

# On Demand Classification of Data Streams

Charu C. Aggarwal  
IBM T. J. Watson Research  
Center  
charu@us.ibm.com

Jiawei Han<sup>\*</sup>  
Jianyong Wang<sup>†</sup>  
UIUC  
{ hanj, wangj }@uiuc.edu

Philip S. Yu  
IBM T. J. Watson Research  
Center  
psyu@us.ibm.com

## ABSTRACT

Current models of the classification problem do not effectively handle bursts of particular classes coming in at different times. In fact, the current model of the classification problem simply concentrates on methods for one-pass classification modeling of very large data sets. Our model for data stream classification views the data stream classification problem from the point of view of a dynamic approach in which *simultaneous* training and testing streams are used for *dynamic* classification of data sets. This model reflects real life situations effectively, since it is desirable to classify test streams in real time over an evolving training and test stream. The aim here is to create a classification system in which the training model can adapt quickly to the changes of the underlying data stream. In order to achieve this goal, we propose an on-demand classification process which can dynamically select the appropriate window of past training data to build the classifier. The empirical results indicate that the system maintains a high classification accuracy in an evolving data stream, while providing an efficient solution to the classification task.

## Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—  
*Data Mining*

## General Terms

Algorithms

## Keywords

data streams, classification

<sup>\*</sup>The work was supported in part by the U.S. National Science Foundation Grant IIS-03-08215 and an IBM Faculty Award.

<sup>†</sup>Current Address: University of Minnesota at Twin-Cities, Minneapolis, MN 55455. **Email:** jianyong@cs.umn.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'04, August 22–25, 2004, Seattle, Washington, USA.  
Copyright 2004 ACM 1-58113-888-1/04/0008 ...\$5.00.

## 1. INTRODUCTION

In recent years, advances in data storage technology have led to the ability to store the data for real time transactions. Such processes lead to data which often grow without limit and are referred to as data streams [3]. One important data mining problem which has been studied in the context of data streams is that of classification [5]. The main thrust on data stream mining in the context of classification has been that of one-pass mining [4, 6]. In general, the use of one-pass mining does not recognize the changes which have occurred in the model since the beginning of the stream construction process [2]. While the work in [6] works on time changing data streams, the focus is on providing effective methods for incremental updating of the classification model. We note that the accuracy of such a model cannot be greater than the best sliding window model on a data stream. As our empirical results will show, the true behavior of the data stream is captured in a temporal model which is sensitive to the level of evolution of the data stream.

The classification process may require simultaneous model construction and testing in an environment which constantly evolves over time. We assume that the testing process is performed concurrently with the training process. This is often the case in many practical applications, in which only a portion of the data is labeled, whereas the remaining is not. Therefore, such data can be separated out into the (labelled) training stream, and the (unlabelled) testing stream. The most effective classification model to be used does not stay constant over time, but varies with progression of the data stream. If a static classification model were used for an evolving test stream, the accuracy of the underlying classification process is likely to drop suddenly when there is a sudden burst of records belonging to a particular class. In such a case, a classification model which is constructed using a smaller history of data is likely to provide better accuracy. In other cases, a longer history of training provides greater robustness.

In the classification process of an evolving data stream, either the short term or long term behavior of the stream may be more important, and it often cannot be known a priori as to which one is more important. How do we decide the window or horizon of the training data to use so as to obtain the best classification accuracy? While techniques such as decision trees are useful for one-pass mining of data streams [4, 6], these cannot be easily used in the context of an *on-demand classifier* in an evolving environment. This is because such a classifier requires rapid variation in the horizon selection process due to data stream evolution. Further-

more, it is too expensive to keep track of the entire history of the data in its original fine granularity. Therefore, the on-demand classification process still requires the appropriate machinery for efficient statistical data collection in order to perform the classification process.

This paper is organized as follows. The basic statistical constructs of the paper are discussed in the next section. Section 3 discusses how these statistics may be used in order to perform on-demand classification of a test stream. The empirical results are discussed in section 4. Section 5 contains the conclusions and summary.

