Evaluating the Intrinsic Dimension of Evolving Data Streams

Elaine P. M. de Sousa, Agma J. M. Traina, Caetano Traina Jr.
University of São Paulo at São Carlos – Brazil
{parros, agma, caetano}@icmc.usp.br

ABSTRACT

Data streams are fundamental in several data processing applications involving large amount of data generated continuously as a sequence of events. Frequently, such events are not stored, so the data is analyzed and queried as they arrive and discarded right away. In many applications these events are represented by a predetermined number of numerical attributes. Thus, without loss of generality, we can consider events as elements from a dimensional domain. A sequence of events in a data stream can be characterized by its intrinsic dimension, which in dimensional datasets is usually lower than the embedding dimensionality. As the intrinsic dimension can be used to improve the performance of algorithms handling dimensional data (specialy query optimization) measuring it is relevant to improve data streams processing and analysis as well. Moreover, it can also be useful to forecast data behavior. Hence, we present an algorithm able to measure the intrinsic dimension of a data stream on the fly, following its continuously changing behavior. We also present experimental studies, using both real and synthetic data streams, showing that the results on well-understood datasets closely follow what is expected from the known behavior of the data.

Keywords
Data streams, fractals, intrinsic dimension

1. INTRODUCTION

Data streams are the bulk data continuously generated from a variety of data processing tasks and are fundamental in several applications, including telecommunication networks, financial transactions, stock quotes, tollbooth observations, manufacturing, security, medical data and telemetry data collection, such as atmospheric measurements and sensor networks in general, to name just a few. Therefore, a lot of work has been done on data stream management and mining. An interesting survey on stream mining is presented in [2]. Recent results in the stream mining field include analysis of changes in trends of evolving data [1], clustering [8], classification [7], frequent items identification, maintenance and processing [11].

A data stream is a real-time, continuous, potentially unbounded sequence of events $e_1, e_2, \ldots, e_N$, where each event is represented by a set of $E$ measured attributes $e_i = (a_1, \ldots, a_E)$. Usually, data streams are too large to fit in main memory, then as soon as each event is processed, it is either stored on disk or discarded right away. Thus, events must be accessed sequentially and, in transient data, they must be read only once and analyzed as they are obtained.

In traditional databases applications the intrinsic dimension $D$ of a stored dataset is considered one of the characteristics that can directly impact the performance of data analysis processes. Whereas the number of attributes in a dimensional dataset determines its embedding dimension $E$, $D$ is a measure of the amount of information the dataset represents, and thence it is an indication of how the attributes are correlated [15]. Therefore, algorithms developed to analyze large high dimensional datasets with low $D$ can be configured to expect lower data complexity, thus operating with better performance. The concept of intrinsic dimension has been applied to analysis of access methods, such as cost estimate [6], query optimization [12] and selectivity estimation [4]. It has also been useful in data mining tasks, such as clustering [3], attribute selection [15], time series forecasting [5] and time diversification [9].

Similarly, algorithms built to analyze data streams can also benefit from the knowledge of their intrinsic dimension, which can be explored in cost and selectivity estimation of continuous queries performed over persistent and transient streams, query optimization, feature selection and forecasting (as already proposed for time series [5]). Thus, having a fast method to measure $D$ over time is an important aid in trimming analysis processes of data streams.

![Figure 1: Changes on data stream behavior.](image)

As illustrated in Figure 1, the behavior of a data stream may change over time, so its intrinsic dimension should change accordingly. Consider, for instance, a sensor network
measuring air flow in a wind tunnel. If the air flows slowly, its movement tends to be linear, generating a data stream with low $\mathcal{D}$. However, if the air flow increases, its movement becomes turbulent, and the value of $\mathcal{D}$ is expected to be much higher. We thus propose a method to obtain a continuously changing value of $\mathcal{D}$, following the behavior of changing data. A sliding window bounds the events over which $\mathcal{D}$ is computed, such that older events are discarded as new events arrive. Moreover, each arriving event is processed only once, and the value of $\mathcal{D}$ for windowed events can be answered at any time, as soon as a reasonable number of initial events are provided. We consider every event $e_i$ to be defined by a fixed number of attributes, such that $e_i$ is an element from a dimensional domain. Therefore, the value of $\mathcal{D}$ for windowed events can be estimated by the Correlation Fractal Dimension $D_2$, computed through the "Box-Occupancy Counting" approach [13]. A preliminary version of our work is presented in [14].

