A Framework for On-Demand Classification of Evolving Data Streams

Charu C. Aggarwal, *Senior Member*, *IEEE*, Jiawei Han, *Senior Member*, *IEEE*,
Jianyong Wang, *Member*, *IEEE*, and Philip S. Yu, *Fellow*, *IEEE*

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 test 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.

Index Terms—Stream classification, geometric time frame, microclustering, nearest neighbor.

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. Discussions on recent advances in data stream mining may be found in [4], [5], [6], [8], [9], [15], [17], [19], [21].

One important data mining problem which has been studied in the context of data streams is that of classification [10], [11], [12]. The main thrust on data stream mining in the context of classification has been that of one-pass mining [8], [13], [16], [18], [23]. In reality, the nature of the underlying changes in the data stream can impose considerable challenges. Previous work on stream classification simply treats it as a one-pass mining problem. This does not treat the classification problem in the context of the underlying changes which have occurred in the stream. 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. This is generally quite important since data streams may evolve considerably over time [3]. While the work in [13] 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. This is because such a model implicitly uses a sliding window which is equal to the history of the entire 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. Therefore, a more temporally adaptive philosophy is desirable to improve the effectiveness of the underlying algorithms.

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 (labeled) training stream and the (unlabeled) 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. On the other hand, if the stream has been relatively stable over time, then using a longer history for training makes greater sense.

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 [8], [13], [23], these cannot be easily used in the context of an *on-demand classifier* in an evolving

- C.C. Aggarwal and P.S. Yu are with the IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532.
E-mail: {charu, psyu}@us.ibm.com.
- J. Han is with the University of Illinois at Urbana-Champaign, Urbana, IL 61801. E-mail: hanj@cs.uiuc.edu.
- J. Wang is with Tsinghua University, Beijing, China.
E-mail: jianyong@tsinghua.edu.cn.

Manuscript received 28 Mar. 2005; revised 28 Oct. 2005; accepted 19 Jan. 2006; published online 17 Mar. 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0114-0305.

1041-4347/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society

environment. This is because such a classifier requires rapid variation in the horizon selection process due to data stream evolution. In this respect, nearest-neighbor classifiers tend to be more amenable to quick horizon adjustments because of their simplicity. However, it is still 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 and adjustable statistical data collection in order to perform the corresponding operations in an efficient way.

In this paper, we will develop such an on-demand classifier. The on-demand classifier is designed by adapting the (unsupervised) microclustering model [2] to the classification problem. Since microclustering is a data summarization technique, some of the underlying concepts can be leveraged effectively for other problems, such as classification, which utilize the aggregate data behavior over different time horizons. In order to use such an approach for the classification problem, the following adaptations need to be made:

- The microclustering process needs to be supervised, since each microcluster belongs to a specific class. Therefore, the representation of the microclusters and the process of updating, merging, and deleting microclusters needs to be done in a class-specific way. The aim of microclustering is to test the class discrimination of different time horizons.
- A geometric time frame is used instead of the pyramidal time frame in order to store the supervised microclusters. We discuss the similarities and differences of these time frames, and also discuss the advantages of the geometric time frame.
- A testing phase needs to be designed in conjunction with the creation of the supervised microclusters. This testing phase needs to be sensitive to the evolution of the underlying data stream.
- Methods need to be designed to pick the optimum segment of the stream in order to effectively perform the classification process. This is because the evolution of the stream [3] significantly affects the behavior of the classification algorithm. For this purpose, the classification framework needs to divide the training stream into two parts which are discussed below.

The testing phase of the on-demand classifier is constructed by dividing the training stream into two parts:

- A portion which is used for class-specific statistical maintenance of microclusters.
- A portion which is used for testing the nature of the horizon which provides the best classification accuracy.

We will show that these two portions can be effectively integrated to build a classifier which can classify evolving trends in the data accurately without losing efficiency. Our results demonstrate that stream classification cannot be effectively performed by simply viewing it in the context of one-pass mining. The underlying classifier needs to adjust rapidly to the changes so that accurate classification results can be provided to the user on-demand. A recent work discusses methods for using ensemble classifiers [25] in order to improve the classification accuracy in an evolving data stream. However, this method only discusses the issue of optimizing the composition of the ensemble, whereas the individual classifiers are used in black box fashion. This paper is devoted to the more fundamental problem of developing a robust classifier in an evolving environment. Hence, this work is orthogonal to ensemble approaches and can, in fact, be combined with the technique discussed in [25].

This paper is organized as follows: The remainder of this section discusses the contributions of this work. The motivation and concepts behind the basic statistical constructs of the paper are discussed in the next section. These constructs adapt the unsupervised microclustering model discussed in [2] to the supervised case. 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.

1.1 Contributions of this Work

This paper adapts the unsupervised microclustering approach developed in [2] in order to make it work effectively for the classification problem in the context of highly evolving data streams. Recent papers have proposed the classification model in a data stream as a relatively straightforward extension of the traditional classification problem. The only difference is that one-pass mining is required in order to perform the training. In reality, the process of classification should be viewed as a continuous process in which the *training stream* and *test stream* are simultaneously generated by the underlying process. In addition, it is assumed that both the training and test streams are evolving over time. This assumption may be true in many monitoring scenarios in which the activities in the underlying data stream are followed by events which can be tracked in time. For example, in business activity monitoring applications, it may be possible to track various control variables as the underlying training stream and the events of significance as the test stream. The same is true of surveillance applications in which a large number of variables may be tracked in order to monitor events of significance. For applications in which manual labeling is required, this may be true if the class of interest occurs as a rare event. In such cases, the data stream may have a high volume, but the system may be augmented by periodic labeling when the rare events of interest do occur. The assumption of simultaneous test and training streams may not always be necessary when the entire training data is already available, as in the case of static databases. However, in applications in which the classification is used as a means to a rapid response mechanism, this assumption turns out to be very useful. Such applications are also referred to as *on-demand* applications.

Previous work, such as that discussed in [8], [13], [23], is quite effective for dynamic updating of classification models, but does not develop systematic methods for picking the optimal classification model from a temporal point of view. Such a model provides greater classification accuracy than the one that uses the entire history of the data stream or a particular sliding window.

2 Basic Constructs for Maintaining Classification Statistics

We note that the determination of the appropriate window of data depends upon the level of evolution of the data. This level of evolution is not known a priori and is difficult to determine efficiently for a fast stream. The high volume of the data stream makes it essential to store summary statistics in such a way that the determination of the appropriate horizons during the classification process can be performed effectively.

