On Abnormality Detection in Spuriously Populated Data Streams

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

Abstract

In recent years, advances in hardware technology have made it increasingly easy to collect large amounts of multidimensional data in an automated way. Such databases continuously grow over time, and are referred to as data streams. The behavior of such streams is sensitive to the underlying events which create the stream. In many applications, it is useful to predict abnormal events in the stream in a fast and online fashion. This is often a difficult goal in a fast data stream because of the time criticality of the detection process. Furthermore, the rare events may often be embedded with other spurious abnormalities, which affect the stream in similar ways. Therefore, it is necessary to be able to distinguish between different kinds of events in order to create a credible detection system. This paper discusses a method for supervised abnormality detection from multi-dimensional data streams, so that high specificity of abnormality detection is achieved. We present empirical results illustrating the effectiveness of our method.

1 Introduction

In recent years, the advances in hardware technology have made it possible to collect large amounts of data in many applications. Typically, such databases are created by continuous activity over long periods of time, and are therefore databases which grow without limit. The volume of such transactions may easily range in the millions on a daily basis. Examples of such domains include supermarket data, multimedia data and telecommunication applications. The volume of such data is so large that it is often not possible to store it on disk in order to apply standard algorithmic techniques. Such data are referred to as *data streams.* Algorithms which are designed on such data need to take into account the fact that it is not possible to revisit any part of the voluminous data. Thus, only a single scan of the data is allowed during stream processing.

Considerable research has been done on the data stream problem in recent years [7, 8, 9, 13, 18]. Many traditional data mining problems such as clustering and classification have recently been re-examined in the context of the data stream environment [1, 10, 13]. In this paper, we discuss the problem of abnormality detection in data streams. In particular, we will consider the most difficult case, when such events are embedded with other similar but spurious patterns of abnormality. Some examples of applications of interest are as follows:

- In a stock market monitoring application, one may wish to find patterns in trading activity which are indicative of a possible stock market crash in an exchange. The stream of data available may correspond to the real time data available on the exchange. While a stock sell-off may be a relatively frequently occurring event which has similar effects on the stream, one may wish to have the ability to quickly distinguish the rare crash from a simple sell-off. Another example of a valuable event detection application is that of detection of particular patterns of trading activity which result in the sell-off of a particular stock, or a particular sector of stocks. A quick detection of such events is of great value to financial institutions.

- In a business activity monitoring application, it may be desirable to find particular aspects of the stream which are indicative of events of significance to the business activity. For example, certain sets of actions of competitor companies may point to the probably occurrence of significant events in the business. When such events do occur, it is important to be able to detect them very quickly, as they may be used to trigger other business-specific actions in a time-critical manner.

- In a medical application, continuous streams of data from hospitals or pharmacy stores can be used to detect any abnormal disease outbreaks or biological attacks. Certain kinds of diseases caused by biological attacks are often difficult to distinguish from other background diseases. For example, an anthrax infection has similar characteristics to the common flu attack. However, it is essential to be able to make such distinguishing judgements in real time in order to create a credible event detection system.

Many of the events discussed above are inherently rare. For example, events such as disease outbreaks or stock market crashes may happen rarely over long periods of time. On the other hand, the value of event detection is highly dependent on the latency of the detection process. This is because most event detection systems are usually coupled with time-critical response mechanisms. Furthermore, because of efficiency considerations, it is possible to examine a data point only once throughout the entire computation. This creates an additional constraint on how abnormality detection algorithms may be designed. While event detection and anomaly detection are important problems in the data mining community [4, 6, 15, 17], these models do not address the problem in the context of predicting rare anomalies in the presence of many spurious (but similar) patterns. In order to achieve the specificities in abnormality detection, we will utilize a supervised approach in which the abnormality detection process learns from the data stream. A considerable level of specificity may be achieved by using the behavior of the combination of *multiple streams* which are able to distinguish between different kinds of seemingly similar anomalies.

Thus, the additional complexities of the generic event detection problem may be summarized as follows:

- In most real-life situations, data streams may show abnormal behavior for a wide variety of reasons. It is important for an event detection model to be *specific* in its ability to focus and learn a rare event of a particular type. Furthermore, the spurious events may be significantly more frequent than the rare events of interest. Such a situation makes the event detection problem even more difficult, since it increases the probability of a false detection.

- In many cases, even though multiple kinds of anomalous events may have similar effects on the individual dimensions, the relevant event of interest may only be distinguished by its relative effect on these dimensions. Therefore, an event detection model needs to be able to quantify such differences.

- Since the data stream is likely to change over time, not all features remain equally important for the event detection process. While some features may be more valuable to the detection of an event in a given time period, this characteristic may vary with time. It is important to be able to modify the event detection model appropriately with the evolution of the stream.

- We note that a supervised abnormality detection problem is very different from a data stream classification problem in which each record has a label attached to it. In an abnormality detection problem, individual records are unlabeled, whereas the abnormalities of importance are attached only to particular moments in time. Since individual records do not have class labels, the training of the event detection process is more constrained from the limited information availability. Furthermore, the rarity of the abnormality adds an additional level of complexity to the detection process.

- Unlike a traditional data source, a stream is a continuous process which requires simultaneous model construction and event reporting. Therefore, it is necessary for the supervision process to work with whatever information is currently available, and continue to update the abnormality detection model as new events occur.

In this paper we will design an abnormality detection algorithm which can handle the afore-mentioned complexities. Furthermore, the algorithms discussed in this paper do not require any re-scanning of the data, and are therefore useful for very fast data streams.

This paper is organized as follows. In the remainder of this section, we formalize the contributions of this paper and discuss the notations. In the next section we will discuss the algorithm for event detection. We will discuss the empirical results in section 3. Finally, section 4 contains the conclusions and summary.

1.1 Contributions of this paper This paper presents an effective method for learning rare abnormalities from fast data streams. Since a data stream may contain many different kinds of abnormalities, it is necessary to be able to distinguish their characteristic behavior. Therefore, we propose a technique which is able to distinguish particular kinds of events by learning subtle differences in how different streams are affected by different abnormalities. The algorithm performs statistical analysis on multiple dimensions of the data stream in order to perform the detection. Since the technique is tailored for fast responses to continuous monitoring of processes, the entire framework of the algorithm is constructed to facilitate online event monitoring of data streams. Therefore, the process can detect the abnormalities with any amount of historical data, but the accuracy is likely to improve with progression of the stream, as more data is received.

