MOVING GENERATOR: INFRASTRUCTURE FOR SPECIFYING AND SIMULATING EVOLVING STREAMS IN R

Approved by:

Michael Hahsler, Ph. D.

Margaret H. Dunham, Ph. D.

Prof. Mark Fontenot

MOVING GENERATOR:

INFRASTRUCTURE FOR SPECIFYING AND SIMULATING EVOLVING STREAMS IN R

A Thesis Presented to the Faculty of the

Bobby B. Lyle School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Bachelors of Science

with a

Major in Computer Science

by

Matthew Bolanos

May 17, 2014

Bolanos, Matthew

Moving Generator:

Infrastructure for Specifying and Simulating Evolving Streams in R

Advisor: Professor Michael Hahsler Bachelors of Science degree conferred May 17, 2014 Thesis completed April 18, 2014

In recent years, data streams have become an important area of research. Common data mining tasks involve classifying and clustering changing and evolving data over an extended period. Due to the complexity of changing data, however, it is difficult to produce real or simulated datasets that clearly exhibit behaviors such as merging and splitting of clusters. In this paper I introduce the implementation of **streamMovingGenerator** in package stream that provides a means to generate several different types of evolving data models through an intuitive interface. I have built the framework in R, a popular tool for data mining and statistical analysis, as a part of the **stream** framework [6]. It allows researchers to quickly create different scenarios that would otherwise be difficult to reproduce in the real-world without having precise control over a specific environment. With the provided functionality, researchers are able to easily and quickly test different characteristics of data stream algorithms while getting robust evaluations back. At the end of this paper, I will provide in-depth examples of such testing.

TABLE OF CONTENTS

LIST OF FIGURES	vi			
CHAPTER				
1. Introduction	1			
2. Background	4			
2.1. Data Stream Cluster Analysis	4			
2.2. Evaluation Measures	4			
2.3. MOA	6			
3. streamMovingGenerator	9			
3.1. Moving Generator Cluster	11			
3.2. Data Stream Data Moving Generator	14			
4. Analysis with Data Moving Generator	17			
4.1. Splitting Clusters	17			
4.1.1. Initializing Data Stream	17			
4.1.2. Clustering and Evaluation	19			
4.2. Merging Clusters	22			
4.2.1. Initializing Data Stream	23			
4.2.2. Clustering and Evaluation	23			
4.3. Passing Clusters				
4.3.1. Initializing Data Stream	25			
4.3.2. Clustering and Evaluation	26			
5. Conclusion and Future Work	29			
APPENDIX				

A. stream Contributions	30
REFERENCES	31

LIST OF FIGURES

Figure	F	' age
2.1.	Screenshot of the MOA Stream Clustering GUI	7
3.1.	UML Diagram of the Relationship between Moving Generator and Moving Generator Clusters	9
3.2.	UML Diagram of the DSD Inheritance Hierarchy	10
3.3.	UML Diagram of Moving Generator Cluster Inheritance	11
4.1.	Splitting of a Cluster	17
4.2.	Plot Generated by animate_data()	19
4.3.	F1 Score from tNN using the Splitting Data Stream	21
4.4.	F1 Score for Splitting Data Stream	22
4.5.	Merging of Clusters	22
4.6.	F1 Score Over Time for tNN and DenStream for Merging Data Stream .	24
4.7.	F1 Score Over Time for CluStream and ClusTree for Merging Data Stream	25
4.8.	Passing of Clusters	25
4.9.	Adjusted Rand Index for Passing Data Stream	28

Chapter 1

Introduction

The use of datasets to determine particular characteristics of clustering algorithms is a common practice within the field of data mining. Datasets, whether they are real or artificially generated, are useful in serving as benchmarks so that different algorithms can be objectively compared. Examples of popular real datasets used within the data mining community include the KDD Cup datasets and the datasets found within UCI's Machine Learning Repository [4].

It is difficult, however, to find real datasets that have specialized behaviors to test specific features of a clustering algorithm without having precise control of the environment that is producing them. For this reason, static artificial datasets have been created to simulate different scenarios. R packages such as **mlbench** [14] include numerous artificial data generators that can alter several attributes and behaviors of a dataset, including the number of points generated, the density of the data, the proximity of clusters and the shapes of clusters. These static artificial datasets are particularly useful in evaluating traditional clustering algorithms such as k-means and hierarchical clustering because they exhibit characteristics typical of the datasets are small in size and static in nature.