The moments in time at which the summary statistics are stored are organized in the form of a *geometric time frame*. The use of the geometric time frame provides the flexibility of the classification model. This is because the microclusters, which are stored at different moments in time, can be used to quickly construct a classification model over different time horizons. At each moment in time, the classification on the test stream is performed by using a horizon which suits the needs of that particular moment. While the work in [2] discusses a pyramidal time frame in the context of the clustering problem, the geometric time frame provides a more effective implementation of the storage process. We note that the concept of a geometric time frame is different from the pyramidal frame in terms of reducing the overlap among different snapshots of the data.

It is assumed that the training and test data streams each consist of a set of multidimensional 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 multidimensional 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 $\mathcal{C}_j$. We assume that the class_id of the class $\mathcal{C}_j$ is j.

We will first begin by defining the concept of *supervised* microclusters. While the microclustering concept of [2] is useful for unsupervised clustering, we need to make modifications in order to use this approach for the classification process. The supervised microclusters are created from the training data stream only. Each such microcluster corresponds to a set of points from the training data, all of which belong to the same class.

Definition 2.1. *A supervised microcluster 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 definitions of each of these entries are 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 pth 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 pth 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 microcluster.*

The above definition of the supervised microcluster for the set of points $\mathcal{C}$ is denoted by $\overline{CFT}(\mathcal{C})$. This summary information is an extension of the cluster feature vector concept discussed in [26]. Since each component in the definition of the microcluster is an additive sum over different data points, this data structure can be updated easily over different data streams. We note that the microclustering construct is primarily designed for the case of continuously defined attributes. In order to handle categorical data, a similar construct needs to be designed for such variables; a task which is beyond the scope of this paper.

The statistics of the microclusters and their distribution over different classes may change considerably 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 microclusters at different moments in time. These stored microcluster states are referred to as *snapshots*. Such snapshots are stored away (possibly on disk) at particular moments in time. At the same time, the current state of the microclusters is always maintained in main memory.

The snapshots need to be stored in such a way that a 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, the snapshots are stored at differing levels of granularity depending upon the recency. We note that the geometric time frame is similar in spirit to methods which require the storage of approximate statistics over windows. The problem of storing approximate statistics for querying has been studied extensively in the data stream literature [7], [19], [22], [20], [24]. Some of these methods [7] also propose techniques for designing time frames which can store the data over multiple horizons. Our proposed geometric time frame is unique in providing an excellent and adjustable tradeoff between storage requirements and accuracy.

Without loss of generality, we can assume that one unit of clock time is the smallest level of granularity. 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 the 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

TABLE 1
A Geometric Time Window

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 2
A Pyramidal Time Window

Frame no.	Snapshots (by clock time)
0	70 69 68
1	70 68 66
2	68 64 60
3	64 56 48
4	64 48 32
5	64 32

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 $h < t_c$ 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. Let r be the smallest integer such that $h < 2^{r+1}$. Since r is the smallest such integer, it also means that $h \geq 2^r$. This means that, for any interval $(t_c - h, t_c)$ of length h, at least one integer $t' \in (t_c - h, t_c)$ must exist which satisfies the property that $t' \bmod 2^{r-1} = 0$ and $t' \bmod 2^r \neq 0$. Let t' be the time stamp of the last (most current) such snapshot.

Then, if $max\_capacity$ is at least 2, the second last snapshot of order $(r-1)$ is also stored and has a timestamp value of $t' - 2^r$. Let us pick the time $t_s = t' - 2^r$. By substituting the value of t_s, we get:

$$t_c - t_s = (t_c - t' + 2^r). \qquad (1)$$

Since $(t_c - t') \geq 0$ and $2^r > h/2$, it easily follows from (1) that $t_c - t_s > h/2$.

Since t' is the position of the latest snapshot of frame $(r-1)$ occurring before the current time t_c, it follows that $(t_c - t') \leq 2^r$. Substituting this inequality in (1), we get $t_c - t_s \leq 2^r + 2^r \leq h + h = 2 \cdot h$. Thus, we have:

$$h/2 \leq t_c - t_s \leq 2 \cdot h.$$

□

The above result ensures that every possible horizon can be closely approximated within a modest level of accuracy. At the same time, the corresponding storage requirements are quite modest as well. For example, for a data stream running for 100 years with a clock time granularity of one second and value of $max\_capacity = 2$, the total number of snapshots which need to be maintained are given by $(2) \cdot \log_2(100 * 365 * 24 * 60 * 60) \approx 64$.

In Table 1, we present an example 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 $frame\_number\ i$ and 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 the 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, and 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.

2.1 Comparison with Pyramidal Time Frame

While the geometric time frame shares a number of conceptual similarities with the pyramidal time frame [2], 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 [2]. In order to explain this point, we will present the pyramidal time frame discussed in [2]. In the case of the pyramidal time frame, the snapshots are classified into different orders. The order of a set of snapshots define the level of granularity. As in the case of the geometric frame, we can assume without loss of generality that one unit of clock time is the smallest level of granularity. Then, we assume that the snapshots of the ith order are stored when the clock time is divisible by α^i. As in the previous case, at any moment in time, only the last $max\_capacity$ snapshots are stored. For example, let us consider the snapshots illustrated for the geometric time frame in Table 1. We constructed the pyramidal time frame using the same set of snapshots and current time as in Table 1. We have illustrated these snapshots in Table 2. We note that the orders of both snapshots are the same. However, the snapshots in Table 2 contain a list of doubly counted snapshots. Eliminating the doubly counted snapshots is unwieldy at implementation time in a fast data stream, since one always has to keep track of which snapshots in each order are maintained. At the same time, keeping the doubly counted snapshots is expensive from a storage point of view. For example, while Table 2 contains one more snapshot than that in Table 1, the accuracy of the horizon estimation process is much lower. In order to illustrate this point, we calculate the distance to the closest snapshot in each of Tables 1 and 2 at time stamp 70 for different values of the user horizon in Fig. 1. We have illustrated the user-specified horizon on the X-axis and the

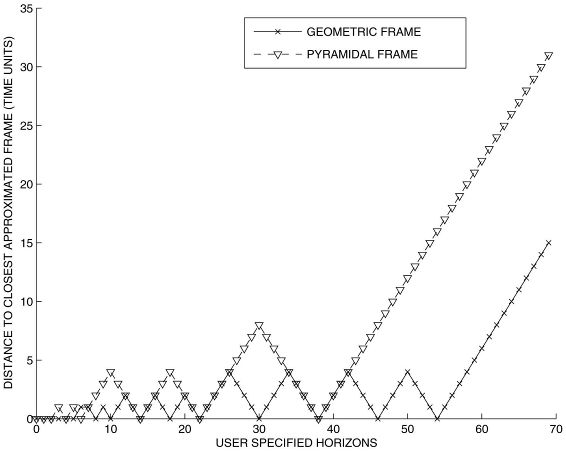

Fig. 1. Comparison between the pyramidal and geometric time window.

distance to the closest snapshot in time units on the Y-axis. It is clear that, in each case, the geometric time frame provides a closer distance to the nearest snapshot as opposed to the pyramidal time frame. In most cases, the accuracy of the pyramidal time frame was less than half the accuracy of the geometric time frame. In order for the pyramidal time frame to provide a competitive level of accuracy, the number of snapshots needs to be increased. This results in increased storage requirements. We note that, while the geometric and pyramidal time frame are conceptually similar, the geometric time frame has the advantage of greater elegance and ease of efficient implementation without double counting.