1.2 Notations and Definitions We discretize the points in time at which the behavior of the stream is monitored as *ticks*. The time stamps associated with the ticks are denoted by $t(1), t(2), \ldots t(k)$. We distinguish between the ticks and time stamps, since the behavior may not necessarily be monitored at equal time intervals. It is assumed that the data points arrive only at one of these ticks or time stamps.

The total number of data streams is N, and the set of data points associated with the ith stream at tick k is denoted by $Y_i(k)$. The data points in the stream $Y_i(k)$ are denoted by $y_i(1), y_i(2), \ldots y_i(k)$. It is assumed that for each stream i, the data point $y_i(j)$ arrives at the time stamp $t(j)$. The entire feed of N streams at tick k is therefore denoted by $\mathcal{Y}(k) = \{Y_1(k) \ldots Y_N(k)\}$.

We assume that the time stamps at which the rare events occur in the data stream are denoted by $T(1) \ldots T(r)$. These events may either be the *primary* events that we wish to detect, or they may be spurious events in the stream. We will also refer to the spurious events as the *secondary* events. Associated with each event k at time $T(k)$, we maintain $flag(k)$ which is 1 only when this event is a primary event. In addition, we also maintain $Q(k)$, which is the time stamp of the last *reported* occurrence of any event. The value of $Q(k)$ is typically larger than the true time of event occurrence $T(k)$. This is because the value of $Q(k)$ refers to the event *report* time, whereas the value of $T(k)$ refers to the *occurrence* time. The last report time is typically larger than time of the actual event itself, since the external sources reporting the abnormality would need a lag to verify it. These external sources may use a variety of domain specific methods or simply personal observation to decide on the final verification of abnormality occurrence. It is assumed that the report of an event is an external input to the algorithm, and is available only after a reasonable lag after the actual occurrence of the event. Clearly, a *detection* algorithm is useful only if it can report events and abnormalities *before* they are independently reported and verified by external sources. Let us assume that $k(r)$ events have occurred till tick r. We denote the sequence $\{(Q(1), T(1), flag(1)) \ldots (Q(k(r)), T(k(r)), flag(k(r)))\}$ until tick r by the event vector $\mathcal{E}(r)$. We note that the length of this sequence depends upon the number of events which have transpired till tick r.

The abnormality detection algorithm outputs a set of time stamps $T^*(1) \ldots T^*(n)$ at which it has predicted the detection. A particular detection $T^*(i)$ is referred to as a true detection, when for some lag threshold *maxlag*, some time stamp $T(j)$ exists, such that $0 \leq T^*(i) - T(j) \leq maxlag$. Otherwise, the detection is referred to as a false positive. Clearly, there is a tradeoff between being able to make a larger number of true detections and the number of false alarms. If the algorithm outputs a larger number of detection time stamps in order to reduce the latency, it is likely to report a greater number of false positives and vice-versa. We will discuss more about this issue in a later section.

```
Algorithm StreamEvent(Initial Stream/Event History: ($\mathcal{Y}_h, \mathcal{E}_h$),
        Current Stream/Event Feeds: ($\mathcal{Y}(\cdot), \mathcal{E}(\cdot)$))
begin
  { Create the initial specificity model based on
    initial stream history available }
  $\mathcal{M}$ = LearnStream($\mathcal{Y}_h, \mathcal{E}_h$)
  $r = 1$;
  for each tick $r$ do
    begin
      $\mathcal{SZ}$ = ComputeStatisticalDeviations($\mathcal{Y}(r+1)$);
      $\mathcal{AL}$ = PredictEvent($\mathcal{SZ}$, $\mathcal{M}$);
      if any event has occurred on tick $k$ then
        $\mathcal{M}$ = LearnStream($\mathcal{Y}(k), \mathcal{E}(k)$)
      { This updating of the model by LearnStream is
          done as a (background) offline process }
    end
end
```

Figure 1: The Abnormality Monitoring Algorithm

2 The Abnormality Detection Model

The supervised abnormality detection algorithm continuously detects events utilizing the data from the history of previous event occurrences. In addition, a learning phase is triggered once after every *reported* occurrence of a primary or secondary event in order to update the model. As discussed earlier, the *reporting* of an abnormality is an independent external process and is not dependent upon the actual detection of an abnormality by the algorithm. In most practical applications, abnormalities are eventually detected and reported because of a variety of factors such as the actual practical consequences of the abnormality. These report times are often too late to be of practical use for event responses. However, they can always be used to improve the accuracy of the abnormality detection model when required.

The model is initialized at the beginning of the detection process. Both the process of initialization and updating are performed by the subroutine *LearnStream* of Figure 1. The learning phase is performed as a background offline process, whereas the abnormality detection phase is performed as an online process. Therefore, the abnormality detection phase is performed at each tick and it consists of two steps:

- Computation of abnormal statistical deviations at a given instant. This is performed by the *ComputeStatisticalDeviations* procedure of Figure 1.

- Computation of specificity of statistical deviations

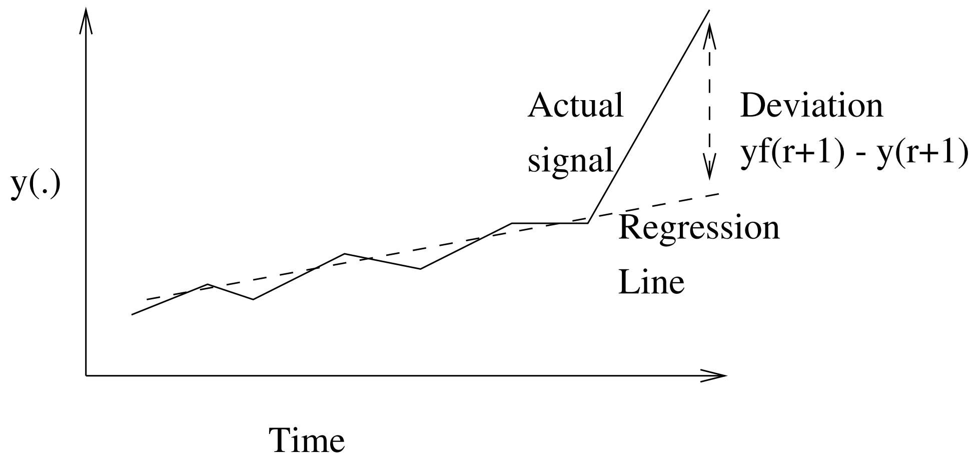

Figure 2: Deviations in data stream values

to occurrences of the primary event. This is performed by the *PredictEvent* procedure of Figure 1.