These two aspects, however, are detrimental to researchers who desire to test and/or evaluate algorithms designed to cluster data streams [1]. This is because data streams are commonly infinite in size and dynamic in nature. Without the ability to generate nearly limitless, evolving data, it becomes effectively impossible to verify the robustness of a data stream clustering algorithm.

Unfortunately, there has been very little effort in developing tools to create artificial data to evaluate data stream clustering algorithms. As a result, it has been incredibly challenging to create useful comparisons between the many algorithms proposed over the past several years or to provide any means of examining data stream clustering behaviors under controlled situations.

The most notable exception has been the development of the MOA framework [5], which provides a means to generate several different data streams including hyperplanes, waveforms and random data. While these generators are useful compared to the dearth of other data stream generators, they still do not provide the level of granularity a user would need to define specific behaviors, such as merging and splitting of clusters.

In addition to MOA, there is also a plethora of publicly accessible datasets that represent naturally occurring data streams, such as Twitter's stream of tweets, network packet streams and stock market activity. It goes without saying, however, that these data streams are incredibly complex in nature. And as mentioned before, any attempt to control the behavior of any real world data stream to generate a desirable and testable behavior would be too difficult or too cost prohibitive, making them poor candidates for benchmarks. Moreover, to serve as a benchmark, a dataset must also include its ground truth so that results can be evaluated. This is nearly impossible for data streams because it would be too time consuming to create.

Because of these reasons, I developed the **streamMovingGenerator** extension in the **stream** package, which will serve as the focus of the remainder of the paper.

The **streamMovingGenerator** extension is unique in the way that it is constructed. Instead of acting as a simple model with different attributes, the generator acts as a driver, or a conductor, of several different individual models that can represent each of the clusters in the data stream. Each of these models are then constructed in such a way that allows for their behavior to be altered over time. By giving such granular control of not only the overall stream but also over the individual clusters, users of the **streamMovingGenerator** package are afforded the ability to orchestrate extremely complex situations that would be impossible to create otherwise.

The paper is organized as follows. I will first begin by discussing the technical details on how the community currently evaluates clustering algorithms in Chapter 2 and then move on towards discussion of the design of the **streamMovingGenera-tor** extension in Chapter 3. Afterwords, I will discuss how to use the **streamMov-ingGenerator** by providing an analysis of various clustering algorithms with data simulated with **streamMovingGenerator**.

Chapter 2

Background

This section defines clustering, provides an overview of the evaluation process and summarizes the data generation methods in the field today.

2.1. Data Stream Cluster Analysis

Clustering within the field of data mining is the process by which one attempts to group or cluster similar objects together. Cluster analysis is useful in a range of fields [19]. In biology, for example, it is helpful in finding genetically similar organisms, and on the World Wide Web, clustering can be leveraged to find different communities within social networks. Today, the most popular methods include k-means, hierarchical clustering and density based models like DBSCAN.

The classic approach to clustering makes the assumption that datasets are static. Unfortunately, this assumption does not hold for datasets that arrive continuously, like internet usage, financial transactions, weather data and the data produced by this package. These datasets are called data streams and are becoming more and more common in today's data driven world. To solve for this problem, data stream clustering algorithms were introduced that are capable of clustering a dataset using a single pass and with limited memory. Current data stream clustering algorithms include CluStream [2], ClusTree [13] and DenStream [7].

2.2. Evaluation Measures

After an algorithm has been proposed, it is common to evaluate how well it performs, or in other words, how well the algorithm is capable of splitting data into its true parts.

The simplest evaluation method is to calculate the average purity of all of the clusters. In information retrieval this score is also known as precision.

	Condition positive	Condition negative
Test outcome positive	True positive	False positive
Test outcome negative	False negative	True negative

$$Purity = \frac{True \ Positive}{True \ Positive + False \ Positive}$$
(2.1)

Additionally, one can also solve for the sensitivity, or what is also known as recall, of the model. Sensitivity is useful in measuring how well an algorithm is capable of returning an entire class of objects back in a single cluster.

$$Sensitivity = \frac{True \ Positive}{True \ Positive + False \ Negative}$$
(2.2)

Unfortunately, however, both precision and recall can be easily biased. For example, producing a cluster for each individual object would yield a perfect purity score. Meanwhile, a single cluster consisting of all objects would result in a perfect recall score. Both outcomes are undesirable because they unsuccessfully group the data in any meaningful way. In an attempt to remedy this problem, the F1 measure was developed as a harmonic mean of the two in order to measure the overall accuracy of a clustering.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$
(2.3)