## 2. BASIC CONSTRUCTS

It is assumed that the training and test data streams each consist of a set of multi-dimensional records  $\overline{X}_1 \dots \overline{X}_k \dots$  arriving at time stamps  $T_1 \dots T_k \dots$ . Each  $\overline{X}_i$  is a multi-dimensional record containing  $d$  dimensions which are denoted by  $\overline{X}_i = (x_i^1 \dots x_i^d)$ . In addition, each record  $\overline{X}_i$  in the training data stream is associated with a class label  $C_i$ . We assume that the class\_id of the class  $C_i$  is  $i$ .

We will first begin by defining the concept of *supervised* micro-clusters. While the micro-clustering concept of [1] is useful for unsupervised clustering, we need to make modifications in order to use this approach for the classification process. The supervised micro-clusters are created from the training data stream only. Each such micro-cluster corresponds to a set of points from the training data, all of which belong to the same class.

**DEFINITION 2.1.** *A supervised micro-cluster for a set of  $d$ -dimensional points  $X_{i_1} \dots X_{i_n}$  with time stamps  $T_{i_1} \dots T_{i_n}$  and belonging to the class  $class\_id$  is defined as the  $(2 \cdot d + 4)$  tuple  $(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n, class\_id)$ , wherein  $\overline{CF2^x}$  and  $\overline{CF1^x}$  each correspond to a vector of  $d$  entries. The definition of each of these entries is as follows:*

- For each dimension, the sum of the squares of the data values are maintained in  $\overline{CF2^x}$ . Thus,  $\overline{CF2^x}$  contains  $d$  values. The  $p$ -th entry of  $\overline{CF2^x}$  is equal to  $\sum_{j=1}^n (x_{i_j}^p)^2$ .
- For each dimension, the sum of the data values are maintained in  $\overline{CF1^x}$ . Thus,  $\overline{CF1^x}$  contains  $d$  values. The  $p$ -th entry of  $\overline{CF1^x}$  is equal to  $\sum_{j=1}^n x_{i_j}^p$ .
- The sum of the squares of the time stamps  $T_{i_1} \dots T_{i_n}$  are maintained in  $CF2^t$ .
- The sum of the time stamps  $T_{i_1} \dots T_{i_n}$  are maintained in  $CF1^t$ .
- The number of data points are maintained in  $n$ .
- The variable corresponding to  $class\_id$  corresponds to the class label of that micro-cluster.

The above definition of the supervised micro-cluster for the set of points  $\mathcal{C}$  is denoted by  $\overline{CF\overline{T}}(\mathcal{C})$ . This summary information is an extension of the cluster feature vector concept discussed in [7]. Since each component in the definition of the micro-cluster is an additive sum over different data points, this data structure can be updated easily over different data streams.

We note that the nature of the micro-clusters and their distribution over different classes may change considerably

Frame no.	Snapshots (by clock time)
0	69 67 65
1	70 66 62
2	68 60 52
3	56 40 24
4	48 16
5	64 32

**Table 1: A geometric time window**

over time. Therefore, the effectiveness of the classification model may be highly sensitive to the length of the horizon used for the training process. In general, we would like to use a horizon which provides the highest accuracy of the corresponding classification model. This can be achieved by storing the behavior of the micro-clusters at different moments in time. These stored micro-cluster states are referred to as *snapshots*.

The snapshots need to be stored in such a way that sufficient amount of information is maintained about different time horizons. At the same time, the storage of an unnecessarily large number of time horizons makes the scheme time and space inefficient. This is achieved with the use of a *geometric time frame*. In this technique, snapshots are classified into different *frame numbers* which can vary from 0 to a value no larger than  $\log_2(T)$ , where  $T$  is the maximum length of the stream. The frame number of a particular class of snapshots defines the level of granularity in time at which the snapshots are maintained. Specifically, snapshots of frame number  $i$  are stored at clock times which are divisible by  $2^i$ , but not by  $2^{i+1}$ . Therefore, snapshots of frame number 0 are stored only at odd clock times. It is assumed that for each frame number, at most *max\_capacity* snapshots are stored.

