Extending the stream Framework

Michael Hahsler
Southern Methodist University

Matthew Bolaños
Microsoft Corporation

John Forrest
Microsoft Corporation

Abstract

This document describes how to add new data stream sources `DSD` and data stream tasks `DST` to the **stream** framework.

Keywords: data streams, data mining, clustering.

1. Extending the stream framework

Since stream mining is a relatively young field and many advances are expected in the near future, the object oriented framework in **stream** is developed with easy extensibility in mind. Implementations for data streams (DSD) and data stream mining tasks (DST) can be easily added by implementing a small number of core functions. The actual implementation can be written in either R, Java, C/C++ or any other programming language which can be interfaced by R. In the following we discuss how to extend **stream** with new DSD and DST implementations.

1.1. Adding a new data stream source (DSD)

DSD objects can be a management layer on top of a real data stream, a wrapper for data stored in memory or on disk, or a generator which simulates a data stream with know properties for controlled experiments. Figure 1 shows the relationship (inheritance hierarchy) of the DSD classes as a UML class diagram (Fowler 2003). All DSD classes extend the abstract base class `DSD`. There are currently two types of DSD implementations, classes which implement R-based data streams (`DSD_R`) and MOA-based stream generators (`DSD_MOA`) provided in **streamMOA**. Note that abstract classes define interfaces and only implement common functionality. Only implementation classes can be used to create objects (instances). This mechanism is not enforced by S3, but is implemented in **stream** by providing for all abstract classes constructor functions which create an error.

The class hierarchy in Figure 1 is implemented using the S3 class system (Chambers and Hastie 1992). Class membership and the inheritance hierarchy is represented by a vector of class names stored as the object's class attribute. For example, an object of class `DSD_Gaussians` will have the class attribute vector `c("DSD_Gaussians", "DSD_R", "DSD")` indicating that the object is an R implementation of DSD. This allows the framework to implement all

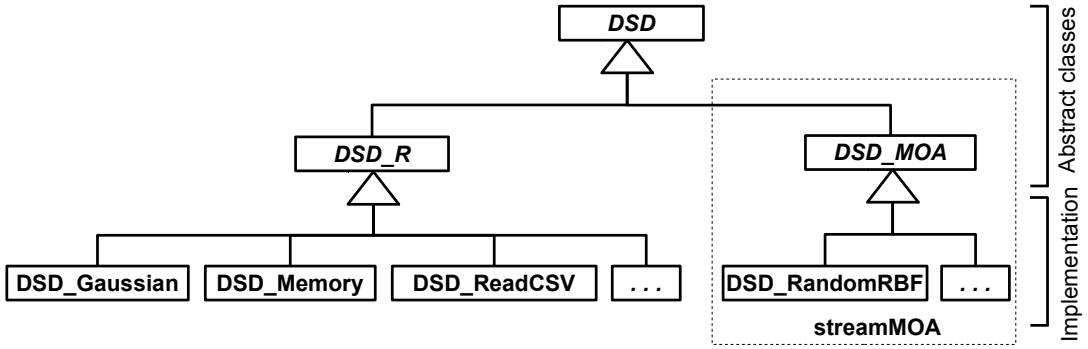

Figure 1: Overview of the data stream data (DSD) class structure.

common functionality as functions at the level of `DSD` and `DSD_R` and only a minimal set of functions is required to implement a new data stream source. Note that the class attribute has to contain a vector of all parent classes in the class diagram in bottom-up order.

For a new DSD implementation only the following two functions need to be implemented:

1. A creator function (with a name starting with the prefix `DSD_`) and
2. the `get_points()` method.

The creator function creates an object of the appropriate `DSD` subclass. Typically this S3 object contains a list of all parameters, an open R connection and/or an environment or a reference class for storing state information (e.g., the current position in the stream). Standard parameters are `d` and `k` for the number of dimensions of the created data and the true number of clusters, respectively. In addition an element called `"description"` should be provided. This element is used by `print()`.

The implemented `get_points()` needs to dispatch for the class and create as the output a data frame containing the new data points as rows. If called with `info = TRUE` additional information columns starting with `.` should be returned. For example, a column called `.class` with the ground truth (true cluster assignment as an integer vector; noise is represented by `NA`) should be returned for data streams for clustering or classification. Other information columns include `.id` for point IDs and `.time` for time stamps.

For a very simple example, we show here the implementation of `DSD_UniformNoise` available in the package's source code in file `DSD_UniformNoise.R`. This generator creates noise points uniformly distributed in a d-dimensional hypercube with a given range.

```
R> library("stream")
```

```
R> DSD_UniformNoise <- function(d = 2, range = NULL) {
+   if(is.null(range)) range <- matrix(c(0, 1), ncol = 2, nrow = d,
+     byrow = TRUE)
+   structure(list(description = "Uniform Noise Data Stream", d = d,
+     k = NA_integer_, range = range),
+         class = c("DSD_UniformNoise", "DSD_R", "DSD"))
```

```
+   }
R> get_points.DSD_UniformNoise <- function(x, n = 1,
+   info = TRUE, ...) {
+     data <- data.frame(t(replicate(n, runif(
+       x$d, min = x$range[, 1], max = x$range[, 2]))))
+
+      if (info) data[[".class"]] <- NA
+
+     data
+ }
```

The constructor only stores the description, the dimensionality and the range of the data. For this data generator `k`, the number of true clusters, is not applicable. Since all data is random, there is also no need to store a state. The `get_points()` implementation creates n random points and if class assignment info is requested, then a `.class` column is added containing all `NA`s indicating that the data points are all noise.