Another popular method of evaluation includes the Rand index [11] and its derivatives. The Rand index attempts to find the similarity between two clusterings by measuring the number of correctly identified pairs of results that correspondence to the original clustering X and the solved clustering Y.

$$Rand = \frac{a+b}{a+b+c+d},\tag{2.4}$$

where a = the number of pairs of elements that are in the same

cluster in X and in the same cluster in Y,

- b = the number of pairs of elements that are in different clusters in X and in different clusters in Y,
- c = the number of pairs of elements that are in the same cluster in X and in different clusters in Y,
- d = the number of pairs of elements that are in different clusters in X and in the same cluster in Y

Similar to precision and recall, the Rand index can also exhibit bias [21]. For this reason, the adjusted Rand index [20] was created, which serves as a version of the original Rand index that corrects for randomness in the data. In other words, two random partitions will result in a constant score.

For the remainder of this paper, both F1 and the adjusted Rand index will be used to evaluate the accuracy by which different algorithms are able to cluster data generated by the **streamMovingGenerator** extension. These are by no means the only evaluation measures available in **stream**. Other examples include: chi squared coefficient [16], Fowlkes-Mallows index [8], Mirkin metric [15] and the Jaccard index [12].

2.3. MOA

To run evaluation tests, one first needs test data with a ground truth. At the time of writing this, MOA, which stands for Massive On-line Analysis [5], contains one of the only data stream generators publicly available for researchers. Written in Java, it comes with its own GUI that allows users to easily create experiments based on different machine learning algorithms. MOA also has strong connections with WEKA [10], allowing users to use any WEKA algorithm within the context of MOA.



Figure 2.1. Screenshot of the MOA Stream Clustering GUI

For this paper, I will only investigate the various stream generators that come packaged within MOA. The exhaustive list of generators includes the following:

- Agrawal [3]
- Hyperplane

- LED
- Random Radial Basis Function
- Random Tree
- SEA Concept Function [18]
- STAGGER Concept Functions [17]
- Waveform

Together, these data stream generators can create a wide range of benchmarks that can be altered using several different parameters, including the number of clusters, the number of attributes and the amount of noise. This allows users to test both the algorithms introduced by MOA and those found within WEKA.

The generators within MOA do have their shortcomings, however. For one, most of the included streams use functions that are purely random. While a user can create repeatable experiments by providing a seed to MOA, it is impossible to dictate exactly how each model should behave. Instead, a user must repeat an experiment with different seeds until the desired behavior is achieved. This process is both cumbersome and time consuming. Secondly, with the exception of the random radial basis function data stream, which is also available via **stream**, none of the generators are able to model evolutionary characteristics like branching and merging. Both of these characteristics are crucial in determining the robustness of a particular algorithm because they are common behaviors found within real data streams.

Chapter 3

streamMovingGenerator

Within the **stream** package there exist two main components from which all other classes extend from, data stream data and data stream tasks. Data stream data (DSD) is responsible for the generation and the presentation of data while data stream tasks (DSTs) are used to perform various actions on the DSDs, such as clustering and evaluation. Because **streamMovingGenerator** is primarily concerned with the artificial generation of evolving datasets, it exclusively extends the DSD class. In addition, **streamMovingGenerator** introduces the concept of moving generator clusters (MGCs) that can belong to different DSDs. MGCs are objects that can encapsulate all of the necessary behaviors of a single cluster. It is then up to a moving generator DSD (DSD_MG), or a similar DSD, to coordinate the behaviors of the individual clusters to generate a single stream. This allows for wildly different behavior between different clusters within the same DSD object.

DSD_MG	
MGC *	

Figure 3.1. UML Diagram of the Relationship between Moving Generator and Moving Generator Clusters

Figure 3.1 shows a high level view of the relationship between a DSD_MG and a set of MGCs. A user merely creates his or her own MGC object(s) and then assigns them to a DSD_MG. Together, each object works to create a data stream. The DSD_MG can then be manipulated and used by a DST from the **stream** framework to accomplish any task already defined in **stream**.

The streamMovingGenerator's DSD_MG is able to tap into the functions already defined in the stream framework because stream uses the S4 class system within R to provide an inheritance hierarchy for the various classes within the framework. By adopting this method of inheritance, streamMovingGenerator is easily able to extend the existing classes. For example, S4 allows for the new DSD_MG class to be identified as any of the following: DSD; the abstract base class of all DSDs; DSD_R, defines the interface for all DSDs implemented directly in R; and DSD_MG, its specific class implemented in the stream framework while also allowing it to seamlessly interact with other stream objects through the use of polymorphism. Figure 3.2 shows how the newly developed DSD relates to the other DSDs currently existing within the stream framework.