We note that for a data stream, the maximum frame number of any snapshot stored at  $T$  time units since the beginning of the stream mining process is  $\log_2(T)$ . Since at most *max\_capacity* snapshots of any order are stored, this also means that the maximum number of snapshots maintained at  $T$  time units since the beginning of the stream mining process is  $(max\_capacity) \cdot \log_2(T)$ . One interesting characteristic of the geometric time window is that for any user-specified time window of  $h$ , at least one stored snapshot can be found within a factor of 2 of the specified horizon. This ensures that sufficient granularity is available for analyzing the behavior of the data stream over different time horizons. We will formalize this result in the lemma below.

**LEMMA 2.1.** *Let  $h$  be a user-specified time window, and  $t_c$  be the current time. Let us also assume that  $max\_capacity \geq 2$ . Then a snapshot exists at time  $t_s$ , such that  $h/2 \leq t_c - t_s \leq 2 \cdot h$ .*

**Proof.** Omitted.

The above result ensures that every possible horizon can be closely approximated within a modest level of accuracy. While the geometric time frame shares a number of conceptual similarities with the pyramidal time frame [1], it is actually quite different and also much more efficient. This is because it eliminates the double counting of the snapshots over different frame numbers, as is the case with the pyramidal time frame [1]. In Table 1, we present an exam-

ple of a frame table illustrating snapshots of different frame numbers. The rules for insertion of a snapshot  $t$  (at time  $t$ ) into the snapshot frame table are defined as follows: (1) if  $(t \bmod 2^i) = 0$  but  $(t \bmod 2^{i+1}) \neq 0$ ,  $t$  is inserted into *frame\_number*  $i$  (2) each slot has a *max\_capacity* (which is 3 in our example). At the insertion of  $t$  into *frame\_number*  $i$ , if the slot already reaches its *max\_capacity*, the oldest snapshot in this frame is removed and the new snapshot inserted. For example, at time 70, since  $(70 \bmod 2^1) = 0$  but  $(70 \bmod 2^2) \neq 0$ , 70 is inserted into *frame\_number* 1 which knocks out the oldest snapshot 58 if the slot capacity is 3. Following this rule, when slot capacity is 3, the following snapshots are stored in the geometric time window table: 16, 24, 32, 40, 48, 52, 56, 60, 62, 64, 65, 66, 67, 68, 69, 70, as shown in Table 1. From the table, one can see that the closer to the current time, the denser are the snapshots stored.

Since the results of this paper are tailored towards a rapidly evolving data stream, the class structure of the underlying data stream could change quickly during the classification process. For example, a new class which has not been observed in the entire history of the stream may suddenly emerge because of the changes in the underlying process which generates the stream. In such a case, if the entire history of the stream is used for classification, the results are likely to be inaccurate because the recent class arrivals are not reflected in the rest of the stream. Therefore, in such a case it may be desirable to use a smaller and more recent portion of the stream for the classification process. Another example is a case in which a particular class may not have arrived in the stream for a long period of time, but may suddenly re-emerge at some point. In such a case, a well chosen horizon would use a sufficiently long history, so that the previous occurrence of the classes would be included in the training process. Therefore, mechanisms are needed to make decisions on the suitability of using a horizon of a given length.

In order to achieve this goal, the incoming training data stream is divided into two parts:

- A small portion of the stream is used for the process of *horizon fitting*. The corresponding portion of the training stream is referred to as the horizon fitting stream segment. The number of points in the data used is denoted by  $k_{fit}$ . We note that the value of  $k_{fit}$  is typically very small such as 1% of the data.
- The remaining majority of the stream is used for accumulation of the pertinent statistics corresponding to the micro-clusters and class information.

The process of maintenance of supervised micro-clusters belonging to different classes derives ideas from the nearest neighbor and  $k$ -means algorithms. Because of the supervised nature of the method, class labels need to be used during the clustering process. At any moment in time, a maximum of  $q$  micro-clusters are maintained by the algorithm. We denote these micro-clusters by  $\mathcal{M}_1 \dots \mathcal{M}_q$ . Associated with each micro-cluster  $i$ , we create a unique *id* whenever it is first created. As we shall subsequently see, the micro-cluster maintenance algorithm requires a merging of multiple micro-clusters into one micro-cluster. Only micro-clusters that belong to the same class may be merged together during the clustering process. When two such micro-clusters are

merged, a *list* of *ids* is created in order to identify the constituent micro-clusters. The value of  $q$  is determined by the amount of main memory available in order to store the micro-clusters. The micro-clusters which are maintained in main memory correspond to the current snapshot of summary statistics.