It is assumed that at the beginning of the stream monitoring process, some amount of historical data is available in order to construct an initial model of event behavior. This historical data consists of the stream and the events in the past time window at the beginning of the event monitoring process. The initial stream is denoted by $\mathcal{Y}_h$, and the initial set of events is denoted by $\mathcal{E}_h$. Once the event detector has been initialized, it then moves to an iterative phase of continuous online monitoring together with occasional offline updating.

Because of the speed of a data stream and the time-criticality of the event reporting process, it is essential that each of the above computations can be performed rapidly during real time. In the remainder of this section, we will describe each of these steps in some detail.

2.1 Detection of Rare Events in the Stream

The first step is to find deviations in the streams from the expected values based on the historical trends. The computation of the level of statistical deviations at a given instant is needed both for the alarm prediction phase as well as learning model. This process is denoted by *ComputeStatisticalDeviations* of Figure 1. For this purpose, we use a polynomial regression technique which can compute the statistically expected value of the data stream at a given moment in time.

Consider the tick r at which the points $y_i(1) \dots y_i(r)$ have already arrived for stream i. For each $l \in \{1, \dots r\}$, the technique approximates the data point $y_i(l)$ by a polynomial in $t(l)$ or order k. In other words, we approximate the data point $y_i(l)$, by the polynomial

```
**Algorithm** *ComputeStatisticalDeviations*(Stream: $\mathcal{Y}(r+1)$)
**begin**
  Determine the set of points in the past window $h_t$
     together with their weights;
  **for** each stream $i \in \{1, \dots N\}$ **do**
     **begin**
     Compute coefficients $a_{i0} \dots a_{ik}$;
     Compute $Err_i(r)$ and $yf_i(r+1)$ using polynomial
        function $f_{(\cdot,\cdot)}$;
     Compute $z_i(r+1) = (yf_i(r+1) - y_i(r+1))/Err_i(r)$;
     **end**
  **return**($z_1(r+1) \dots z_N(r+1)$);
**end**
```

Figure 3: Computation of Statistical Deviations in the Data Stream

function $f_i(k, l)$, where:

$$f_i(k,l) = \sum_{j=0}^{k} a_{ij}^l \cdot t(l)^j \qquad (2.1)$$

Here, the coefficients of the polynomial function are $a_{i1}^l \dots a_{ik}^l$. The values of a_{ij}^l need to be computed using the actual data points in order to find the best fit. Specifically, the data points within a maximum window history of h_t are used in order to compute the coefficients of this polynomial function. While these coefficients can be estimated quite simply by using a polynomial fitting technique technique, we note that not all points are equally important in characterizing this function. Those points which have arrived recently should be given greater importance in the computation. For this purpose, we use the exponential fading function $K \cdot e^{\lambda t(r)}$ in order to compute the importance of each data point. Here K and λ are constants which are specific to that particular time window. Thus, for each data point arriving at time t, it is replaced by $\lfloor K \cdot e^{\lambda t(r)} \rfloor$ data points while computing the polynomial function. Then a polynomial fitting technique is used in order to compute the coefficients. The value of λ is chosen such that the ratio of the weight of the data point at $t(r) - h_t$ to the data point arriving at $t(r)$ is given by *maxratio*. Therefore, we have:

$$(2.2) K \cdot e^{\lambda t(r)} / (K \cdot e^{\lambda (t(r) - h_t)}) = e^{\lambda \cdot h_t} = maxratio$$

The value of K is automatically chosen that the value of the fading function is equal to 1 at the time stamp $t(r) - h_t$. Therefore, once the value of λ has been set, the value of K is chosen using the following relationship:

$$(2.3) \qquad K \cdot e^{\lambda (t(r) - h_t)} = 1$$

The two intuitive parameters which are chosen by the user are the maximum window h_t and the maximum ratio *maxratio*. The choice of h_t depends upon the amount of memory buffer available. This is because all the data points in the past window of h_t need to be held in the memory buffer while processing the stream at a given tick. It is necessary to keep such a buffer because of the large volume of data points arriving in each unit of time. At each tick, when new data points enter the system, the set of points outside the window of h_t are discarded, and the new set of points arriving at the current tick are included. We also note that while the computation of the coefficients of the polynomial function requires a matrix inversion operation [14], the order of the matrix inverted is given by the maximum order of the polynomial function. For polynomials of small order up to 2 or 3, this computation can be performed in a small constant number of operations at each tick. This is a very small overhead compared to the processing of the points in the data stream.

Once the coefficients of the polynomial function $f_i(k, r)$ has been computed, we calculate the fitted values $yf_i(r - h_t) \dots yf_i(r)$ of the points in the data stream. (The value of $yf_i(r)$ is simply the instantiation of the function $f_i(k, r)$ at the tick r.) The average standard error of fitting is given by:

$$(2.4) \; Err_i(r) = \sqrt{\frac{\sum_{q=r-h_t}^{r} e^{\lambda(q)} (y_i(q) - yf_i(q))^2}{\sum_{q=r-h_t}^{r} e^{\lambda(q)}}}$$

The predicted value of the data point from stream i at the tick $(r+1)$ is given by $yf_i(r+1)$. Since, the predicted value $yf_i(r+1)$ is based on the behavior of the stream i up to tick r, the true value may vary considerably from the prediction when a rare event occurs. An example of such an occurrence is illustrated in Figure 2. We quantify this deviation at tick $(r + 1)$ by the corresponding z-number of the stream:

$$(2.5) \; z_i(r+1) = (yf_i(r+1) - y_i(r+1))/Err_i(r)$$

The z-number is equal to the number of standard deviations by which the true value of the stream i at tick $(r+1)$ is greater than the expected value. A high absolute magnitude of the z-number indicates significant statistical deviation from expected behavior of the stream. The process *ComputeStatisticalDeviations* of Figure 1 outputs $\mathcal{SZ} = (z_1(r+1) \dots z_N(r+1))$ which are the statistical deviations from the predicted values in each data stream. A high absolute value of these statistical deviations is indicative of the occurrence of a rare event in the streams. However, such an event could either correspond to the primary event or a secondary event. The event detector needs to distinguish between the two situations using the property that different kinds of events have a different signature as to the amount by which the events affect the different streams. The exact signature of a particular kind of event needs to be learned from the previous history of event occurrences. We will discuss this issue in the next subsection.