Figure 3.2. UML Diagram of the DSD Inheritance Hierarchy

Not all of the new classes developed for **streamMovingGenerator** are extensions of classes within **stream**, however. Because MGCs are a novel addition to the framework, they were added as a new class independent of both the DSDs and DSTs found within **stream**. Nevertheless, the MGCs introduced also took advantage of the S4 class system, allowing us to create a similar hierarchical inheritance system to create several different types of MGCs with a common ancestor. By taking advantage of inheritance within the S4 class system, the DSD_MG class was able to call the same generic functions shared by all of the MGCs.

The next several sections provide further details on the different components of **streamMovingGenerator** along with how to use each component.

3.1. Moving Generator Cluster

Traditionally, the first step in using the **stream** framework would be to begin initializing a DSD, but because MGCs are components of the new moving generator DSD, defining MGCs should be the first step in one's workflow. As mentioned previously, all MGC classes extend from the same abstract base class, MGC. The currently available MGC classes include the following: MGC_Linear, an MGC that is defined by a set of keyframes; MGC_Function, an MGC controlled by a set of defined functions; MGC_Noise, an MGC that generates noise; and MGC_Random, an MGC that creates randomly generated behavior. Figure 3.3 shows the various different types of MGCs and how they all inherit from the same core MGC class.



Figure 3.3. UML Diagram of Moving Generator Cluster Inheritance

Each MGC has a unique constructor that is used to determine the core aspects of each cluster, such as the variance, density, and center. The density is the number of points generated at each time step. Examples of each of the constructors is demonstrated below.

First, there is MGC_Function, that requires a list of all of the functions that should define the cluster's behavior.

```
> mgc_function <- MGC_Function(variance = function(t){sin(t/5)},
+ density = function(t){25},
+ center = list(function(t){t},
+ function(t){t}))
> mgc_function
```

Functional Moving Generator Cluster (MGC_Function, MGC)

In 2 dimensions

Secondly is MGC_Random, which requires a starting point and how much it should move in one time interval with the **randomness** parameter.

```
> mgc_random <- MGC_Random(variance = 1, density = 20,
+ center = c(50, 50), randomness = .25)
> mgc_random
```

```
Random Moving Generator Cluster (MGC_Random, MGC)
```

In 2 dimensions

Thirdly, there is MGC_Noise, which requires the area in which noise should be present.

```
> mgc_noise <- MGC_Noise(density = 1,
+ range = rbind(c(-20,120),c(-20,120)))
> mgc_noise
```

Noise Moving Generator Cluster (MGC_Noise, MGC) In 2 dimensions

And then lastly, there is MGC_Linear.

> mgc_linear <- MGC_Linear()</pre>

> mgc_linear

```
Linear Moving Generator Cluster (MGC_Linear, MGC)
With 0 keyframes in ? dimensions
```

For MGC_Function, MGC_Noise and MGC_Random, the constructor is enough to provide all of the necessary information required to have it appropriately function within a DSD_MG. You may have noticed, however, that MGC_Linear's constructor is fairly sparse. Unlike the other MGCs, MGC_Linear stores all of its attributes within keyframes.

Similar to keyframes in animation software like Adobe's Flash, the keyframes in an MGC_Linear define the starting and ending points of a smooth transition. Given a set of keyframes, an MGC_Linear will attempt to make a linear transition from one keyframe to the next until all of the keyframes have been exhausted.

To create a keyframe, a user evokes the add_keyframe() function with the necessary parameters.

```
> add_keyframe(mgc_linear, time = 1, variance = 10,
+ density = 10, center = c(0, 0), cluster=1)
> add_keyframe(mgc_linear, time = 50, variance = 10,
+ density = 10, center = c(50, 50), cluster=1)
> add_keyframe(mgc_linear, time = 100, variance = 10,
+ density = 10, center = c(75, 50), cluster=1)
> mgc_linear
```

Linear Moving Generator Cluster (MGC_Linear, MGC)

With 3 keyframes in 2 dimensions

Likewise, to remove a keyframe, a user calls the **remove_keyframe()** function with the timestamp of the keyframe he or she wants to remove.

```
> remove_keyframe(mgc_linear, 50)
```

> mgc_linear

```
Linear Moving Generator Cluster (MGC_Linear, MGC)
With 2 keyframes in 2 dimensions
```

Lastly, to get a list of all keyframes the user calls the get_keyframes() function.

```
> get_keyframes(mgc_linear)
```

	time	variance	density	cluster	centers	reset
1	1	10	10	1	0, 0	FALSE
3	100	10	10	1	75, 50	FALSE

As of right now, MGC_Function and MGC_Linear can only create Gaussian clusters.