The creation of the initial micro-clusters is achieved as an offline process at the beginning of the data stream processing. For this purpose, the first *InitNumber* points are stored on disk. An offline clustering algorithm is applied to the disk resident points. An equal number of micro-clusters is created for each class by using a separate  $k$ -means algorithm on each class of data points. Thus, a separate set of micro-clusters is created for each class of the data in the initialization phase.

After the initialization phase, the micro-cluster maintenance phase is initiated. Whenever a new data point  $\overline{X}_{i_k}$  arrives, it needs to be inserted into a micro-cluster belonging to its own class. In some cases, no micro-cluster may be relevant to the current data point. In such a case, a new micro-cluster is created, and the current data point is placed in it.

First, we attempt to place the data point in some pre-existing micro-cluster. In order to do so, we find the distance of each data point to the micro-cluster centroids which belong to the same class. We find the closest cluster  $\mathcal{M}_p$  to the data point  $\overline{X}_{i_k}$ . However, in many cases, the data point  $\overline{X}_{i_k}$  may not be sufficiently close to  $\mathcal{M}_p$  and should be placed in a micro-cluster of its own. This may happen because of the arrival of an outlier point, or because of sudden changes in the data stream itself. A sudden change may lead to a new trend in the data stream which often exhibits itself in the form of a new micro-cluster. In order to decide whether a data point should belong to a pre-existing micro-cluster or should be placed in a new cluster of its own, we use the cluster feature vector of  $\mathcal{M}_p$ . This is done in order to compute a *maximum boundary* of the micro-cluster  $\mathcal{M}_p$ . If the data point  $\overline{X}_{i_k}$  lies within this maximum boundary, then it is added to the micro-cluster  $\mathcal{M}_p$  using the CF additivity property. The maximum boundary of the micro-cluster  $\mathcal{M}_p$  is defined in terms of the average deviation of the other points from the centroid of this cluster. Specifically, the micro-cluster is defined as a factor of  $t$  of the RMS deviation of the data points in  $\mathcal{M}_p$  from the centroid. We define this as the *maximal boundary factor*. This definition of the maximal boundary factor is valid only for clusters with more than one point. For a cluster with only one previous point, the maximum boundary is defined in a heuristic way. Specifically, we choose it to be  $r$  times that of the next closest cluster.

If the data point does not lie within the maximum boundary of the nearest micro-cluster, then a new micro-cluster must be created containing the data point  $\mathcal{X}_{i_k}$ . The class\_id of this newly created micro-cluster is the same as the class of  $\mathcal{X}_{i_k}$ . This newly created micro-cluster is assigned a new id which can identify it uniquely at any future stage of the data stream process. In order to insert this new micro-cluster, the number of other clusters must be reduced by one. This can be achieved by either deleting an old cluster or joining two micro-clusters which belong to the same class.

The micro-cluster statistics maintenance algorithm determines if it is desirable to delete any of the current micro-clusters. It is desirable to delete a micro-cluster, when it

is determined that such a cluster no longer has an active presence in the stream. In order to do so, we find the average time stamp of the last  $m$  arrivals in the current micro-cluster. This is estimated by using the mean and standard deviation of the last time stamps which have arrived in the current micro-cluster. The mean stamp of the micro-cluster is given by  $\mu = CF1^t/n$ . The standard deviation is given by  $\sigma = \sqrt{CF2^t/n - (CF1^t/n)^2}$ . The average time stamp of the last  $m$  arrivals is estimated by the  $m/(2 \cdot n)$ -th percentile of the points in each cluster. This timestamp is used as the approximate value of the recency. This value is also referred to as the *relevance stamp* of the corresponding micro-cluster. We determine if the least relevant time stamp is below a user-defined threshold  $\delta$ . In such a case, it can be eliminated and a new micro-cluster can be created with a unique *id* corresponding to the newly arrived data point  $\overline{X}_{i_k}$ . When all relevance stamps are larger than the user-defined threshold  $\delta$ , two micro-clusters are merged in order to create space for the new micro-cluster. Since the micro-clusters are class-specific, we always merge micro-clusters belonging to the same class. In order to determine the micro-clusters to be merged, we find the pairwise distance between micro-clusters belonging to the same class. The closest pair of micro-clusters are merged together. The process of merging micro-clusters also requires some additional book-keeping in terms of the *ids* associated with the individual micro-clusters. An *idlist* is created, which is a union of the corresponding *idlists* of the pair being merged. This ensures that there is clear tracking of the constituent micro-clusters after the merging process.