The remainder of the paper is organized as follows. Section 2 defines fundamental concepts and tools. Section 3 discusses data streams' properties and the algorithm we developed. Section 4 discusses experimental studies. Section 5 concludes the paper.

2. BACKGROUND

Traditional datasets in dimensional domains can be described as sets of elements (data items) with fixed numbers of attributes. Thus, let $S = \{s_1, s_2, \ldots, s_N\}$ denote a dataset containing $N$ elements composed of $E$ attributes, i.e., $s_i = (a_{i1}, a_{i2}, \ldots, a_{iE})$.

**Embedding Dimension** $E$. Given a dataset $S$, $E \in \mathbb{N}$ is the number of attributes defining each element of $S$.

**Intrinsic Dimension** $\mathcal{D}$. Given a dataset $S$, $\mathcal{D} \in \mathbb{R}^+$ is the dimensionality of the object represented by the data, regardless of the dimension of the space where $S$ is embedded.

A dataset can represent a spatial object with dimensionality lower than or equal to the dimension of the space where it is embedded. For example, a set of points disposed along a line has $\mathcal{D}$ equal to one, no matter if the set is embedded in any higher dimensional space. Notice that, if the attributes of a dataset obey uniformity and independence properties, $\mathcal{D}$ equals $E$. On the other hand, whenever there is a correlation between two or more attributes, the value of $\mathcal{D}$ is accordingly lower. Uniformity and independence, however, are rarely found in real data. Indeed, experimental evidences have shown that the distribution of distances between elements in the majority of real datasets does not follow Uniform or any of the traditional statistical distributions, such as Gaussian or Poisson. Instead, most real data present "fractal behavior" [4].

**Fractal Behavior**. A dataset exhibiting fractal behavior is self-similar over a large range of scales. In other words, it exhibits roughly the same properties for a wide variation in scale or size, such that parts of any size of the data are similar (exactly or statistically) to the whole dataset [13]. The fractal behavior of a real, self-similar dataset leads to a distribution of distances which follows a power law [4].

**The intrinsic dimension of datasets.** It has been shown [4] that, given a dataset $S$ of $N$ elements and a distance function $d(s_i, s_j) \rightarrow \mathbb{R}^+$, the average number $k$ of neighbors within a distance $r$ is proportional to $r$ raised to $\mathcal{D}$. Thus, the number of pairs of elements within distance $r$ (the pair-count $PC(r)$) follows a power law, where $K_p$ is a proportionality constant:

$$PC(r) = K_p \cdot r^\mathcal{D}$$  (1)

Note that by evaluating the distances between every two elements of a dataset $S$ we can plot a graph depicting the distribution of distances in $S$, namely $PC(r)$ versus $r$. For a self-similar (fractal) dataset, the distribution of distances plotted in log-log scale is straight for a significant range of $r$, thus the slope of the best-fitting line corresponds to the exponent in Equation 1, and closely approaches the intrinsic dimension $\mathcal{D}$ of the dataset [4, 15].

**Measuring $\mathcal{D}$ in dimensional datasets.** The value of $\mathcal{D}$ can be estimated by the Correlation Fractal Dimension $D_2$ in a range of scales $[r_1, r_2]$. Hence, given a dataset $S$, $D_2$ can be measured by using the “Box-Occupancy Counting” approach, as follows [4].

$$D_2 = \frac{\partial \log \left( \sum_{i} C_{r,i}^2 \right)}{\partial \log (r)}, \quad r \in [r_1, r_2]$$  (2)

where $r$ is the side of the cells in a bounding (hyper) cubic grid which divides the address space of $S$, and $C_{r,i}$ is the count of elements (‘occupancy’) in the $i$-th cell.