3.2. Data Stream Data Moving Generator

The data stream moving generator (DSD_MG) is the real workhorse of the **stream-MovingGenerator** package. As the conductor for all MGCs, the DSD_MG class is responsible for keeping track of the current time relative to the data stream, requesting the attributes from the MGCs and using the extracted attributes to create a stream that is understandable to the other components within the **stream** framework.

The DSD_MG's constructor is primarily used to set up the initial set of MGCs. While it may be common to only use one type of MGC within a DSD_MG, a DSD_MG can accommodate any mixture of MGCs. A common combination is to include an MGC_Noise object to a set of other MGCs so that a layer of noise can be added to the entire stream. Below is an example of creating a new DSD_MG with MGCs created in the previous section.

> dsd_mg <- DSD_MG(dimension = 2, mgc_linear, mgc_function)</pre>

If one forgets to add an MGC to a DSD_MG or if one ends up creating a new MGC, one could add it after initializing an DSD_MG object through the use of the add_cluster() function.

```
> add_cluster(dsd_mg, mgc_random)
```

> dsd_mg

Moving Data Generator (DSD_MG, DSD_R, DSD) With 3 clusters in 2 dimensions

Similarly, one can remove a cluster by calling the **remove_cluster()** function with the cluster index as the second parameter.

```
> remove_cluster(dsd_mg, 3)
```

> dsd_mg

Moving Data Generator (DSD_MG, DSD_R, DSD) With 2 clusters in 2 dimensions

Lastly, one can get all of the clusters by using the get_clusters() function.

> get_clusters(dsd_mg)

[[1]]

Linear Moving Generator Cluster (MGC_Linear, MGC) With 2 keyframes in 2 dimensions

[[2]]

Functional Moving Generator Cluster (MGC_Function, MGC) In 2 dimensions

The remainder of the functions for DSD_MG are inherited from both DSD_R and DSD. To gain a further grasp on these functions, either refer to the **stream** documentation or follow the examples provided during the analysis portion of this paper.

Chapter 4

Analysis with Data Moving Generator

The following section of the paper is dedicated to using the **streamMovingGenerator** extension to test and analyze tNN [9], CluStream [2], ClusTree [13] and Den-Stream [7] on various behaviors that a data stream may exhibit over a period of time.

4.1. Splitting Clusters

The most rudimentary evolutionary behavior that a data stream may display is the splitting or branching of two clusters. This occurs when a single cluster becomes two distinct clusters, such as in Figure 4.1. An ideal algorithm should be able to readily detect separations as soon as they occur and react accordingly.



Figure 4.1. Splitting of a Cluster

4.1.1. Initializing Data Stream

The split behavior can be recreated using the **streamMovingGenerator** package by first creating two different MGC_Linear objects. > c1 <- MGC_Linear()
> c2 <- MGC_Linear()</pre>

Afterwards, set the keyframes for each individual cluster. At the beginning, both clusters will belong to the same cluster. To achieve this effect, each cluster off in the same location and assign each cluster the same cluster label with the cluster attribute.

```
> add_keyframe(c1, time = 1, variance = 10, density = 10,
+ center = c(0, 50), cluster = 1)
> add_keyframe(c2, time = 1, variance = 10, density = 10,
+ center = c(0, 50), cluster = 1)
```

Next, define the point where the two clusters diverge. One can accomplish this by changing the class label of one of the MGCs at the midway point.

```
> add_keyframe(c1, time = 50, variance = 10, density = 10,
+ center = c(50, 50), cluster = 1)
> add_keyframe(c2, time = 50, variance = 10, density = 10,
+ center = c(50, 50), cluster = 2)
```

Lastly, define the final keyframe for each cluster and add each of the clusters to a DSD_MG.

```
> add_keyframe(c1, time = 100, variance = 10, density = 10,
+ center = c(100, 100), cluster = 1)
> add_keyframe(c2, time = 100, variance = 10, density = 10,
+ center = c(100, 0), cluster = 2)
> dsd <- DSD_MG(dimension = 2, c1, c2)</pre>
```

By creating a DSD_MG object one can perform various tasks defined within **stream**, such as previewing an animated version of the data with the **animate_data()** function shown below.