In addition to the process of micro-cluster maintenance, it is also necessary to store the micro-clusters on disk at each moment in time. At each such time, we store away the current set of micro-clusters (possibly on disk) together with their *id* list, and indexed by their time of storage. Outdated micro-clusters are deleted from disk.

### 3. CLASSIFICATION ON DEMAND

In this section, we will discuss the *On Demand Stream Classification Process*. In order to perform effective classification of the stream, it is important to find the correct time-horizon which should be used for classification. How do we find the most effective horizon for classification at a given moment in time? In order to do so, a small portion of the training stream is not used for the creation of the micro-clusters. This portion of the training stream is referred to as the horizon fitting stream segment. The number of points in the stream used for horizon fitting is denoted by  $k_{fit}$ . The remaining portion of the training stream is used for the creation and maintenance of the class-specific micro-clusters as discussed in the previous section.

Since the micro-clusters are based on the entire history of the stream, they cannot directly be used to test the effectiveness of the classification process over different time horizons. This is essential, since we would like to find the time horizon which provides the greatest accuracy during the classification process. For this purpose, we use the following subtractive property:

**PROPERTY 3.1.** *Let  $\mathcal{C}_1$  and  $\mathcal{C}_2$  be two sets of points such that  $\mathcal{C}_1 \supseteq \mathcal{C}_2$ . Then, the cluster feature vector  $\overline{CFT}(\mathcal{C}_1 - \mathcal{C}_2)$  is given by  $\overline{CFT}(\mathcal{C}_1) - \overline{CFT}(\mathcal{C}_2)$ .*

We denote the snapshot of micro-clusters at time  $t$  by  $\mathcal{S}(t)$ . Let us consider an example in which the current clock time is  $t_c$ , and it is desirable to use a horizon of length  $h$  in order to find the micro-clusters in the time period  $(t_c - h, t_c)$ . In such a case, we find the stored snapshot which occurs just before the time  $t_c - h$ . For each micro-cluster in the current set  $\mathcal{S}(t_c)$ , we find the list of *ids* in each micro-cluster. For each of the list of *ids*, we find the corresponding micro-clusters in  $\mathcal{S}(t_c - h')$ , and subtract the CF vectors for the corresponding micro-clusters in  $\mathcal{S}(t_c - h')$ . The resulting set of micro-clusters correspond to the time horizon  $(t_c - h, t_c)$ . We will denote this final set of micro-clusters created from the subtraction process by  $\mathcal{N}(t_c, h')$ .

Once the micro-clusters for a particular time horizon have been determined, they are utilized to determine the classification accuracy of that particular horizon. This process is executed periodically in order to adjust for the changes which have occurred in the stream in recent time periods. For this purpose, we use the horizon fitting stream segment. The last  $k_{fit}$  points which have arrived in the horizon fitting stream segment are utilized in order to test the classification accuracy of that particular horizon. The value of  $k_{fit}$  is chosen while taking into consideration the computational complexity of the horizon accuracy estimation. In addition, the value of  $k_{fit}$  should be small enough so that the points in it reflect the immediate locality of  $t_c$ . Typically, the value of  $k_{fit}$  should be chosen in such a way that the least recent point should be no larger than a pre-specified number of time units from the current time  $t_c$ . Let us denote this set of points by  $\mathcal{Q}_{fit}$ . Note that since  $\mathcal{Q}_{fit}$  is a part of the training stream, the class labels are known a-priori.