Now the new stream type can already be used.

```
R> stream <- DSD_UniformNoise()
R> stream

Uniform Noise Data Stream
Class: DSD_UniformNoise, DSD_R, DSD

R> plot(stream, main = description(stream))
```

The resulting plot is shown in Figure 2.

1.2. Adding a new data stream tasks (DST)

DST refers to any data mining task that can be applied to data streams. The design is flexible enough for future extensions including even currently unknown tasks. Figure 3 shows the class hierarchy for DST.

DST classes implement mutable objects which can be changed without creating a copy. This is more efficient, since otherwise a new copy of all data structures used by the algorithm would be created for processing each data point. Mutable objects can be implemented in R using environments or the recently introduced reference class construct (see package **methods** by the R Core Team (2014)). Alternatively, pointers to external data structures in Java or C/C++ can be used to create mutable objects.

To add a new data stream mining tasks (e.g., frequent pattern mining), a new package with a subclass hierarchy similar to the hierarchy in Figure 3 for data stream clustering (DSC) can be easily added. This new package can take full advantage of the already existing infrastructure in **stream**. An example is the package **streamMOA** Hahsler and Bolanos (2015), which can be used as a model to develop a new package. We plan to provide more add-on packages to **stream** for frequent pattern mining and data stream classification in the near future.

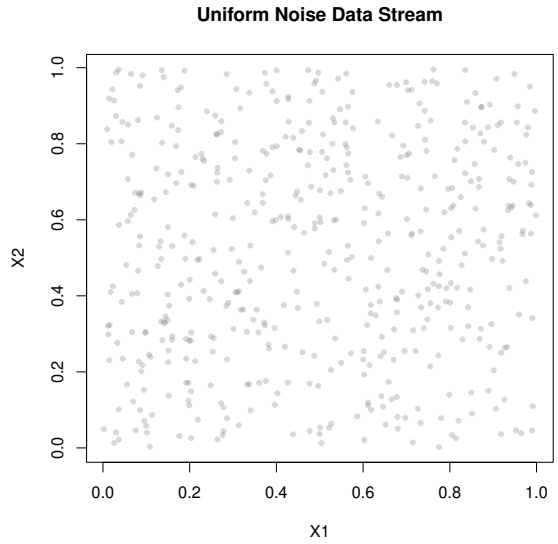

Figure 2: Sample points from the newly implemented `DSD_UniformNoise` object.

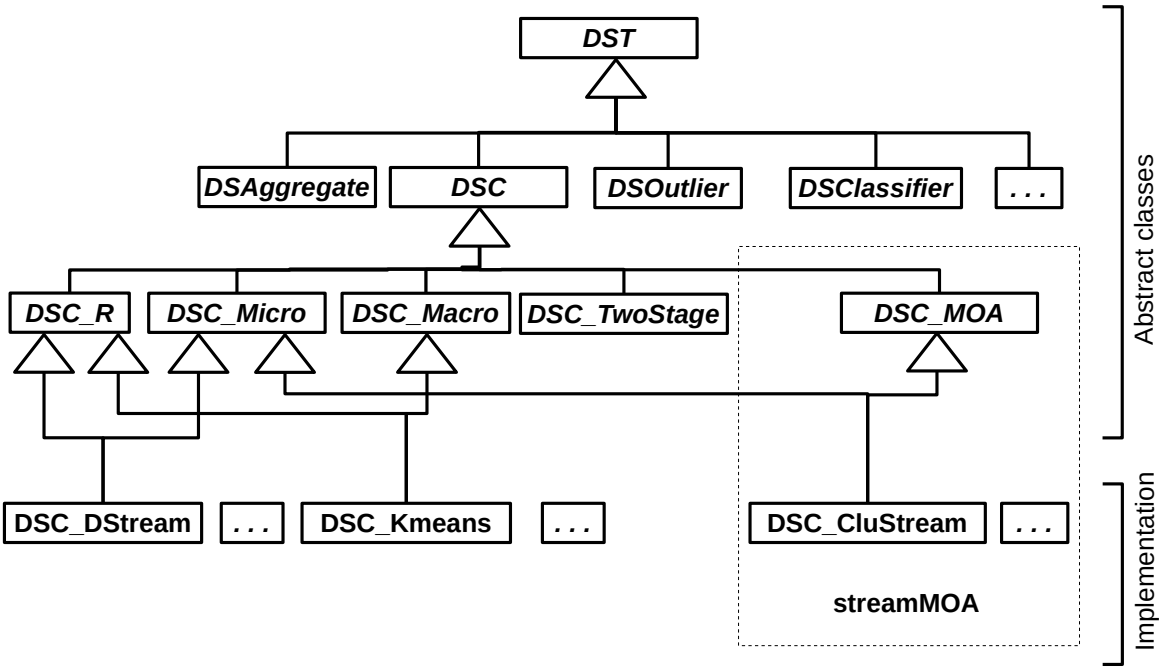

Figure 3: Overview of the data stream task (DST) class structure with subclasses for clustering (DSC), classification (DSClassify) and frequent pattern mining (DSFP) and outlier detection (DSOutlier).