> animate_data(dsd, 2500, xlim=c(-20,120), ylim=c(-20,120))



Figure 4.2. Plot Generated by animate_data()

4.1.2. Clustering and Evaluation

The next logical step one would take once one has created his or her own data stream model is to run one or many clustering algorithms to determine how each algorithm behaves.

There are several different ways of clustering DSD objects in the **stream** package, such as the **cluster()** and **animate_cluster()** functions. I will use the **animate_cluster()** function for the remainder of the paper because it provides a visualization of the clustering process over time and an accompanying graph of the evaluation score over time, both of which are useful in determining the behavior of each algorithm during the span of the data stream's life. The evaluation process is described in detail within **stream** [6].

For the first example of clustering, I will use the tNN algorithm found within the **stream** package. The results should be similar to those found in Figure 4.3.

```
> reset_stream(dsd)
```

```
> dsc <- DSC_tNN(r=10, lambda=.1, shared_density=TRUE, alpha=.1)</pre>
```

```
> animate_cluster(dsc, dsd, n=2500, interval=.1,
```

```
+ evaluationMethod="f1", evaluationType="macro",
```

```
+ x \lim = c(-20, 120), y \lim = c(-20, 120),
```

```
+ type="shared")
```

It is clear from Figure 4.3, that tNN is extremely effective at finding the clusters before and after the cluster splits apart. While the score dips during the actual separation process, tNN is able to quickly realize the existence of two individual clusters and is able to recover fairly quickly, revealing that the tNN algorithm is quite adept at modeling separations in the data stream. The tNN algorithm achieves this by aging older clusters with the lambda parameter until they disapear.

Running DenStream produces similar results. CluStream and ClusTree, however, reveal a different outcome. Because they rely on a reclustering algorithm like k-means or hierarchical clustering, they are unable to detect a variable number of clusters throughout the clustering process. In other words, they can either look for one cluster and miss the separation, or search for two clusters and attempt to find separations at the beginning where none exists.

Figure 4.4 shows the F1 scores for all of the algorithms. Notice that scores for CluStream and ClusTree initially start lower. This is due to the fact that both algorithms are searching for two clusters where there are none.



Figure 4.3. F1 Score from tNN using the Splitting Data Stream



Figure 4.4. F1 Score for Splitting Data Stream

4.2. Merging Clusters

The converse of splitting a cluster would be merging two or more together. A depiction of two clusters merging together can be seen in Figure 4.5. An ideal algorithm would be able to quickly combine separate clusters as soon as they came into contact with each other. To increase the difficulty of modeling this data stream, noise will also be introduced.



Figure 4.5. Merging of Clusters

4.2.1. Initializing Data Stream

To create the desired merging effect, I will once again use a pair of MGC_Linears as I did in the splitting example. Only this time I will reverse the keyframes so that the opposite behavior is achieved. I will also add an MGC_Noise to the DSD_MG to provide background noise for the data stream.

4.2.2. Clustering and Evaluation

In this example, I will begin with tNN and DenStream.

```
> tnn <- DSC_tNN(r=5, lambda=.1, shared_density=TRUE,
+ alpha=.3, noise=.15)
> animate_cluster(tnn, dsd, n=2500, interval=.1,
+ evaluationMethod="f1", evaluationType="macro",
+ xlim = c(-20, 120), ylim = c(-20, 120),
+ type="shared")
```



Figure 4.6. F1 Score Over Time for tNN and DenStream for Merging Data Stream

```
> reset_stream(dsd)
> denstream <- DSC_DenStream(epsilon=5,lambda=.2)
> animate_cluster(denstream, dsd, n=2500, interval=.1,
+ evaluationMethod="f1", evaluationType="macro",
+ xlim = c(-20, 120), ylim = c(-20, 120),
```

```
+ type="both")
```

The resulting F1 scores can be seen in Figure 4.6. Both tNN and Denstream are easily able to maintain an F1 score above 0.9 for the majority of the time, revealing they can tolerate a significant amount of noise. The largest drop in score occurs when the two clusters begin to touch. Even though the two clusters have yet to fully merge, both algorithms immediately link them together, resulting in a plummeting score. This reveals that tNN and DenStream are extremely susceptible to prematurely linking two clusters.