In order to test the classification accuracy of the process, each point  $\overline{X} \in \mathcal{Q}_{fit}$  is used in the following nearest neighbor classification procedure:

- We find the closest micro-cluster in  $\mathcal{N}(t_c, h)$  to  $\overline{X}$ .
- We determine the class label of this micro-cluster and compare it to the true class label of  $\overline{X}$ . The accuracy over all the points in  $\mathcal{Q}_{fit}$  is then determined. This provides the accuracy over that particular time horizon.

The accuracy of all the time horizons which are tracked by the geometric time frame are determined. The  $p$  time horizons which provide the greatest dynamic classification accuracy (using the last  $k_{fit}$  points) are selected for the classification of the stream. Let us denote the corresponding horizon values by  $\mathcal{H} = \{h_1 \dots h_p\}$ . We note that since  $k_{fit}$  represents only a small locality of the points within the current time period  $t_c$ , it would seem at first sight that the system would always pick the smallest possible horizons in order to maximize the accuracy of classification. However, this is often not the case for evolving data streams. Consider for example, a data stream in which the records for a given class arrive for a period, and then subsequently start arriving again after a time interval in which the records for another class have arrived. In such a case, the horizon which includes previous occurrences of the same class is likely to provide higher accuracy than shorter horizons. Thus, such a system dynamically adapts to the most effective horizon for classification of data streams. In addition, for a stable stream the system is also likely to pick larger horizons because of the greater accuracy resulting from use of larger data sizes.

The classification of the test stream is a separate process which is executed continuously throughout the algorithm. For each given test instance  $X_t$ , the above described nearest neighbor classification process is applied using each  $h_i \in \mathcal{H}$ . It is often possible that in the case of a rapidly evolving data stream, different horizons may report results in the determination of different class labels. The majority class among these  $p$  class labels is reported as the relevant class.

## 4. EMPIRICAL RESULTS

In order to evaluate the accuracy, efficiency, scalability, and sensitivity of our *On-Demand-Stream Classifier*, a thorough experimental and performance study was conducted using both real and synthetic datasets. The study validates the following claims: (1) The *On-Demand-Stream Classifier* has much higher classification accuracy in comparison with the simple one-pass classification algorithms over either the entire dataset or a selected sliding window; (2) The *On-Demand-Stream Classifier* has very good scalability in terms of dimensionality and the number of class labels, and has rather stable processing rate; and (3) The *On-Demand-Stream Classifier* is also rather space-efficient: it only needs to maintain a moderate number of class-labeled micro-clusters for each snapshot and a very small number of snapshots (i.e., the *max\_capacity*) for each order of the geometric time window in order to achieve a high stream classification accuracy.

All of our experiments were conducted on a PC with Intel Pentium III processor, 512 MB memory, and Windows XP professional operating system installed. We first tested the accuracy of our *On-Demand-Stream classifier* against the simple one-pass algorithms over either the entire data stream or a selected sliding window of the stream data. The *On-Demand-Stream classifier* was implemented according to the description in this paper. To compute the accuracy of the one-pass algorithm over the entire dataset, we used the latest snapshot of micro-clusters to classify the test stream, and for the one-pass algorithm over a sliding-window, we always used a fixed horizon (e.g., 8 time units) and computed the net snapshot of micro-clusters, and used these net micro-clusters to classify the test stream data.

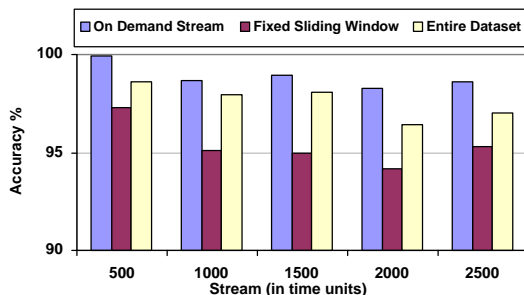
To test the processing rate and scalability of our classifier, we used the network intrusion data set and a synthetic data set. For the synthetic data set, we generated data points belonging to different classes according to a certain probability distribution and to reflect the evolution of the stream data over time, we randomly re-computed the probability of the appearance of a certain class periodically. This makes the classes evolve over time while several classes may appear in the same time slot. In addition, we use the following notations in naming the synthetic datasets: ‘B’ indicates the base size, i.e., the number of data points in the dataset, while ‘C’ and ‘D’ indicate the number of class labels and the dimensionality of each point, respectively. For example, for dataset B200kC30D30, it contains totally 200K 30-dimensional data points and 30 class labels.

