0.990 0.977     0.928    0.7152 0.7600 0.7590
 [2,]    0.982 0.998     0.937    0.7930 0.8078 0.1491
 [3,]    0.976 0.997     0.930    0.8226 0.8355 0.8280
 [4,]    0.990 0.972     0.828    0.7326 0.7884 0.7884
 [5,]    0.981 0.988     0.943    0.7125 0.7718 0.7757
 [6,]    0.982 0.988     0.878    0.7672 0.8371 0.8371
 [7,]    1.000 0.992     0.892    0.8617 0.8846 0.8833
 [8,]    1.000 0.976     0.848    0.7910 0.8164 0.8164
 [9,]    0.991 0.978     0.955    0.7892 0.8359 0.8359
[10,]    0.249 0.936     0.267    0.7806 0.0605 0.7943
[11,]    0.220 0.227     0.208    0.1193 0.0795 0.0285
[12,]    0.199 0.209     0.192    0.0757 0.0759 0.0759
[13,]    0.334 0.314     0.253    0.0873 0.7507 0.0769
[14,]    1.000 1.000     0.137    0.0644 0.8903 0.8903
[15,]    0.983 0.988     0.180    0.8154 0.8646 0.8646
[16,]    0.977 0.977     0.868    0.6783 0.1364 0.7629
[17,]    1.000 0.991     0.921    0.7977 0.8448 0.8451
[18,]    0.992 0.996     0.855    0.7308 0.8194 0.8147
[19,]    0.983 0.983     0.875    0.7178 0.8675 0.8688
[20,]    0.985 0.982     0.945    0.7691 0.1444 0.8021
```

```
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
```

```
      D-Stream   tNN DenStream CluStream Sample Window
 [1,]     4.00  4.09      4.03     11.39  11.19  10.67
 [2,]     4.23  4.22      4.98      7.46   8.78  76.57
 [3,]     4.31  4.23      7.46      7.33   8.51   7.99
 [4,]     4.43  4.24     10.39     11.56  11.10  11.28
 [5,]     4.19  4.15     13.18     11.68   9.63  10.96
 [6,]     4.16  4.17      8.65     14.50   4.95   7.17
 [7,]     4.25  4.19      4.38     15.76   4.34   6.30
 [8,]     4.05  3.94      4.16     16.17   9.03  10.70
 [9,]     4.25  4.25      4.98     18.13   5.76   7.33
[10,]    11.29  4.07     11.73     19.31  12.05   5.47
[11,]     5.35  5.35      6.12     16.54   6.64   5.41
[12,]     6.42  6.43      6.99     12.39   6.71   7.20
[13,]    13.41 13.39     14.84     16.64   5.82  13.12
[14,]     4.62  4.35     22.70     25.14   5.73   6.36
[15,]     4.58  4.31     30.16     17.68   6.04   6.99
[16,]     3.82  3.82      3.96     15.76  36.32   6.60
[17,]     4.24  4.19      4.46     19.58   4.84   5.23
[18,]     4.41  4.36      4.51     18.98   4.72   5.42
[19,]     4.48  4.28      4.41     21.53   6.21   5.07
[20,]     3.86  3.87      4.05     21.85  70.50   8.06
```

```
R> matplot(Position, SSQ, type="l", lwd=2)
```

```
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

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.

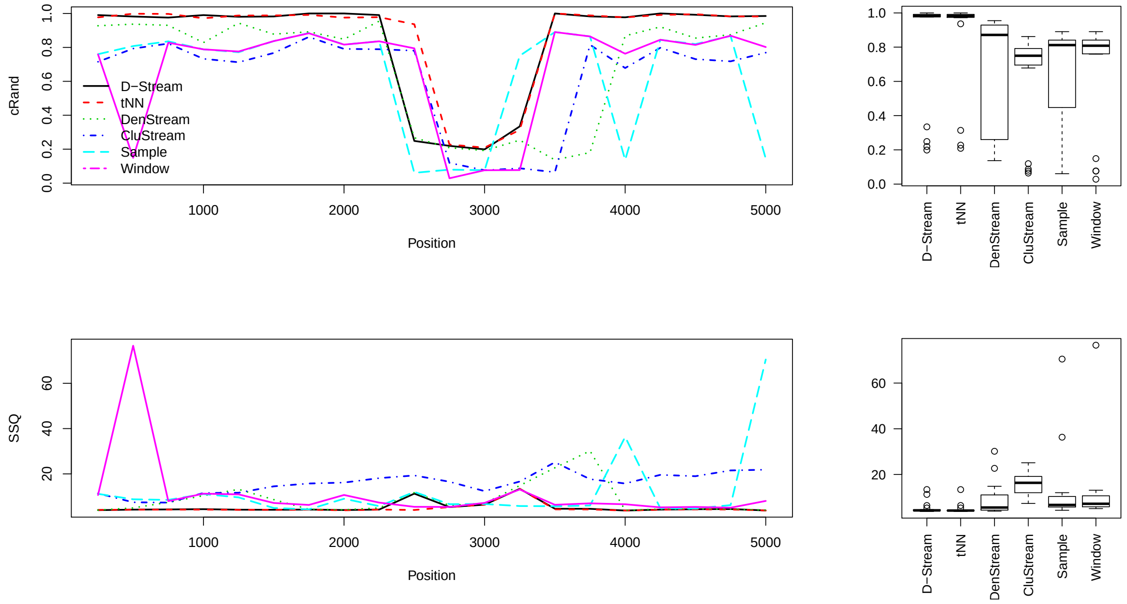


Figure 6: Evaluation of data stream clustering of an evolving stream.

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