As before, the other two algorithms perform poorly due to their inability to dynamically alter the number of clusters they produce. They are able to do somewhat well because the secondary cluster moves to where the noise is. Additionally, CluStream and ClusTree are significantly impacted by noise, as can be seen in Figure 4.7 by the widely inconsistent scores.



Figure 4.7. F1 Score Over Time for CluStream and ClusTree for Merging Data Stream

4.3. Passing Clusters

Unlike the other two examples, this simulation preserves the class of each cluster throughout the entire data stream. Instead of merging when the clusters come into contact with one another, they merely pass each other's path as seen in Figure 4.8. An ideal algorithm would be able to maintain the separate identities of each cluster even as they overlap one another. Noise will be applied to this situation to make it more difficult to cluster.

MGC_Linear could be used again, but I will show how to model this behavior with MGC_Function instead.



Figure 4.8. Passing of Clusters

4.3.1. Initializing Data Stream

In this situation I will use two MGC_Function objects and an MGC_Noise to create the desired behavior. The following code produces two clusters. The first cluster sweeps across from left to right while the second moves from bottom to top. They momentarily meet in the middle before continuing on in their respective directions.

```
> c1 <- MGC_Function(variance = function(t){1},</pre>
                       density = function(t){10},
+
                       center = list(function(t){t},
+
                                       function(t){50})
+
> c2 <- MGC_Function(variance = function(t){1},</pre>
                       density = function(t) \{10\},
                       center = list(function(t) \{50\},
+
+
                                       function(t){t}))
> noise <- MGC_Noise(density = 2,</pre>
                       range = rbind(c(-20, 120), c(-20, 120)))
+
> dsd <- DSD_MG(2, c1, c2, noise)</pre>
```

4.3.2. Clustering and Evaluation

Because there is a constant number of clusters throughout the data stream, CluStream and ClusTree can each be successfully used to cluster this data stream instead of just tNN and DenStream. The following code details how to run tests on all four clustering algorithms. It should be noted that reset_stream() is called between each clustering algorithm to ensure that the stream always starts at the beginning.

```
> animate_cluster(tnn, dsd, n=2500, interval=.1,
```

- + evaluationMethod="cRand", evaluationType="macro",
- + $x \lim = c(-20, 120), y \lim = c(-20, 120),$
- + type="shared")

```
> reset_stream(dsd)
```

```
> kmeans <- DSC_Kmeans(2)</pre>
```

> clustream <- DSC_CluStream(k=10,t=5,horizon=10)</pre>

```
> animate_cluster(clustream, dsd, n=2500, interval=.1,
```

+ evaluationMethod="cRand", evaluationType="macro",

```
+ x \lim = c(-20, 120), y \lim = c(-20, 120),
```

```
+ type="both", macro=kmeans)
```

```
> reset_stream(dsd)
```

```
> clustree <- DSC_ClusTree(maxHeight=8,horizon=10)</pre>
```

- > animate_cluster(clustree, dsd, n=2500, interval=.1,
- + evaluationMethod="cRand", evaluationType="macro",

```
+ x \lim = c(-20, 120), y \lim = c(-20, 120),
```

```
+ type="both", macro=kmeans)
```

```
> reset_stream(dsd)
```

```
> denstream <- DSC_DenStream(epsilon=5,lambda=.1)</pre>
```

```
> animate_cluster(denstream, dsd, n=2500, interval=.1,
```

```
+ evaluationMethod="cRand", evaluationType="macro",
```

```
+ x \lim = c(-20, 120), y \lim = c(-20, 120),
```

```
+ type="both")
```

As seen in Figure 4.9, both CluStream and CluStree perform fairly poorly due to the introduction of noise. Both tNN and DenStream, on the other hand, are significantly more stable, highlighting their ability to effectively ignore noise.



Figure 4.9. Adjusted Rand Index for Passing Data Stream

Chapter 5

Conclusion and Future Work

The **streamMovingGenerator** extension provides a robust method of creating evolving data streams, a process that would normally be an expensive and arduous task due to the complicated nature of non-static datasets. By harnessing the power of artificial data streams, structured tests can be carried out to measure the quality of existing and future data stream clustering algorithms.

Additionally, in the spirit of the original **stream** package, **streamMovingGenerator** is capable of being extended to accomodate a multitude of use cases. Most notably, it is capable of allowing for custom MGCs that can be used in conjunction with the defined DSD_MG class to develop even more complicated streams, such as streams with non-gaussian clusters or looping behavior.