We first evaluated the stream classification accuracy using the Network Intrusion Detection dataset. Here we define *microcluster-ratio* as the number of micro-clusters used in the *On-Demand-Stream classifier* divided by the number of natural class labels. Unless otherwise specified, the parameters were set as follows: *maximum boundary factor*  $t=2$ , *microcluster-ratio* $=5$ , *max\_capacity* $=32$ , *init\_number* $=400$ ,

and sliding window  $H=8$  time units. In addition, in the following experiments we store away the current snapshot of micro-clusters on disk every 1/4 time unit. This means that the smallest best horizon in our testing is 1/4.

The first experiment was conducted with a stream speed at 80 connections per time unit (i.e., there are 40 training stream points and 40 test stream points per time unit). We set the *buffer\_size* at 1600 points, which means upon receiving 1600 points (including both training and test stream points) we’ll use a small set of the training data points (In this case  $k_{fit}=80$ ) to choose the best horizon. We compared the accuracy of the *On-Demand-Stream classifier* with two simple one-pass stream classifiers over the entire data stream and the selected sliding window (i.e., sliding window  $H=8$ ). Figure 1 shows the accuracy comparison among the three algorithms. We can see the *On-Demand-Stream classifier* consistently beats the two simple one-pass classifiers. For example, at time unit 2000, the *On-Demand-Stream classifier*’s accuracy is about 4% higher than the classifier with fixed sliding window, and is about 2% higher than the classifier with the entire dataset. Because the class distribution of this dataset evolves significantly over time, either the entire dataset or a fixed sliding window may not always capture the underlying stream evolution nature. As a result, they always have a worse accuracy than the *On-Demand-Stream classifier* which always dynamically chooses the best horizon for classifying.

Figure 2 shows the distribution of the best horizons (They are the smallest ones if there exist several best horizons at the same time). Although about 78.4% of the (smallest) best horizons have a value 1/4, there do exist about 21.6% best horizons ranging from 1/2 to 32 (e.g., about 6.4% of the best horizons have a value 32). This also illustrates that there is no fixed sliding window that can achieve the best accuracy and the reason why the *On-Demand-Stream classifier* can outperform the simple one-pass classifiers over either the entire dataset or a fixed sliding window.



**Figure 1: Accuracy comparison (Network Intrusion dataset, stream\_speed=80, buffer\_size=1600,  $k_{fit}=80$ , *init\_number*=400)**

We have also generated one synthetic dataset B300kC5D20 to test the classification accuracy of these algorithms. This dataset contains 5 class labels and 300K data points with 20 dimensions. We first set the stream speed at 100 points per time unit. Figure 3 shows the accuracy comparison among the three algorithms: The *On-Demand-Stream classifier* always has much better accuracy than the other two classi-

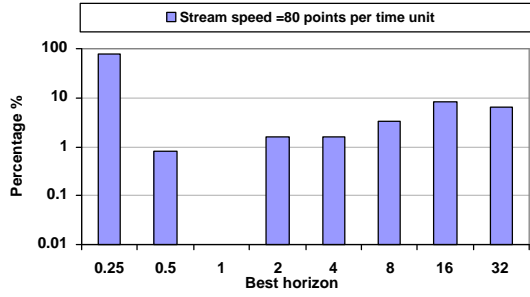


Figure 2: Distribution of the (smallest) best horizon (Network Intrusion dataset, Time units=2500, buffer\_size=1600,  $k_{fit}=80$ ,  $init\_number=400$ )

ifiers. Figure 4 shows the distribution of the (smallest) best horizons which can explain very well why the *On-Demand-Stream classifier* has better accuracy.