2.2 Online Algorithms for Supervised Microcluster Maintenance

In this section, we will discuss the process of online maintenance of microclusters and class statistics along with time. The microclusters and class statistics will be used in conjunction with a nearest-neighbor classification process in order to perform the final data stream classification.

Since the results of this paper are tailored toward 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 reemerge 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 percent of the data.
- The remaining majority of the stream is used for accumulation of the pertinent statistics corresponding to the microclusters and class information.

The process of maintenance of supervised microclusters 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 microclusters are maintained by the algorithm. We denote these microclusters by $\mathcal{M}_1 \dots \mathcal{M}_q$. Associated with each microcluster i, we create a unique id whenever it is first created. As we shall subsequently see, the microcluster maintenance algorithm requires a merging of multiple microclusters into one microcluster. Only microclusters that belong to the same class may be merged together during the clustering process. When two such microclusters are merged, a *list* of ids is created in order to identify the constituent microclusters. The value of q is determined by the amount of main memory available in order to store the microclusters. The microclusters which are maintained in main memory correspond to the current snapshot of summary statistics.

The first step of creating the initial microclusters 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 microclusters is created for each class by using a separate k-means algorithm on each class of data points in the initialization phase.

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

First, we attempt to place the data point in some preexisting microcluster. In order to do so, we find the distance of each data point to the microcluster 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 microcluster 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 streams which often exhibits itself in the form of a new microcluster. In order to decide whether a data point should belong to a preexisting microcluster 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 microcluster $\mathcal{M}_p$. If the data point $\overline{X_{i_k}}$ lies within this maximum boundary, then it is added to the microcluster $\mathcal{M}_p$ using the CF additivity property. The maximum boundary of the microcluster $\mathcal{M}_p$ is defined in terms of the average deviation of the other points from the centroid of this cluster. Specifically, the microcluster 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. We note that, in some cases, when the first two points are chosen identically, no new points will be added to the microcluster, and it eventually expires. This is a rare occurrence and, even when it does happen, it does not significantly affect the overall algorithm.

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

The microcluster statistics maintenance algorithm determines if it is desirable to delete any of the current microclusters. It is desirable to delete a microcluster 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 microcluster. This is estimated by using the mean and standard deviation of the last time stamps which have arrived in the current microcluster. The mean stamp of the microcluster 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 $m/(2 \cdot n)\mathrm{th}$ 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 microcluster. We determine if the least relevant time stamp is below a user-defined threshold δ. In such a case, it can be eliminated and a new microcluster 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 δ, two microclusters are merged in order to create space for the new microcluster. Since the microclusters are class-specific, we always merge microclusters belonging to the same class. In order to determine the microclusters to be merged, we find the pairwise distance between microclusters belonging to the same class. The closest pair of microclusters are merged together. The process of merging microclusters also requires some additional bookkeeping in terms of the ids associated with the individual microclusters. 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 microclusters after the merging process. This tracking is needed in order to create *horizon-specific microclusters* as will be discussed in the next section. As discussed later, the entire classification process is performed in a horizon-specific way.

In addition to the process of microcluster maintenance, we store the microclusters periodically in accordance with the geometric time frame. The microclusters at lower orders of the geometric time frame are dynamically maintained in main memory, whereas those at higher levels are written to disk. At each such time, we store away the current set of microclusters together with their id list and indexed by their time of storage. Since a maximum number of frames of each order are stored, some frames expire as time progresses. These frames are deleted.

3 Classification on Demand

In this section, we will discuss the *On-Demand Stream Classification Process*. In order to perform an 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 microclusters. 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 microclusters as discussed in the previous section.

Since the microclusters 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.

Property 3.1. *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two sets of points. Then, the cluster feature vector $\overline{CFT(\mathcal{C}_1 \cup \mathcal{C}_2)}$ is given by the sum of $\overline{CFT(\mathcal{C}_1)}$ and $\overline{CFT(\mathcal{C}_2)}$.*

This property can be directly extended to the following subtractive property:

Property 3.2. *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 $\overline{CFT(\mathcal{C}_1)} - \overline{CFT(\mathcal{C}_2)}$.*

The above property helps in approximate determination of the microclusters for a particular time horizon by using two snapshots. In order to do so, we use the idlists, which are stored with each microcluster. We denote the snapshot of microclusters 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 microclusters 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 microcluster in the current set $\mathcal{S}(t_c)$, we find the list of ids in each microcluster. For each of the lists of ids, we find the corresponding microclusters in $\mathcal{S}(t_c - h')$ and subtract the

CF vectors for the corresponding microclusters in $\mathcal{S}(t_c - h')$. The resulting set of microclusters corresponds to the time horizon $(t_c - h, t_c)$. We will denote this final set of microclusters created from the subtraction process by $\mathcal{N}(t_c, h')$.

Once the microclusters 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 prespecified 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 microcluster in $\mathcal{N}(t_c, h)$ to $\overline{X}$.
- We determine the class label of this microcluster 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 $\overline{X_t}$, 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 data sets. 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 data set 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. 3) The *On-Demand-Stream Classifier* is also rather space-efficient: It only needs to maintain a moderate number of class-labeled microclusters 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.

4.1 Test Environment and Data Sets

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 data set, we used the latest snapshot of microclusters to classify the test stream and, for the one-pass algorithm over a sliding-window, we always used a fixed horizon (e.g., eight time units) and computed the net snapshot of microclusters and used these net microclusters to classify the test stream data.

One real data set, the Network Intrusion detection data set, was used in our study to compare the classification accuracy of the *On-Demand-Stream classifier* with that of the simple one-pass stream classifier and test its sensitivity. Moreover, some synthetic data sets, generated by controlling the number of data points, the dimensionality, and the number of class labels, with different distribution or evolution characteristics, were used to test the classification accuracy, the processing rate (i.e., the number of test data points classified per second), and the scalability against the dimensionality and the number of class labels.