2.2 Learning Specific Events from the Data Stream We note that a given data stream may have different kinds of events which have different effects on the various components. For example, consider a biological attack application in which we are tracking the number of people admitted to emergency rooms with flu like symptoms. Let us also assume that we have different data streams corresponding to adults and children. While the early phase of a large anthrax attack may be indistinguishable from a flu epidemic, the data stream corresponding to children and adults admitted is likely to be different. For both anthrax attacks and flu epidemics, the z-numbers of both streams are likely to rise. However, the z-number of the stream for children admitted is likely to be affected to a greater extent in the case of a flu epidemic, while both streams are likely to be almost equally affected in the case of an anthrax attack. How do we magnify these subtle differences in the extent to which the different curves are affected?

In order to achieve this we use the data from previous event occurrences in order to create a distinguishing model for the particular kind of event which is being tracked. This model for distinguishing different kinds of events needs modeling which is done offline. However, this modeling needs to be done only in the following cases: (1) At the very beginning of the stream monitoring process as an initialization step. It is assumed that an initial amount of event history $\mathcal{E}_h$ and $\mathcal{Y}_h$ is available for this purpose. (2) At the occurrence of each kind of primary event as an updating step. In this case,

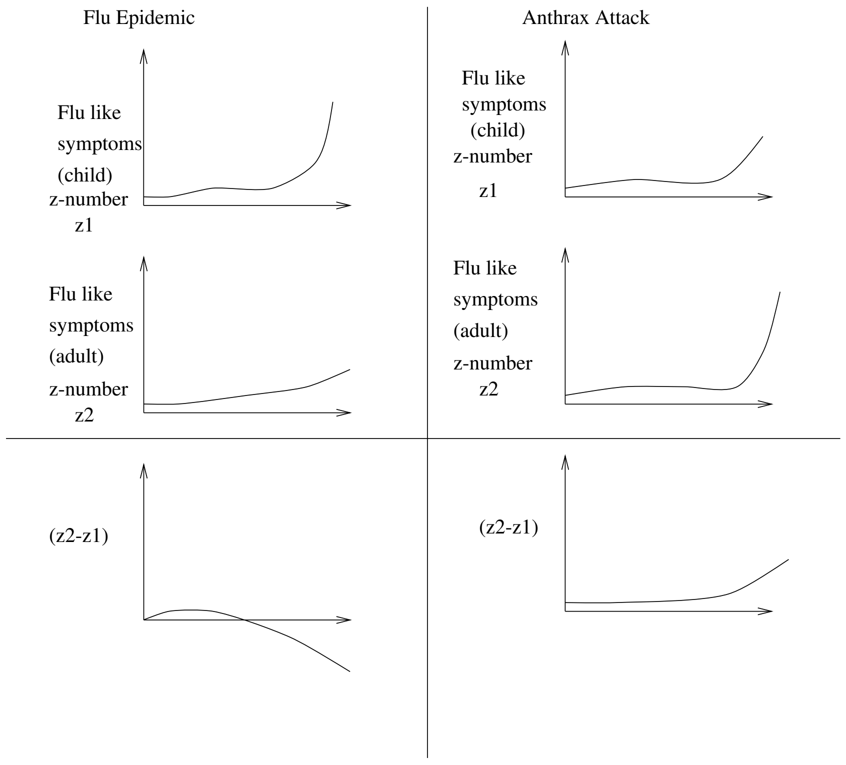

Figure 4: Distinguishing between Primary and Secondary Abnormalities

```
Algorithm LearnStream(Stream History: $\mathcal{Y}(\cdot)$, Event History: $\mathcal{E}(\cdot)$)
begin
  { Let $T(1) \ldots T(s)$} be the time-stamps at
  which the primary events have occurred }
  for each time stamp $T(j) \in \{T(1) \ldots T(s)\}$
    and stream $i$ find the largest value $max_{ij}$ for the
    $z$-number of stream $i$ in the interval $(T(j), T(j) + maxlag)$
  Let $\mathcal{S}$ be the set of streams such that $max_{ij}$ is greater
  than a pre-defined threshold $z_{min}$ for each $j \in \{1 \ldots s\}$;
  { Assume that the ticks at which primary events
   have occurred are denoted by $j_1 \ldots j_s$ and the
   time stamps of secondary events are $i_1 \ldots i_l$; }
  for each tick $j_k$ in $\{j_1 \ldots j_s\}$ and stream $i$
   find the tick $tp_i^*(k)$ such that $tp_i^*(k) \in \{t(j_k), t(j_k) + maxlag\}$
     and $z_i(\cdot)$ attains its maximum value in this time interval;
  $Tp^*(k) = \sum_{i \in \mathcal{S}} tp_i^*(k)/|\mathcal{S}|$;
  for each tick $i_k$ in $\{i_1 \ldots i_l\}$ and stream $j$
   find the tick $ts_j^*(k)$ such that $ts_j^*(k) \in \{t(i_k), t(i_k) + maxlag\}$
     and $z_j(\cdot)$ attains its maximum value in this time interval;
  $Ts^*(k) = \sum_{i \in \mathcal{S}} ts_j^*(k)/|\mathcal{S}|$;
  Define the algebraic expression $Z^s(\alpha) = \sum_{k=1}^{l} \sum_{i \in \mathcal{S}} \alpha_i \cdot z_i(Ts^*(k))/l$
  Define the algebraic expression $Z^p(\alpha) = \sum_{k=1}^{s} \sum_{i \in \mathcal{S}} \alpha_i \cdot z_i(Tp^*(k))/s$
  Find the value of the vector $\alpha$ which maximizes
    $Z^p(\alpha) - Z^s(\alpha)$ subject to the constraints
    $||\alpha|| = 1$ and $\alpha_i = 0$ for $i \notin \mathcal{S}$
  { The gradient descent method is used for the maximization }
  return($\alpha$);
  end
```

Figure 5: Learning Specific Events from the Data Stream

the model uses the effects of the event on the stream in order to update the current model.