3. MEASURING THE INTRINSIC DIMENSION IN DATA STREAMS

We consider a data stream as an unbounded sequence of events $e_1, e_2, \ldots$, each of which represented by an array of $E$ measurements, $e_i = (a_{i1}, \ldots, a_{iE})$. Although a stream is potentially unbounded, by using a sliding window we are able to deal with windowed events as a dimensional dataset, thus estimating its intrinsic dimension through $D_2$ calculation. However, algorithms based on the Box-Occupancy Counting approach, applied to measure $\mathcal{D}$ from traditional datasets, usually present two characteristics which restrict their use for data streams:

- $\mathcal{D}$ is computed once, after the full dataset is available;
- the bounding hyper-cube is defined once and elements outside it are not allowed.

Figure 2: Dataset MGCounty - $D_2 \approx 1.80$. Based on Equation 2, the Box-Counting plot is constructed by plotting, in log-log scales, the sum of squared occupancies $\sum_i C_{r,i}^2$ for distinct values of $r$. For a fractal dataset, the curve closely approximates a line, whose slope approaches $D_2$ in the range of scale $[r_1, r_2]$. Figure 2 shows the Box-Counting plot of the MGCounty dataset, whose elements are the geographic coordinates of streets and roads of Montgomery County. As it can be noted, the curve is linear for a significant range of $r$.  

![Box-Counting Plot of MGCounty](image)
In essence, an algorithm to calculate changing values of \( \mathcal{D} \) from a data stream must comply with the following requirements.

1. The updated value of \( \mathcal{D} \) can be asked at any time.
2. Events are processed as they arrive and may never be used again, thus requiring a single pass over the data.
3. The address space of windowed events can scale or translate during time, as new events arrive and older ones are discarded.
4. \( \mathcal{D} \) can change slowly as new events arrive and older ones are no longer considered.
5. A sequence of events may exceed main memory.
6. Each individual event presents very little information by itself, thus a meaningful change on the stream behavior requires a sequence of events.

In order to meet these requirements, we designed a new algorithm to measure \( \mathcal{D} \) based on the Box-Occupancy Counting approach but not limited by the aforementioned restrictions. Figure 3 illustrates a conceptual framework using our algorithm, named \( \text{SID-meter} \) (data Stream Intrinsic Dimension meter). The events coming to the Analyzer Process are captured by the \( \text{SID-meter} \) module, which evaluates and updates the current value of \( \mathcal{D} \). The Analyzer Process can thus read \( \mathcal{D} \) asynchronously.

The \( \text{SID-meter} \) works as follows. As the address space of the stream may change over time, accompanying the data evolution, the first step of the \( \text{SID-meter} \) is to estimate an initial bounding hyper-cube by using the first few arriving events as soon as the process is started. Although the meter cannot provide a value for \( \mathcal{D} \) during this initial stage, it is in accordance with requirement 6, as a meaningful measure of \( \mathcal{D} \) always requires a minimum number of events.

The first events of the stream are kept in sequence in main memory until a predefined, parameterized number \( n_h \) of events is received, such that an initial hyper-cube can be determined. The lowest \( r_l \) and highest \( r_h \) values for each attribute \( a_i \) of the received events are computed. The range \( r = \max(r_h - r_l) \) determines the size of the side in the initial hyper-cube. The hyper-grid structure is created by generating up to \( R \) successive \( E \)-dimensional grids of cell side \( r = r_{j-1}/2 \), where \( R \) determines the number of points in the Box-Counting plot. Notice that, for each cell at level \( j \), \( 2^j \) cells are generated in level \( j + 1 \).

The data structure supporting the algorithm is a tree, named counting tree, maintained in main memory. Each level \( j \) in the tree corresponds to the grid of the cell side \( r_j \) (with \( r_0 \) corresponding to the root level) and each node corresponds to a cell, as shown in Figure 4. Every cell is identified by \( \text{identifier}[b_1,b_2,...,b_E] \) as part of one cell in the immediate upper level, such that \( b_i = 0 \) for cells in the lower half of dimension \( i \) and \( b_i = 1 \) otherwise. It is relevant to highlight that a tree node is created only when there is at least one event on the corresponding cell of the grid. Therefore, the total number of nodes at each level is at most the number of events inside the sliding window.