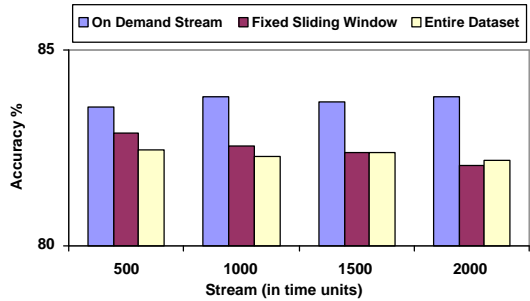


Figure 3: Accuracy comparison (Synthetic dataset B300kC5D20, stream\_speed=100, buffer\_size=500,  $k_{fit}=25$ ,  $init\_number=400$ )

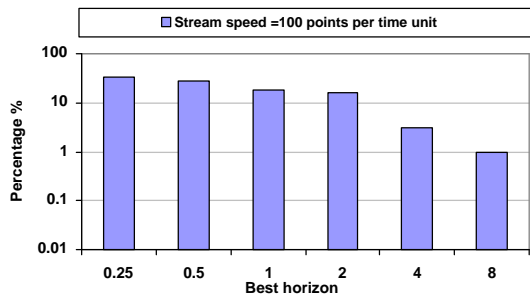


Figure 4: Distribution of the (smallest) best horizon (Synthetic dataset B300kC5D20, Time units=2000, buffer\_size=500,  $k_{fit}=25$ ,  $init\_number=400$ )

We used both the Network Intrusion dataset and one synthetic dataset B200kC30D30 to evaluate the processing rate of the *On-Demand-Stream classifier*. The synthetic dataset contains 200K points with 30 attributes and 30 class labels. Figure 5 demonstrates the result. The figure shows that the *On-Demand-Stream classifier* can classify about 600 test

points per second for the Network Intrusion data set, while it can process about 320 test points per second for synthetic dataset B200kC30D30. Also, we can see that the processing rate will become a little slower as time goes on, but at a later stage it will be very stable over time for both datasets.

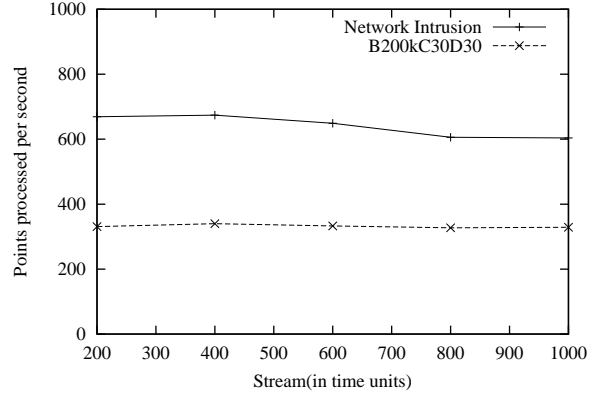


Figure 5: Processing rate

## 5. DISCUSSION AND CONCLUSIONS

In this paper, we presented a framework for classification of dynamic evolving data streams. While previous research has developed methods on the development of one-pass algorithms for data stream classification, this paper proposes a new framework and a different methodology for online classification and *continuous* adaption to the fast evolving data streams. The new framework has been designed carefully based on our analysis and reasoning and has been tested based on our experiments on real intrusion detection data sets. As evidenced by the empirical results, the system developed here is able to provide significantly better results than a static classification model on classification accuracy. In addition, it is efficient and scalable at handling large data streams.

## 6. REFERENCES

- [1] C. C. Aggarwal, J. Han, J. Wang, P. Yu. CluStream: A Framework for Clustering Evolving Data Streams. *VLDB Conference*, 2003.
- [2] C. C. Aggarwal. A Framework for Diagnosing Changes in Evolving Data Streams. *ACM SIGMOD Conference*, 2003.
- [3] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom. Models and Issues in Data Stream Systems, *ACM PODS Conference*, 2002.
- [4] P. Domingos, G. Hulten. Mining High-Speed Data Streams. *ACM SIGKDD Conference*, 2000.
- [5] R. Duda, P. Hart. *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
- [6] G. Hulten, L. Spencer, P. Domingos. Mining Time Changing Data Streams. *ACM KDD Conference*, 2001.
- [7] T. Zhang, R. Ramakrishnan, M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Conference*, 1996.