4.1.1 Real Data Set

We need to select some real data sets that evolve significantly over time in order to test the effectiveness of the *On-Demand-Stream classifier*. We have found a good choice: The KDD-CUP '99 Network Intrusion Detection stream data set which has been used earlier to evaluate several stream clustering algorithms. Since this data set was originally designed for testing classification algorithms and contains the class label for each point (here, each point is a connection), it is straightforward to use it to test the

TABLE 3
Default Parameter Settings

Parameter	value	meaning
microcluster-ratio	5	number_of_micro-clusters/number_of_natural_clusters
max_capacity	32	maximum number of snapshots for each frame number
$InitNumber$	400	number of initial points
t	2	maximum boundary factor
H	8	size of sliding window
δ	512	relevance stamp threshold
p	1	number of best time horizons used for classification
m	$0.32 \times n$	number of last arrivals

accuracy of the *On-Demand-Stream classifier*. This data set corresponds to the important problem of automatic and real-time detection of cyber attacks. This is a challenging problem for dynamic stream classification in its own right. The one-pass classifying algorithms may not be suitable to classify such intrusions from the normal connections.

The Network Intrusion Detection data set consists of a series of TCP connection records from two weeks of LAN network traffic managed by MIT Lincoln Labs. Each record can either correspond to a normal connection or an intrusion (or attack). The attacks fall into 22 types, such as buffer-overflow, guess-passwd, neptune, portsweep, rootkit, smurf, warezclient, spy, and so on. As a result, the data contains a total of 23 classes including the class for "*normal connections*." Most of the connections in this data set are *normal*, but, occasionally, there could be a burst of attacks at certain times. Also, each connection record in this data set contains 42 attributes, such as the duration of the connection, the number of data bytes transmitted from source to destination (and vice versa), percentile of connections that have "SYN" errors, the number of "root" accesses, etc. As in [5], one outlier point has been removed and all 34 continuous attributes were used for testing the classification accuracy. As we have pointed out, it is reasonable to assume that the training stream and test stream are generated by a similar underlying process. As a result, this data set is used in such a way: We divided it equally into the training stream and the test stream; each training connection is followed by a test connection and about 1 percent to 10 percent connections in the training stream were used to track the accuracy of all the geometric time horizons (i.e., horizons 1, 2, 4, 8, 16, ...).

4.1.2 Synthetic Data Sets

To test the processing rate and scalability of our classifier, we generated some synthetic data sets by varying base size from 100K to 1000K points, the number of class labels from 10 to 40, and the dimensionality in the range of 20 to 80. The data points of each class follow a Gaussian distribution, while different classes have different means varying from—5.0 to 5.0 and variances with a range from 0.5 to 1.5. At different time periods, 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 recomputed 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 willl use the following notations in naming the synthetic data sets: "B" indicates the base size, i.e., the number of data points in the data set, while "C" and "D" indicate the number of class labels and the dimensionality of each point, respectively. For example, for data set B200kC30D30, it contains totally 200K 30-dimensional data points and 30 class labels.

4.2 Accuracy Evaluation

We first evaluated the stream classification accuracy using the Network Intrusion Detection data set. Here, we define *microcluster-ratio* as the number of microclusters used in the *On-Demand-Stream classifier* divided by the number of natural class labels. Unless otherwise specified, the algorithm parameters were set at $microcluster\text{-}ratio = 5$, $max\_capacity = 32$, $InitNumber = 400$, maximum boundary factor $t = 2$, sliding window $H = 8$ time units, relevance stamp threshold $\delta = 512$, the number of time horizons used for classification $p = 1$,[1] and the number of last arrivals $m = 0.32 \times n$, where n is the number of points in the corresponding microcluster. In addition, in the following experiments, we store away the current snapshot of microclusters on disk every 1/4 time unit, which means the smallest possible horizon in our testing is 1/4. Table 3 summarizes the default parameter settings used in our experiments.

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 1,600 points, which means that, upon receiving 1,600 points (including both training and test stream points), we will 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 $H = 8$). Fig. 2 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 2,000, the *On-Demand-Stream classifier*'s accuracy is about 4 percent higher than the classifier with fixed sliding window and is about 2 percent

[1] For simplicity and efficiency, in the experiments, we always chose the time horizon which provides the greatest classification accuracy, and in the case that there exist multiple best time horizons, we always chose the smallest one among them for classification.

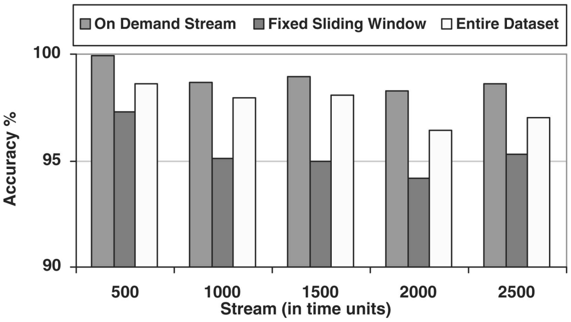

Fig. 2. Accuracy comparison (Network Intrusion data set, $\text{stream\_speed} = 80$, $\text{buffer\_size} = 1,600$, $k_{fit} = 80$, $InitNumber = 400$).

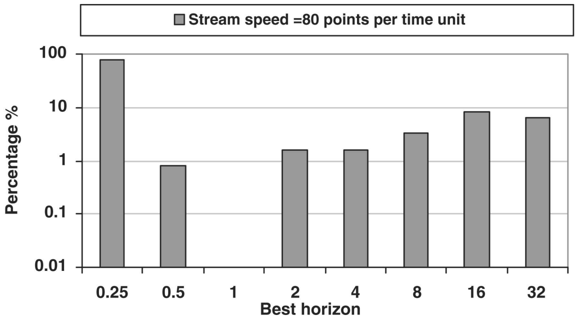

Fig. 3. Distribution of the (smallest) best horizon (Network Intrusion data set, $\text{Time units} = 2,500$, $\text{buffer\_size} = 1,600$, $k_{fit} = 80$, $InitNumber = 400$).

higher than the classifier with the entire data set. Because the class distribution of this data set evolves significantly over time, either the entire data set 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.

In Section 4, we stated that we use the horizon fitting stream segment to determine the classification accuracy of a certain horizon periodically. Fig. 3 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 percent of the (smallest) best horizons have a value 1/4, there do exist about 21.6 percent best horizons ranging from 1/2 to 32 (e.g., about 6.4 percent 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 beat the simple one-pass classifiers over either the entire data set or a fixed sliding window.

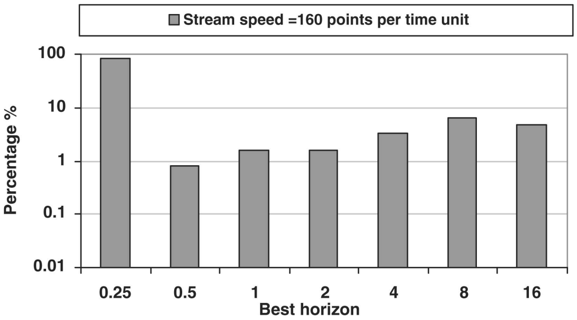

Fig. 5. Distribution of the (smallest) best horizon (Network Intrusion data set, $\text{Time units} = 1,250$, $\text{buffer\_size} = 1,600$, $K_{fit} = 80$, $InitNumber = 400$).