Using the **streamMovingGenerator** extension can be used to evaluate and test existing clustering algorithms to help clearly identify the strengths of weaknesses of the field today so that more robust algorithms can be developed in the coming years.

Appendix A

stream Contributions

- Refactored significant portions of the code base
- Developed the ability to recluster models
- Added support for macro clustering algorithms
- Helped extensivley develope tNN
- Created tNN_fast, a version of tNN written primarily in C
- Developed the evaluation functions
- Provided means to animate data streams and clustering algorithms

REFERENCES

- AGGARWAL, C. C., Ed. Data Streams Models and Algorithms, vol. 31 of Advances in Database Systems. Springer, 2007.
- [2] AGGARWAL, C. C., HAN, J., WANG, J., AND YU, P. S. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference* on Very Large Data Bases - Volume 29 (2003), VLDB '03, VLDB Endowment, pp. 81–92.
- [3] AGRAWAL, R., IMIELINSKI, T., AND SWAMI, A. Database mining: A performance perspective. *IEEE Trans. on Knowl. and Data Eng.* 5, 6 (Dec. 1993), 914–925.
- [4] BACHE, K., AND LICHMAN, M. UCI machine learning repository, 2013.
- [5] BIFET, A., HOLMES, G., PFAHRINGER, B., KRANEN, P., KREMER, H., JANSEN, T., AND SEIDL, T. MOA: Massive online analysis, a framework for stream classification and clustering. In *Journal of Machine Learning Research* (*JMLR*) Workshop and Conference Proceedings, Volume 11: Workshop on Applications of Pattern Analysis (2010), Journal of Machine Learning Research, pp. 44–50.
- [6] BOLANOS, M., FORREST, J., AND HAHSLER, M. Infrastructure for Data Stream Mining, 2013. R package version 0.1-1.
- [7] CAO, F., ESTER, M., QIAN, W., AND ZHOU, A. Density-based clustering over an evolving data stream with noise. In *In 2006 SIAM Conference on Data Mining* (2006), pp. 328–339.
- [8] FOWLKES, E. B., AND MALLOWS, C. L. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association* 78, 383 (1983), 553–569.
- [9] HAHSLER, M., AND DUNHAM, M. H. remm: Extensible markov model for data stream clustering in r. Journal of Statistical Software 35, 5 (7 2010), 1–31.
- [10] HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., AND WITTEN, I. H. The weka data mining software: An update. SIGKDD Explor. Newsl. 11, 1 (Nov. 2009), 10–18.

- [11] HUBERT, L., AND ARABIE, P. Comparing partitions. Journal of Classification 2 (1985), 193–218.
- [12] JACCARD, P. Étude comparative de la distribution florale dans une portion des alpes et des jura. Bulletin del la Société Vaudoise des Sciences Naturelles 37 (1901), 547–579.
- [13] KRANEN, P., ASSENT, I., BALDAUF, C., AND SEIDL, T. The clustree: Indexing micro-clusters for anytime stream mining. In *Knowledge and Information Systems Journal (Springer KAIS), Volume 29, Issue 2* (London, 2011), Springer, pp. 249–272.
- [14] LEISCH, F., AND DIMITRIADOU, E. mlbench: Machine Learning Benchmark Problems, 2010. R package version 2.1-1.
- [15] LEVIN, M. B. Mirkin, mathematical classification and clustering. Journal of Global Optimization 12, 1 (1998), 105–108.
- [16] MIRKIN, B. Eleven ways to look at the chi-squared coefficient for contingency tables. *The American Statistician* 55 (2001), 111–120.
- [17] SCHLIMMER, J. C., AND GRANGER, JR., R. H. Incremental learning from noisy data. Mach. Learn. 1, 3 (Mar. 1986), 317–354.
- [18] STREET, W. N., AND KIM, Y. A streaming ensemble algorithm (SEA) for largescale classification. In Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA, 2001), KDD '01, ACM, pp. 377–382.
- [19] TAN, P.-N., STEINBACH, M., AND KUMAR, V. Introduction to Data Mining, (First Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
- [20] VINH, N. X., EPPS, J., AND BAILEY, J. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? In *Proceedings of* the 26th Annual International Conference on Machine Learning (New York, NY, USA, 2009), ICML '09, ACM, pp. 1073–1080.
- [21] WAGNER, S., AND WAGNER, D. Comparing Clusterings An Overview. Tech. Rep. 2006-04, Universität Karlsruhe (TH), 2007.