While the *LearnStream* procedure is triggered by the occurrence of a primary event, the iterative event monitoring process continues in the foreground. Therefore, if another primary event occurs before the *LearnStream* procedure has finished execution, the slightly stale model is always available for the detection process.

We note that in many cases, even when streams are similarly affected by different kinds of events, the relative magnitudes of different streams could vary considerably. Our aim is to create a function of the z-numbers of the different streams which is a "signature" of that particular kind of event. In order to achieve this goal, we create a new signal at each tick which is a linear combination of the signals from the different streams. Let $\alpha_1 \dots \alpha_N$ be N real coefficients. We define the following new signal $Z(r)$ in terms of the original signal and the α values:

$$Z(r) = \sum_{i=1}^{N} \alpha_i \cdot z_i(r) \tag{2.6}$$

For example, in the case illustrated in Figure 4, the signals for the two streams corresponding to adult and children admission to hospitals are illustrated. The signal for the adult stream is denoted by $z2$ and the signal for the child stream is denoted by $z1$. A choice of $(\alpha_1, \alpha_2) = (-1, 1)$ creates the composite signal $z2 - z1$. In Figure 4(a), the different signals for the case of a flu epidemic are illustrated, whereas in the case of Figure 4(b) the signals for an anthrax attack are illustrated. It is clear that even though both streams are affected in a similar way by either a flu epidemic or an anthrax attack, the proportional effects are different. These effects can be magnified by the use of the composite signal $z2 - z1$, which is affected in a clearly different way in the two cases.

We note that many of the data streams may be noisy and will not have any correlation with the primary event. Such streams need to be discarded from the event distinguishing process. In other words, the corresponding values of α_i need to be set to zero. The first step is to identify such streams. For each of the time stamps $T(j) \in \{T(1) \dots T(s)\}$ at which an event of interest has occurred, we find the largest value[1] max_{ij} of $z_i(r)$ for each r such that $T(j) \leq t(r) \leq T(j) + maxlag$. A stream i is said to be interesting to the event detector, when for each $j \in \{1 \dots s\}$ the value of max_{ij} is larger than a pre-defined threshold z_{min}.[2] We denote this subset of streams $\{i_1 \dots i_w\} \in \{1, \dots N\}$ by $\mathcal{S}$.

Once we have selected a small number of streams which are meaningful for the event detection process, we need to find the value of the *discrimination vector* α which distinguishes the primary events from other similar events. The main idea is to choose α in such a way so that the value of $Z(r)$ peaks just after the occurrence of each primary event to a much greater extent that any other event. Let us assume that the time stamps at which all secondary events which have happened within the previous history of h_t, are given by $t(i_1) \dots t(i_l)$, whereas the time stamps of the primary event are given by $\{T(1) \dots T(s)\} = \{t(j_1) \dots t(j_s)\}$. For each secondary event i_k and each stream j, we compute the maximum value of $z_j(r)$ for each value of r, such that $t(r) \in (t(i_k), t(i_k) + maxlag)$. Let the corresponding time stamp be given by $ts_j^*(k)$ for each $k \in \{1 \dots l\}$. This time stamp is then averaged over all streams which lie in $\mathcal{S}$. Therefore, for each secondary event k, we compute $Ts^*(k) = \sum_{i \in \mathcal{S}} ts_j^*(k)/|\mathcal{S}|$. Similarly, for each occurrence of the primary event, we can compute the average time stamp $Tp^*(k)$ for each $k \in \{1, \dots s\}$. In order for the discrimination between primary and secondary events to be as high as possible, the difference in the average value of the composite signal at the time stamps of the true and spurious events must be maximized. The average composite signal $Z^s(\alpha)$ at the time of occurrences of the true events is given by the following:

$$Z^s(\alpha) = \sum_{k=1}^{l} \sum_{i \in \mathcal{S}} \alpha_i \cdot z_i(Ts^*(k))/l \tag{2.7}$$

Similarly, the average composite signal at the time of occurrence of the primary events is given by the following expression:

$$Z^p(\alpha) = \sum_{k=1}^{s} \sum_{i \in \mathcal{S}} \alpha_i \cdot z_i(Tp^*(k))/s \tag{2.8}$$

For maximum discrimination between true and spurious occurrences of primary events, we must choose α in such a way that the difference $Z^p(\alpha) - Z^s(\alpha)$ is maximized. Therefore, we have:

Maximize $Z^p(\alpha) - Z^s(\alpha)$
subject to:
$\alpha_j = 0 \quad j \in \{i_1 \dots i_w\}$
$||\alpha|| = 1$

[1]Strictly speaking, the value of max_{ij} should be based on the absolute value of the z-numbers. However, the above definition does not lose generality. For those streams in which events of interest correspond to highly negative z-numbers, the sign of the stream is flipped.

[2]For a normal distribution, more than 99.9% of the points are located within 3 standard deviations from the mean.

```
Algorithm PredictEvent(Statistical Deviations: $\mathcal{SZ}$,
  Learned Data:$\mathcal{M}$)
begin
{ The learned data $\mathcal{M}$ consists of the
  vector $\alpha$. The deviations $\mathcal{SZ}$
  consist of the statistical deviations at tick $(r+1)$ }
  $ZP(r+1) = \sum_{i=1}^{N} \alpha_i \cdot z_i(r+1)$
  Output event detection signature $ZP(r+1)$;
end
```

Figure 6: Rare Event Prediction

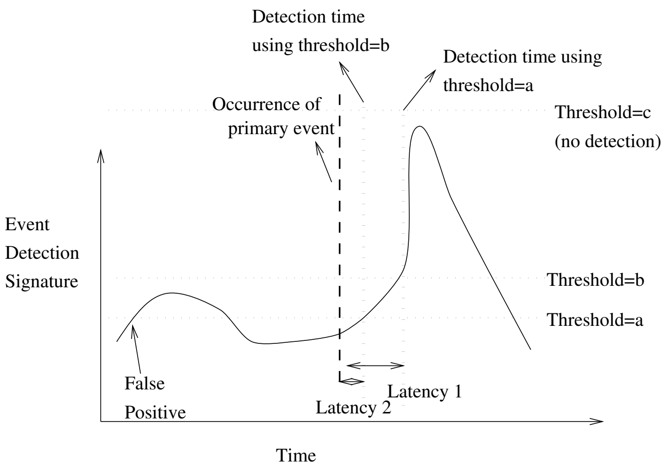

Figure 7: Illustrating the trade-off between Latency and False Positives

The first set of constraints corresponds to the fact that only the subset of streams which were determined to be significant to the event detection process are used for detection. The second constraint on α is required simply for the purpose of scaling as a boundary condition to the maximization problem. (Without the boundary condition, the maximization problem has an infinite solution.) We note that the objective function is linear in α, and all constraints are either linear or convex. Therefore, the optimum value of α can be found using a simple iterative gradient descent method which is discussed in [3].