Similarly, we compared the accuracy of these three algorithms using the Network Intrusion data set at a series of different stream speeds ranging from 40 to 2,000 connections per time unit. These results all consistently show that the *On-Demand-Stream classifier* significantly beats the other two one-pass stream classifiers. Figs. 4 and 5 show the accuracy comparison among the three algorithms and the distribution of the (smallest) best horizons with stream_ speed=160, respectively. From Fig. 4, we can see that, on average, the accuracy of the *On-Demand-Stream classifier* can be 2 percent higher than the classifier with fixed sliding window and is about 3 percent higher than the classifier with the entire data set.

We have also generated one synthetic data set B300kC5D20 to test the classification accuracy of these algorithms. This data set contains five class labels and 300K data points with 20 dimensions. We first set the stream speed at 100 points per time unit and Fig. 6 shows the accuracy comparison among the three algorithms: The *On-Demand-Stream classifier* always has much better accuracy than the other two classifiers. Fig. 7 shows the distribution of the (smallest) best horizons, which can explain very well why the *On-Demand-Stream classifier* has better accuracy. We then set the stream speed at 200 points per time unit, Figs. 8 and 9 demonstrate the similar results.

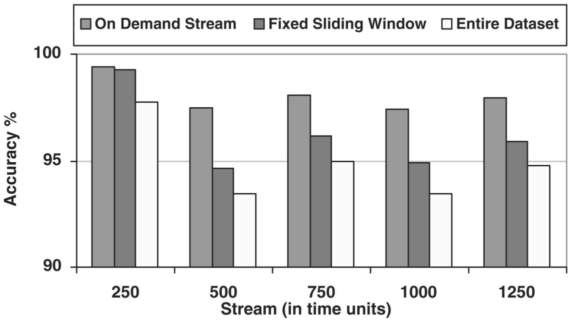

Fig. 4. Accuracy comparison (Network Intrusion data set, $\text{stream\_speed} = 160$, $\text{buffer\_size} = 1,600$, $k_{fit} = 80$, $InitNumber = 400$).

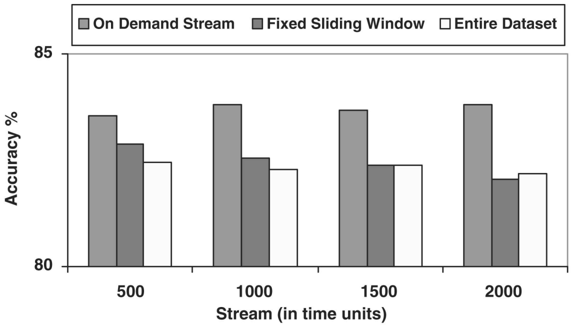

Fig. 6. Accuracy comparison (Synthetic data set B300kC5D20, $\text{stream\_speed} = 100$, $\text{buffer\_size} = 500$, $k_{fit} = 25$, and $InitNumber = 400$).

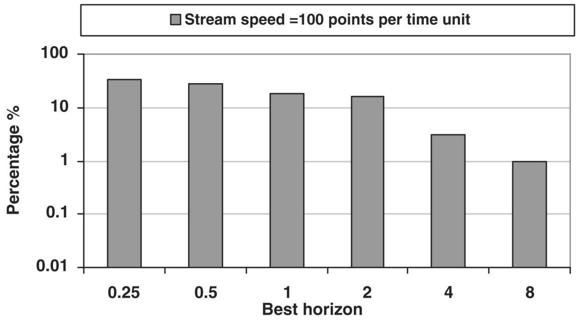

Fig. 7. Distribution of the (smallest) best horizon (Synthetic data set B300kC5D20, $\text{Time units} = 2,000$, $\text{buffer\_size} = 500$, $k_{fit} = 25$, and $InitNumber = 400$).

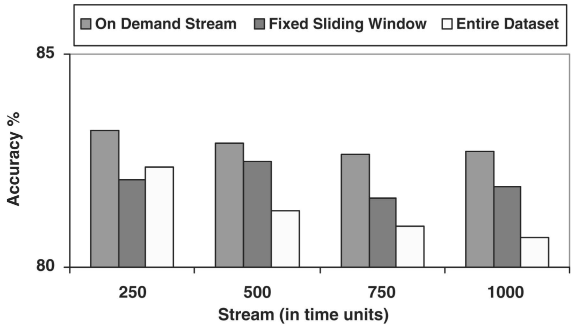

Fig. 8. Accuracy comparison (Synthetic data set B300kC5D20, $\text{stream\_speed}{=}200$, $\text{buffer\_size}{=}1,000$, $k_{fit}{=}50$, and $InitNumber{=}400$).

4.3 Efficiency and Scalability Test

In a stream environment, the data points keep arriving, a well-designed stream classifier should match the stream speed and scale to large dimensionality (i.e., the number of attributes) and the large number of class labels. In the following, we present our result related to the stream processing rate and the scalability in terms of the dimensionality and the number of class labels. In the following experiments, we fixed *microcluster-ratio* at 5, stream_speed at 200 points (among which, 100 points are test points) per time unit, $\text{buffer\_size} = 2,000$, $k_{fit} = 10$, and $InitNumber = 400$.

We first used both the Network Intrusion data set and one synthetic data set B200kC30D30 to evaluate the processing rate of the *On-Demand-Stream classifier*. The synthetic data set contains 200K points with 30 attributes and 30 class labels. Fig. 10 demonstrates the result. From this figure, we know, for the Network Intrusion data set, the *On-Demand-Stream classifier* can classify about 600 test points per second at different time while it can process about 320 test points per second for synthetic data set 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 data sets.

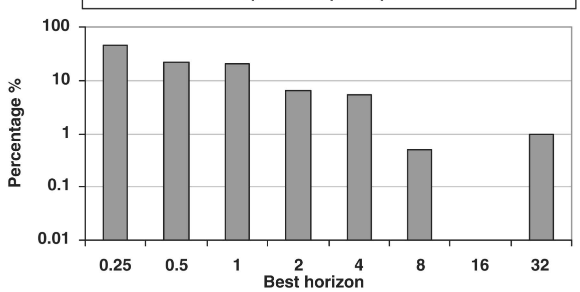

Fig. 9. Distribution of the (smallest) best horizon (Synthetic data set B300kC5D20, $\text{Time units} = 1,000$, $\text{buffer\_size} = 1,000$, $k_{fit} = 50$, and $InitNumber = 400$).