In the following we discuss how to interface an existing algorithm with **stream**. We concentrate again on clustering, but interfacing algorithms for other types of tasks is similar. To interface an existing clustering algorithm with **stream**,

1. a creator function (typically named after the algorithm and starting with `DSC_`) which created the clustering object,
2. an implementation of the actual cluster algorithm, and
3. accessors for the clustering

are needed. The implementation depends on the interface that is used. Currently an `R` interface is available as `DSC_R` and a MOA interface is implemented in `DSC_MOA` (in **streamMOA**). The implementation for `DSC_MOA` takes care of all MOA-based clustering algorithms and we will concentrate here on the R interface.

For the R interface, the clustering class needs to contain the elements `"description"` and `"RObj"`. The description needs to contain a character string describing the algorithm. RObj is expected to be a reference class object and contain the following methods:

1. `cluster(newdata, ...)`, where `newdata` is a data frame with new data points.
2. `get_assignment(dsc, points, ...)`, where the clusterer `dsc` returns cluster assignments for the input `points` data frame.
3. For micro-clusters: `get_microclusters(...)` and `get_microweights(...)`
4. For macro-clusters: `get_macroclusters(...)`, `get_macroweights` and `microToMacro(micro, ...)` which does micro- to macro-cluster matching.

Note that these are methods for reference classes and do not contain the called object in the parameter list. Neither of these methods are called directly by the user. Figure 4 shows that the function `update()` is used to cluster data points, and `get_centers()` and `get_weights()` are used to obtain the clustering. These user facing functions call internally the methods in RObj via the R interface in class `DSC_R`.

For a comprehensive example of a clustering algorithm implemented in R, we refer the reader to `DSC_DStream` (in file `DSC_DStream.R`) in the package's `R` directory.

References

Chambers JM, Hastie TJ (1992). Statistical Models in S. Chapman & Hall. ISBN 9780412830402.

Fowler M (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3 edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0321193687.

Hahsler M, Bolanos M (2015). streamMOA: Interface for MOA Stream Clustering Algorithms. R package version 1.1-2, URL http://CRAN.R-project.org/package=streamMOA.

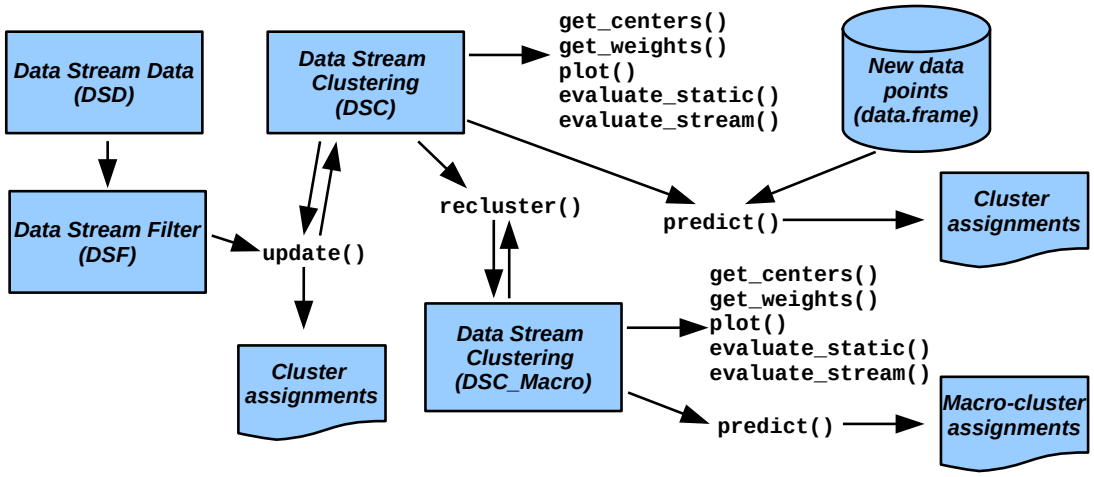

Figure 4: Interaction between the DSD and DSC classes.

R Core Team (2014). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Affiliation:

Michael Hahsler
Computer Science
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
Research Now
5800 Tennyson Pkwy # 600
Plano, TX 75024 E-mail: mbolanos@curiouscrane.com

John Forrest
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-7329
E-mail: jforrest@microsoft.com