2.3 Rare Event Prediction Process The event prediction is done by the procedure *PredictEvent* using the statistical information collected by the other procedures. The event prediction process uses the statistical deviations calculated by *ComputeStatisticalDeviations*, and the combination vector α computed by the *LearnStream* procedure. Let $\mathcal{SZ} = (z_1(r+1)\dots z_N(r+1))$ be the statistical deviations returned by *ComputeStatisticalDeviations* and α be the combination vector computed by *LearnStream.* The *PredictEvent* procedure then computes the value $ZP(r+1)$ which is defined as follows:

$$ZP(r+1) = \sum_{i \in \mathcal{S}} \alpha_i \cdot z_i(r+1) \tag{2.9}$$

This value is the signal which is specific to the primary event. The greater this value, the higher the likelihood that a primary event has indeed occurred in the stream. A primary event is predicted by using a minimum threshold on the value of $ZP(r+1)$. Whenever the value of $ZP(r+1)$ exceeds this threshold, a discrete signal is output which indicates that the event has indeed occurred. The use of higher thresholds on the event detection signature results in lower number of false positives, but lower detection rates as well as higher lags. For example, in Figure 7, we have illustrated the variation in latency level with time. We have illustrated two different thresholds on the event detection signature. It is clear that with the use of the lower threshold value of a, at least one false positive is created, which does not appear with the use of the higher threshold value of b. On the other hand, when the event does occur, it is detected much later with the use of the higher threshold. Therefore, the latency of detection is also much higher, when the threshold value of b is used. If the threshold level were increased even further to a value of c as indicated in Figure 7, then the algorithm misses detection completely. We will explore the effects of these trade-offs on a number of data sets in the empirical section. For the purpose of the algorithmic description of Figure 6, we output the value of $ZP(r+1)$ as the event detection indicator.

3 Empirical Results

In this section, we will illustrate the effectiveness of the event detection system. Since many of the application-specific methods such as those in [15] were designed with the assumption of a single channel (stream) of data for classification purposes, it is difficult to make a direct comparison with any previous algorithm. Furthermore, in many cases, the algorithms were not designed to handle the computational issues arising in the context of handling the massive volumes of a fast, spuriously populated data stream. We found however, that the anomaly detection method of [15] could be adapted to the data stream environment in a relatively straightforward way. The algorithm in [15] uses a nearest neighbor classification on the previous event history in order to output the detection of an event. By calculating the distance using multiple features, the technique could be applied to the more general data stream problem discussed in this paper. We will refer to this technique as the CL detector in the empirical results.

In order to test the algorithm, we used a number of data sets. The synthetic data sets were created by first generating each base stream j from a normal distribution with mean μ_j and standard deviation $\sigma_j = (1/3) \cdot \mu_j$. The value of μ_j for each stream is generated from a uniform distribution in the range $[0, 1]$. The events of significance are assumed to occur at w randomly distributed times $t_1 \dots t_w$ throughout the distribution of the data stream. We assume that there are f different types of events. Only one of these f different types of events corresponds to the primary event and the remaining events correspond to the secondary event. It is assumed that the occurrence of any event is equally likely to be of any type. Let us now consider a particular event at time $t_r \in \{t_1 \dots t_w\}$ which is of type $l \in \{1 \dots f\}$. The event of type l results in addition of a further signal to each data stream j. This signal is normally distributed with a mean of $\phi_{lj}(t - t_r)$ and a standard deviation of $\psi_{lj}(t - t_r)$. The value of $\phi_{lj}(s)$ was chosen to be quadratic function of s with a maxima at $s = \theta_{lj}$ and a maximum value of b_{lj}. The values of θ_{lj} and b_{lj} are chosen depending upon the event type and data stream. For each event type l and stream j, the value of θ_{lj} and b_{lj} are chosen from a uniform data distribution with ranges $[5, 50]$ and $[0, 1]$ respectively. The value of $\psi_{lj}(s)$ at each point was chosen to be $(1/3) \cdot \phi_{lj}(s)$. A value of $f = 5$ was used in each case. Two 10-dimensional synthetic data sets (for different random seeds) were generated using this methodology and are referred to as SStream-A and SStream-B respectively. In addition, two 20-dimensional data sets were generated with the same methodology. These are referred to as SStream-C and SStream-D respectively.

An interesting question which arises in the context of an empirical evaluation is that of choice of appropriate metrics. This is because a trade-off exists between the latency time and the number of false positives. Therefore, if the latency time of two algorithms is compared, it needs to be evaluated for the same number of false positives, and vice-versa. For this purpose, we need to provide a measure of the benefit that a true detection provides for a given level of detection latency. A more precise way of defining this would be with the use of a *benefit function.* While the exact function depends heavily on the application at hand, we designed a function which seems to be a reasonably intuitive metric for a variety of applications. We designated a maximum period of time latency δ_z after which detection provided no benefit. This is because anomalies of importance are often discovered in most applications through external factors such as simple human observation. There is no benefit to a detection, when the detection latency is