![Figure 3: Measuring \( \mathcal{D} \) through data streams.](image)

![Figure 4: A 2-dimensional 3-level grid structure and the corresponding counting tree.](image)

![Figure 5: Counting periods in a sliding window through a 3-dimensional data stream.](image)

The counting periods are represented, at each node of the tree, by an array \( C_i \) of \( n_c \) counters, one for each period \( k \), as illustrated in Figure 5. \( C_i \) is used as a circular list, such that a current counter computes the occupancy (number of events) of the arriving events on the corresponding current counting period. When a period expires, the next counter is zeroed and used to count the events of the following period (Figures 5a-c). Thus, when \( n_c \) periods are over, the oldest events are discarded and the next counter is used to compute occupancy of new events. (Figure 5d). By using a sliding window divided into counting periods and the corresponding list of counters we meet requirement 4, as the value of \( \mathcal{D} \) is always computed with the contribution of the most recent events only, following the behavior of a evolving stream.
value \( r_i \). Finally, the slope of the line that best fits the graph gives an estimate of \( D \) for the events inside the sliding window. The memory occupied by the events is then released to be used by the counting tree.

In data streams of very high \( E \) the first events can be monitored without being stored, delaying measurements of \( D \) until further events arrive, thus meeting requirement 5. To meet requirement 1, our method allows \( D \) to be calculated whenever a counting period is complete or at any time it is needed. In the last case, as the current period may not be complete, the current counter is not considered to compute \( C_{r_j,i} \). Requirement 2 is also satisfied, as not even the first \( n_i \) events need to be stored or read twice in order to compute \( D \).

Finally, as mentioned in requirement 3, the address space and the bounding hyper-cube of windowed events may change. In fact, there are four possible movements of the address space over time, as shown in Figure 6.

- Occupation of regions (Figure 6a).
- Release of regions (Figure 6b).
- Expansion of the address space (Figure 6c).
- Contraction of the address space (Figure 6d).

![Figure 6: Address space and tree movements.](image)

The "occupation of regions" occurs whenever a new event falls in a previously empty cell and a new node is allocated in the corresponding level of the counting tree (Figure 6a). Accordingly, when a node is created at level \( j \), new nodes are recursively created at every level \( j + 1 \), up to the required number of levels \( R \).

The "release of regions" occurs whenever every counter \( C[k] \) of a node \( i \) equals zero after a counting period expires. In other words, if every counter of a node is zero (i.e., \( C_{r_j,i} = 0 \)) then no event bounded by the sliding window has fallen in the corresponding cell. Therefore, as every node in the subtree rooted at \( i \) is also related to empty cells, the whole subtree can be released (Figure 6b).

The "expansion of the address" space can be fulfilled by Algorithm 1. If a new event \( e \) falls outside the bounding hyper-cube, it is scaled to bound \( e \), i.e., the lowest (\( rl_i \)) or the highest (\( rh_i \)) value of every attribute \( i \) is updated accordingly, keeping the same range size for every dimension of the new hyper-cube. As shown in Figure 6c, the counting tree is expanded one level at the root and, to maintain the same number \( R \) of points in the log-log plot, the last level of the tree is released. The event \( e \) is then counted.

The "contraction of the address space" occurs whenever every node but one in the first level under the root has \( C_{r_j,i} = 0 \), i.e., there is only one occupied cell in the first level of the hyper-grid. Thus, two main operations are performed: firstly, empty subtrees are removed, as done in "release of regions"; secondly, the root is released and the node \( i \) in the first level with \( C_{r_j,i} > 0 \) is set as the new root (Figure 6d). The bounding hyper-cube is also reduced, as presented in Algorithm 2.