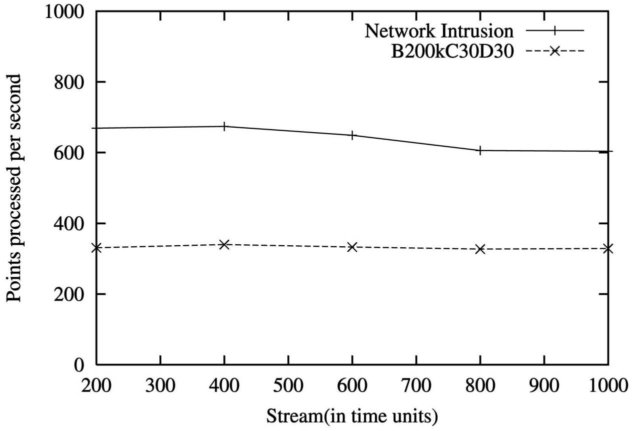

Fig. 10. Processing time.

We also generated a series of synthetic data sets to test the scalability of the *On-Demand-Stream classifier*. The first set of data sets were generated by fixing the number of points at 100K and the number of class labels at 10 and 20, respectively. The dimensionality has a range from 20 to 80. Fig. 11 shows that the *On-Demand-Stream classifier* has linear scalability in terms of the dimensionality. For example, for data set series B100kC20Dx, the runtime changes from 92 seconds to 220 seconds when the dimensionality increases from 20 to 80.

Fig. 12 shows the scalability result in terms of the number of class labels. In this experiment, we generated two series of data sets by fixing the dimensionality at 20 and 40, respectively, and each data set contains 100K points. By varying the number of class labels from 10 to 40 in these data sets, we tested the scalability of the *On-Demand-Stream classifier*. From Fig. 12, we see that it has a very good linear scalability, e.g., for data set series B100KCxD40, its runtime

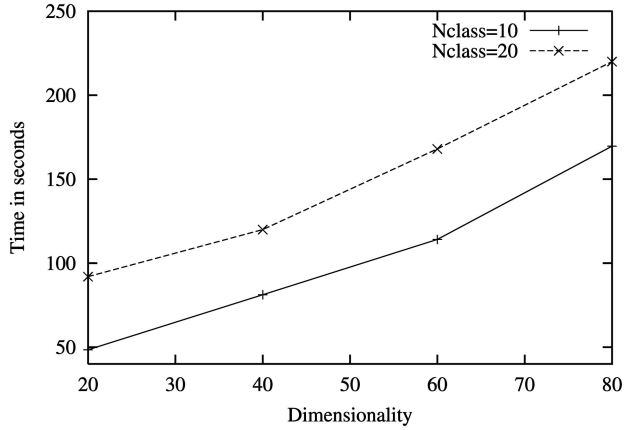

Fig. 11. Scalability with dimensionality.

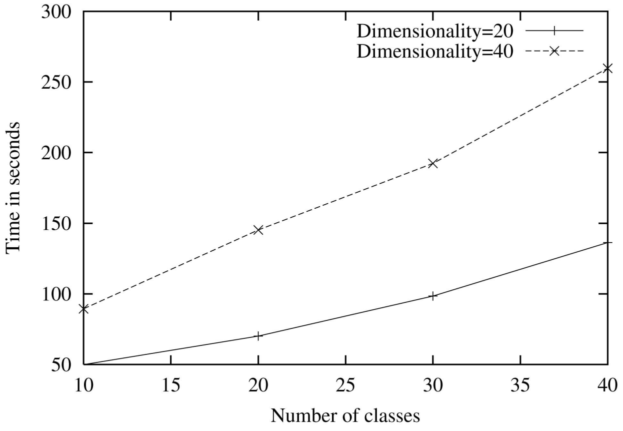

Fig. 12. Scalability with number of classes.

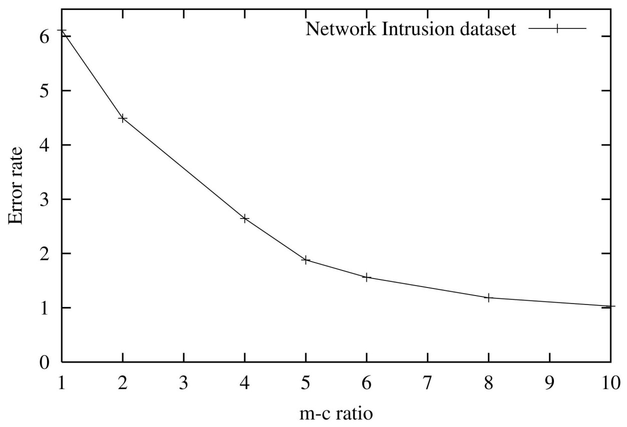

Fig. 14. Sensitivity analysis with MicroclusterRatio.

increases from 89 seconds to 259 seconds when the number of class labels changes from 10 to 40.

4.4 Sensitivity Analysis

As noted in Section 2, we assumed that at most *max_capacity* snapshots are stored for each frame number under the *geometric time frame*. We need to choose a proper value for *max_capacity* in order to achieve a high accuracy while consuming as little memory as possible. We used the Network Intrusion data set to test the sensitivity against *max_capacity*. In the experiment, we set $\text{stream\_speed} = 100$, $\text{buffer\_size} = 1,000$, $k_{fit} = 10$, $InitNumber = 400$, and $\text{total\_time\_units} = 2,000$. Fig. 13 shows the result. The y-axis is the error rate. When we vary *max_capacity* from 3 to 257, the error rate decreases only a little (about 0.1 percent), which means a moderate *max_capacity* can achieve a very high accuracy and we do not need to maintain too many snapshots for each frame number; as a result, the *On-Demand-Stream classifier* is very space efficient.

The *microcluster-ratio* parameter determines how many microclusters we should maintain for each snapshot. We also tested its impact on the classification accuracy. By using the Network Intrusion data set and varying the *microcluster-ratio* from 1 to 10, Fig. 14 shows that the *microcluster-ratio* does have a big impact on the accuracy: When the *microcluster-ratio* increases from 1 to 6, the error rate decreases from 6.1 percent to 1.6 percent, but when we continue to increase the ratio, the error rate decreases slowly. For example, when the *microcluster-ratio* value increases from 6 to 10, the error rate decreases less than 1 percent. This means a moderate value for *microcluster-ratio* is good enough to achieve a high accuracy and we do not need to store too many microclusters for each snapshot. This again illustrates from another angle the space efficiency of the *On-Demand-Stream classifier*.