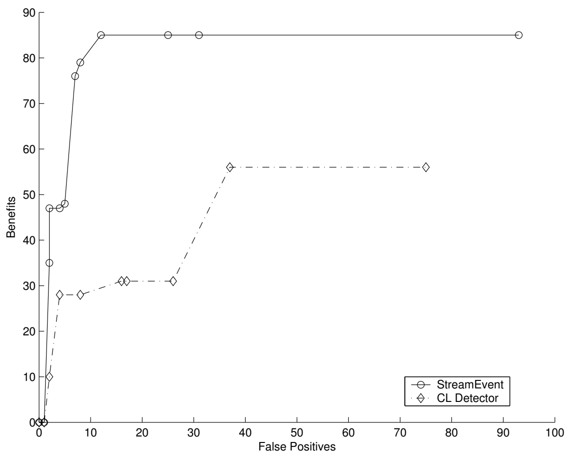

Figure 8: False Positives versus Benefits of Quick Detection (SStream-A)

larger than such external latencies. Thus, when the latency of detection is larger than δ_z, the benefit function is set to zero. For latencies less than δ_z, a linear function was used in order to characterize the benefit. Therefore, for a latency of t, the the benefit function $B(t)$ is defined as follows:

$$B(t) = \max\{0, \delta_z - t\} \tag{3.10}$$

We note that in this case, t is the latency of *first* detection of the anomaly after its actual occurrence. Similarly, the number of false positives is defined as the number of times an alarm is raised outside the δ_z interval after the true occurrence of any anomaly. The value of δ_z was chosen to be 100 in each case.

The detector proposed in this paper outputs an analogue signature level which is used for the prediction process. This analog signature can easily be converted into a binary decision by using thresholding on the intensity of the signature. When the signature level exceeds a given threshold, the detector outputs an anomaly detection indicator. The higher the threshold, the greater the latency[3] but the greater number of false positives. Therefore, higher number of false positives correspond to lower values of the benefit function and and vice-versa. In order to quantify this relationship, we create a corresponding AMOC curve which plots the false positives on the X-axis and the benefits on the Y-axis. Since the CL detector also uses thresholding on a similarity measure for the classification process [15], it is also possible to create AMOC curves for that case by varying this threshold.

[3]We note that in many cases, higher values of the threshold result in the detector completely missing the anomaly detection. This corresponds to infinite latency, and therefore zero benefit.

Data Set	Offline Update Time	Maximum Throughput (online)
SStream-A	3 seconds	2881 per second
SStream-B	4 seconds	2755 per second
SStream-C	3 seconds	1829 per second
SStream-D	3 seconds	1954 per second

Table 1: Time Requirements of the Event Detector

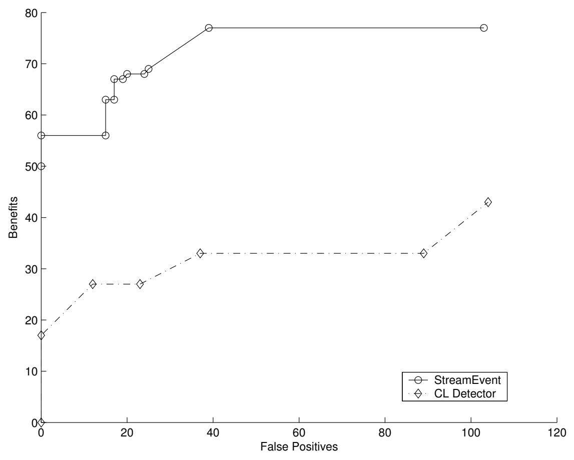

Figure 9: False Positives versus Benefits of Quick Detection (SStream-B)

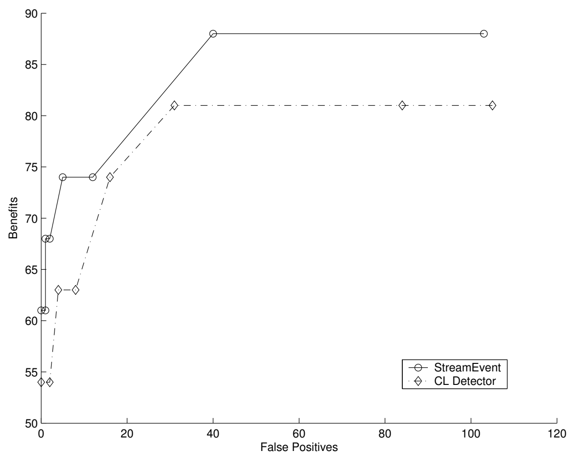

Figure 10: False Positives versus Benefits of Quick Detection (SStream-C)

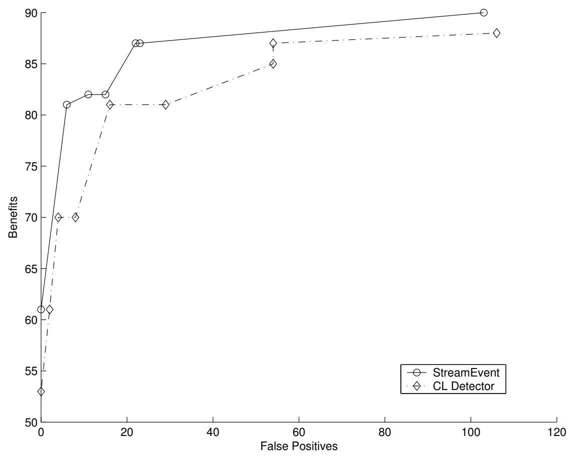

Figure 11: False Positives versus Benefits of Quick Detection (SStream-D)

The AMOC curves for both the *EventStream* and CL detector are illustrated in Figures 8, 9, 10, and 11 respectively. We note that in each case, the AMOC curve for the CL-detector algorithm was dominated by our data stream anomaly detector. In fact, in each case, there was not even one point at which the CL detector was superior to the *StreamEvent* algorithm. Furthermore, the gap between the two algorithms was significant in most cases. For example, in most cases, the *StreamEvent* algorithm was able to achieve relatively low latencies for less than 10 false positives. Modest latencies were usually achieved for only about 0-2 false positives in a majority of the cases. On the other hand, the CL Detector was never able to find the anomaly within the maximum required time latency of δ_z in the range of 10 false positives or less. Thus, the region in which the CL detector provided non-zero benefit was one in which the number of false positives was too high for the algorithm to be of practical use. As in the previous case, the *StreamEvent* algorithm significantly outperforms the CL-detector.

We tested the efficiency of the anomaly detection algorithm for processing large data streams. The algorithm was tested on 2.2 GHz laptop running Windows

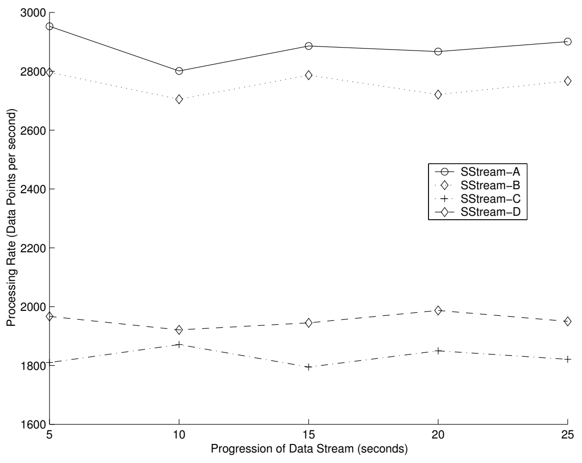

Figure 12: Processing Rate of Data Stream

XP operating system and 256 MB of main memory. We note that represents very modest hardware available today. There were two aspects which needed to be measured:

- **(1) The online efficiency:** This quantity was defined in terms of the maximum number of ticks for which a prediction could be made per second. We define this as the *maximum throughput*, since it is reflects the maximum processing capabilities of the algorithm per unit of time. When the stream arrives at a faster rate than the maximum throughput, then it is necessary to use load shedding techniques in order to reduce the throughput being handled by the data stream. These load shedding techniques can be implemented in the form of sampling points from the data stream. While sampling reduces the effectiveness of the prediction process to some extent, it is an acceptable solution in many practical scenarios.

- **(2) Efficiency of the model update process:** We computed the time required by the offline component in order to update the model. As long as this average time was significantly lower than the time between the occurrence of two events, the model was always up to date at the time of prediction.

In Figure 12. we have illustrated the processing rate of the online portion of the event detection process. It is interesting to see that the processing rates for all the streams were quite stable over time and ranged in the order of thousands of data points per second. In Table 1, we have also illustrated the summary of the overall online efficiency of the algorithm for the entire data stream. While all the data sets are processed at the rate of thousands of data points per second, the differences between the different data sets is because of the varying dimensionality of the data sets. An increased dimensionality resulted in lower processing rates, since a larger number of data points needed to be processed every second. However, since the absolute processing rates are quite high, this means that the streams can be processed efficiently using this technique.

In Table 1, we have also illustrated the time required by the offline component of the event detector. The most expensive process in the offline learning algorithm are the iterations of the gradient descent algorithm for finding the weights of the different channels. Since the gradient descent method is an iterative approach, it required a few computations in order to update the model. However, in each case, these computations require less than 5 seconds to execute. On the other hand, since the algorithm is specifically designed for detecting rare events in data streams, this time interval of a few seconds is likely to be negligible compared to the inter-arrival period between two events. Therefore, such an offline update process is easily implementable because of its relative rarity. As in the previous case, are some differences among the update times for the different data sets because of varying dimensionality of different data sets.

4 Conclusions and Summary

In this paper, we proposed a new technique for anomaly detection in massive data streams. The method is capable of fast and accurate anomaly detection in the presence of other non-relevant anomalies in the data. Therefore, the detector is able to distinguish between spurious and true anomalies. Such a system is quite unique in retaining *specificity* in anomaly detection from multi-dimensional data streams. It also has applications in many domains in which real time detection is necessary for quick response times to anomalous events. In future work, we will construct a decision support system for quick anomaly detection in massive data streams. Such a decision support system would utilize active involvement of the user in making decisions about the data.

References

[1] C. C. Aggarwal, *A Framework for Diagnosing Changes in Evolving Data Streams*, Proceedings of the ACM SIGMOD Conference, (2003).

[2] R. Abbott, and H. Garcia-Molina, *Scheduling real-time transactions with disk resident data*, Proceedings of the VLDB Conference, (1989).

[3] D. Bertsekas, *Nonlinear Programming*, Athena Scientific, 2nd Edition, (1999).

[4] M. Berndtsson, and J. Hansson, *Issues in Active Real-Time Databases*, Active and Real-Time Databases, (1995), pp. 142–157.

[5] D. Bonachea, K. Fisher, A. Rogers, and F. Smith, *Hancock: A language for processing very large data*, USENIX 2nd Conference on Domain-Specific Languages, (1999), pp. 163–176.

[6] H. Branding, A. Buchmann, T. Kudrass, and J. Zimmermann, *Rules in an open system: The reach rule system*, First Workshop of Rules in Database Systems, (1993).

[7] J. Feigenbaum, S. Kannan, M. Strauss, and M. Vishwanathan, *Testing and spot-checking of data streams*, Proceedings of the ACM SODA Conference, (2000).

[8] J. Fong, and M. Strauss, *An approximate L^p-difference algorithm for massive data streams*, Annual Symposium on Theoretical Aspects in Computer Science, (2000).

[9] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith, *Hancock: A Language for Extracting Signatures from Data Streams*, Proceedings of the ACM KDD Conference, (2000).

[10] P. Domingos, and G. Hulten, *Mining High-Speed Data Streams*, Proceedings of the ACM KDD Conference, (2000).

[11] V. Ganti, J. Gehrke, and R. Ramakrishnan, *Mining Data Streams under block evaluation*, ACM SIGKDD Explorations, Vol. 3(2), (2002).

[12] J. Gehrke, F. Korn, and D. Srivastava, *On Computing Correlated Aggregates over Continual Data Streams*, Proceedings of the ACM SIGMOD Conference, (2001).

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, *Clustering Data Streams* , Proceedings of the IEEE FOCS Conference, (2000).

[14] R. A. Johnson, and D. W. Wichern, *Applied Multivariate Statistical Analysis*, Fourth Edition, Prentice Hall, Upper Saddle River, NJ, (1999).

[15] T. Lane, and C. E. Brodley, *An Application of Machine Learning to Anomaly Detection*, Proceedings of the 20th National Information Systems Security Conference, (1997), pp. 366-380.

[16] W. Labio, and H. Garcia-Molina, *Efficient Snapshot Differential Algorithms for Data Warehousing*, Proceedings of the VLDB Conference, (1996).

[17] W. Lee, S. J. Stolfo, and P. K. Chan, *Learning Patterns from Unix Process Execution Traces for Intrusion Detection*, AAAI Workshop: AI Approaches to Fraud Detection and Risk Management, (1997), pp. 50–56.

[18] B-K Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris, *Online Data Mining for Co-Evolving Time Sequences*, Proceedings of the ICDE Conference, (2000).