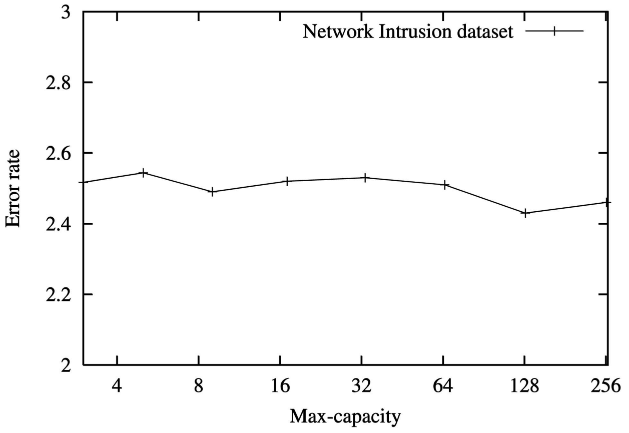

Fig. 13. Sensitivity analysis with Max_capacity.

As described in Section 3, the p time horizons which provide the greatest dynamic classification accuracy are selected for the classification of the stream. Thus, it is very helpful to evaluate the impact of different choices of parameter p. Table 4 shows the comparison results in terms of accuracy for the Network Intrusion data set. In the experiment, we set $\text{stream\_speed} = 100$, $\text{buffer\_size} = 400$, $k_{fit} = 40$, $InitNumber = 400$, and $\text{total\_time\_units} = 200$. We can see that the algorithm has a very similar classification accuracy with different choices of p. For simplicity and efficiency, by default, we set $p = 1$ for the algorithm.

5 Discussion and Conclusions

In this paper, we presented a framework for the classification of dynamic evolving data streams by adapting a previously developed (unsupervised) approach for microclustering [2]. 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 fast evolving data streams.

The stream classification framework proposed in this study has the following fundamental differences from the previous stream classification work in design philosophy.

First, due to the dynamic nature of evolving data streams, it may not be appropriate to either use a

TABLE 4
Sensitivity Analysis with p

Parameter p	1	2	4	6	8
Accuracy	99.74	99.87	99.87	99.88	99.87

predefined, fixed sliding window (no matter how carefully one selects the length of a sliding window) or use the "*entire*" data stream (no matter how carefully one defines the "entirety") to construct models for dynamic data streams. The selection of an *appropriate horizon* itself should be part of the stream classification process. Our experimental results on both real and synthetic data sets convincingly demonstrate this crucial point. Thus, different from static data classification, the classification of dynamic data streams should treat the temporal dimension as a special kind of the data and integrate it into the classification process.

Second, stream data classification needs to use temporal data and historical summary in its analysis. However, it is too costly to keep track of the entire history of data in uniform, fine granularity. Thus, a geometric time window, with more recent time in finer granularity and more remote time in coarser granularity, strikes a good balance. Previous studies on stream data clustering and stream time-series analysis present logarithmic window [4], natural pyramidal time window [7], and pyramidal time window [2]. Our design of geometric time window follows this trail but strikes a balance between logarithmic compression and sufficiently detailed coverage of recent events. Geometric time window gives us great flexibility at selection of the appropriate horizon in stream classification. It also provides good potential to carry out other powerful stream classification tasks, such as construction and comparison of models at different time frames, discovery of the evolution of models with time, and so on. This should be an interesting issue for further study.

Third, for dynamic model construction, compression should be performed not only along with time, but also on the data objects themselves due to the numerocity of the data. In order to achieve this goal, summary data is generated using microclustering in conjunction with a geometric time window. However, we keep reasonably more data than simple methods such as decision tree induction. This minor overhead leads to both good flexibility and high quality in classification, as shown by our experiments.

Finally, geometric time window and microclustering facilitate incremental update and maintenance of summary statistics and, thus, keep sufficient data statistics up-to-date with minor storage and processing overhead. Such online temporal, statistical data collection, and maintenance provides great flexibility for online stream classification in both *continuous query mode* (as a built-in watchdog) and *ad hoc query mode* upon user's mining request.

In summary, we have proposed an interesting framework for online classification of dynamically 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 a real intrusion detection data set. 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.

This new framework, based on our view, can be applied to many other stream data analysis tasks. The refinement and extension of this framework for other stream data analysis tasks is an interesting direction for future study.

Acknowledgments

The work of the second author was supported in part by the US National Science Foundation NSF IIS-03-8215/IIS-05-13678 and an IBM Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

[1] C.C. Aggarwal, J. Han, J. Wang, and P. Yu, "On Demand Classification of Data Streamsm," *Proc. ACM KDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 503-508, Aug. 2004.

[2] C.C. Aggarwal, J. Han, J. Wang, and P. Yu, "CluStream: A Framework for Clustering Evolving Data Streams," *Proc. Int'l Conf. Very Large Data Bases,* pp. 81-92, Sept. 2003.

[3] C.C. Aggarwal, "A Framework for Diagnosing Changes in Evolving Data Streams," *Proc. ACM SIGMOD Conf.,* pp. 575-586, June 2003.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems,* pp. 1-16, June 2002.

[5] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "Streaming-Data Algorithms For High-Quality Clustering," *Proc. 18th Int'l Conf. Data Eng.,* pp. 685-696, Feb. 2002.

[6] P. Bradley, U. Fayyad, and C. Reina, "Scaling Clustering Algorithms to Large Databases," *Proc. Knowledge Discovery and Data Mining Conf.,* pp. 9-15, 1998.

[7] Y. Chen, G. Dong, J. Han, B.W. Wah, and J. Wang, "Multi-Dimensional Regression Analysis of Time-Series Data Streams," *Proc. 28th Int'l Conf. Very Large Data Bases,* pp. 323-334, Aug. 2002.

[8] P. Domingos and G. Hulten, "Mining High-Speed Data Streams," *Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 71-80, Aug. 2000.

[9] P. Domingos and G. Hulten, "A General Method for Scaling Up Machine Learning Algorithms and Its Application to Clustering," *Proc. Int'l Conf. Machine Learning,* pp. 106-113, 2001.

[10] R. Duda and P. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[11] J.H. Friedman, "A Recursive Partitioning Decision Rule for Non-Parametric Classifiers," *IEEE Trans. Computers,* vol. 26, pp. 404-408, 1977.

[12] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh, "BOAT: Optimistic Decision Tree Construction," *Proc. 1999 ACM SIGMOD Int'l Conf. Management of Data,* pp. 169-180, June 1999.

[13] G. Hulten, L. Spencer, and P. Domingos, "Mining Time Changing Data Streams," *Proc. Seventh ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 97-106, Aug. 2001.

[14] F. Farnstrom, J. Lewis, and C. Elkan, "Scalability for Clustering Algorithms Revisited," *SIGKDD Explorations,* vol. 2, no. 1, pp. 51-57, 2000.

[15] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, "Testing and Spot-Checking of Data Streams," *Proc. 11th Ann. ACM-SIAM Symp. Discrete Algorithms,* pp. 165-174, Jan. 2000.

[16] F.J. Ferrer-Troyano, J.S. Aguilar-Ruiz, and J.C. Riquelme, "Discovering Decision Rules from Numerical Data Streams," *ACM Symp. Applied Computing,* pp. 649-653, 2004.

[17] J. Fong and M. Strauss, "An Approximate L^p-Difference Algorithm for Massive Data Streams," *Proc. 17th Ann. Symp. Theoretical Aspects of Computer Science,* pp. 193-204, Feb. 2000.

[18] J. Gama, R. Rocha, and P. Medas, "Accurate Decision Trees for Mining High-Speed Data Streams," *Proc. Ninth Int'l Conf. Knowledge Discovery and Data Mining,* pp. 523-528, Aug. 2003.

[19] J. Gehrke, F. Korn, and D. Srivastava, "On Computing Correlated Aggregates over Continual Data Streams," *Proc. 2001 ACM SIGMOD Int'l Conf. Management of Data,* pp. 271-282, May 2001.

[20] S. Guha and N. Koudas, "Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation," *Proc. 18th Int'l Conf. Data Eng.,* pp. 567-578, Feb. 2002.

[21] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering Data Streams," *Proc. 41st Annual Symp. Foundations of Computer Science,* pp. 359-366, Nov. 2000.

[22] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries," *Proc. 27th Int'l Conf. Very Large Data Bases,* pp. 79-88, Sept. 2001.

[23] R. Jin and G. Agrawal, "Efficient Decision Tree Construction on Streaming Data," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 571-576, Aug. 2003.

[24] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query Processing, Resource Management, and Approximation in a Data Stream Management System," *Proc. First Biennial Conf. Innovative Data Systems Research,* Jan. 2003.

[25] H. Wang, W. Fan, P. Yu, and J. Han, "Mining Concept-Drifting Data Streams Using Ensemble Classifiers," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 226-235, Aug. 2003.

[26] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," *Proc. 1996 ACM SIGMOD Int'l Conf. Management of Data,* pp. 103-114, June 1996.

Charu C. Aggarwal received the BTech degree in computer science from the Indian Institute of Technology in 1993 and the PhD degree from the Massachusetts Institute of Technology in 1996. He has been a research staff member at the IBM T.J. Watson Research Center since June 1996. He has applied for or been granted more than 55 US patents and has published more than 85 papers in numerous international conferences and journals. He has twice been been designated Master Inventor at IBM Research for the commercial value of his patents. His contributions to the Epispire project on real-time attack detection were awarded the IBM Corporate Award for Environmental Excellence in 2003. He has been a program chair of the DMKD 2003, chair for all workshops organized in conjunction with ACM KDD 2003, an associate editor of the *IEEE Transactions on Knowledge and Data Engineering*, and an action editor of *Data Mining and Knowledge Discovery*. He is a senior member of the IEEE. His current research interests include algorithms, data mining, privacy, and information retrieval.

Jiawei Han is a professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He has been working on research into data mining, data warehousing, stream data mining, spatiotemporal and multimedia data mining, biological data mining, social network analysis, text and Web mining, and software bug mining, with more than 300 conference and journal publications. He has chaired or served on many program committees of international conferences and workshops. He also served or is serving on the editorial boards for *Data Mining and Knowledge Discovery*, *IEEE Transactions on Knowledge and Data Engineering*, *Journal of Computer Science and Technology*, and *Journal of Intelligent Information Systems*. He is currently serving as founding editor-in-chief of the *ACM Transactions on Knowledge Discovery from Data* (*TKDD*) and on the board of directors for the executive committee of the ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD). He is an ACM Fellow and a senior member of the IEEE. He has received many awards and recognition, including ACM SIGKDD Innovation Award (2004) and IEEE Computer Society Technical Achievement Award (2005).

Jianyong Wang received the PhD degree in computer science in 1999 from the Institute of Computing Technology, the Chinese Academy of Sciences. Since then, he has worked as an assistant professor in the Department of Computer Science and Technology, Peking University, in the areas of distributed systems and Web search engines, and visited the School of Computing Science at Simon Fraser University, the Department of Computer Science at the University of Illinois at Urbana-Champaign, and the Digital Technology Center and Department of Computer Science and Engineering at the University of Minnesota, mainly working in the area of data mining. He is currently an associate professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is a member of the IEEE.

Philip S. Yu received the BS degree in electrical engineering from National Taiwan University, the MS and PhD degrees in electrical engineering from Stanford University, and the MBA degree from New York University. He is with the IBM Thomas J. Watson Research Center and is currently manager of the Software Tools and Techniques group. His research interests include data mining, Internet applications and technologies, database systems, multimedia systems, parallel and distributed processing, and performance modeling. Dr. Yu has published more than 450 papers in refereed journals and conferences. He holds or has applied for more than 250 US patents. Dr. Yu is a fellow of the ACM and a fellow of the IEEE. He is an associate editor of the *ACM Transactions on Internet Technology* and the *ACM Transactions on Knowledge Discovery in Data*. He is a member of the IEEE Data Engineering steering committee and is also on the steering committee of the IEEE Conference on Data Mining. He was the editor-in-chief of the *IEEE Transactions on Knowledge and Data Engineering* (2001-2004), an editor, advisory board member, and also a guest coeditor of the special issue on mining of databases. He had also served as an associate editor of *Knowledge and Information Systems*. In addition to serving as a program committee member on various conferences, he will be serving as the general chair of 2006 ACM Conference on Information and Knowledge Management and the program chair of the 2006 joint conferences of the Eighth IEEE Conference on E-Commerce Technology (CEC '06) and the Third IEEE Conference on Enterprise Computing, E-Commerce and E-Services (EEE '06). He was the program chair or cochair of the 11th IEEE International Conference on Data Engineering, the Sixth Pacific Area Conference on Knowledge Discovery and Data Mining, the Ninth ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, the Second IEEE International Workshop on Research Issues on Data Engineering: Transaction and Query Processing, the PAKDD Workshop on Knowledge Discovery from Advanced Databases, and the Second IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems. He served as the general chair of the 14th IEEE International Conference on Data Engineering and the general cochair of the Second IEEE International Conference on Data Mining. He has received several IBM honors, including two IBM Outstanding Innovation Awards, an Outstanding Technical Achievement Award, two Research Division Awards, and the 84th Plateau of Invention Achievement Awards. He received a Research Contributions Award from the IEEE International Conference on Data Mining in 2003 and also an IEEE Region 1 Award for "promoting and perpetuating numerous new electrical engineering concepts" in 1999. Dr. Yu is an IBM Master Inventor.