**Algorithm 1** Expand(event \( e \))

**Input:** event \( e\) = \( a_{i1}, a_{i2}, ..., a_{ik} \) > extending the hyper-cube

**Output:** expand the tree of counters accordingly

1: expanded = false;
2: for \( i = 1 \) to \( E \) do
3: if \( a_i < rl \) then
4: \( aux = root; root = newnode; expanded = true; \)
5: \( child\_pointer(root) = aux; sibling\_pointer(root) = null; \)
6: \( identifier(root) = [00..0] \)
7: \( C[k](root) = C[k](aux), k = 1, ..., n \)
8: end if
9: end for
10: if expanded then
11: return expanded;
12: end if

**Algorithm 2** Contract(root)

**Input:** a pointer root to the counting tree

**Output:** contract the counting tree

1: contracted = false; \( aux = child\_pointer(root); \)
2: for \( i = 1 \) to \( E \) do
3: if \( b_i(identifier(aux)) = 0 \) then
4: \( rh_i = rh_i - r_0; \)
5: \( rh_i = rh_i - r_0; \)
6: \( rh_i = rh_i - r_0; \)
7: delete root; root = aux;

To summarize, the three methods that comprise the SIDS-meter can now be described as follows.

**Read a new event** \( e \) - Check if event \( e \) is inside the bounding hyper-cube being considered. If it is not, perform an "expansion of the address space". Hence, execute a point-deep navigation in the tree guided by \( e \), updating the current counter at every level. Whenever \( e \) occupies a previously empty cell at level \( j \) perform an "occupation of regions". If there is no available memory to create new nodes, the last level of the tree is removed and the memory is used by new nodes. It is important to mention that, in order to reduce dynamic memory allocation and deallocation, the SIDS-meter maintain a linked list of free nodes, allocated when the module begins to work and released after the whole data stream is processed.

**Set a new period** - Perform a full navigation through the tree and compute \( \sum C_{r_j,i} \) for every level \( j \). Then, obtain the log-log plot, find the slope of the best fitting line and return it as \( D \).

**Signalize a counting period** - Increment the current counter index \( k \) by one module the number of periods \( n_j \). Perform a full navigation through the tree zeroing the \( C[k] \) counter in every node \( i \). At each node, check if there is at least one \( C[k] > 0 \); otherwise, perform a "release of regions". Then, ensure that the first level below the root has more than one cell; otherwise, perform a "contraction of the address space".

To speedup the process, the SIDS-meter keeps the counting tree in main memory. As the number of nodes at each level
is at most the number of events \( n_c \times n_i \) in the sliding window, the memory requirement is \( O(n_i \times n_c \times R) \), where \( R \) is the number of levels in the tree. Thus, current systems can manage a reasonable number of events. Furthermore, as the events are not stored, the memory usage does not depend on the number of attributes \( E \) of the data stream. The computational complexity of the algorithm to evaluate \( D \) of windowed events is \( \Theta(n_i \times n_c \times E \times R) \).

4. EXPERIMENTAL RESULTS

Several experimental studies on real and synthetic data streams were performed to evaluate and validate the method we propose. Table 1 shows a summary of the datasets we discuss in this work, giving their number of events (\( N \)) and number of attributes (\( E \)). The value of \( D \) was calculated after each counting period considering all the events inside the sliding window and using 20 grid sizes (\( R = 20 \)). The experiments were performed on a P4 3.0GHz, with 0.5GB of RAM, running the MS Windows XP.

Dataset Synthetic3D was constructed such that the values of the attributes in the first 3,000 events of the data stream characterize a line; the following 3,000 events characterize a plane and the values of the last 4,000 ones are distributed like in a cube. We measured the behavior of \( D \) using a sliding window divided into 10 counting periods (\( n_c = 10 \)) with 300 events each (\( n_i = 300 \)).

Figure 7 shows the values of \( D \) in the end of successive counting periods. As expected, \( D \) is very close to 1.0 at the end of period 10, when the sliding window is full and \( D \) is calculated using the first 3,000 events distributed along a line. From the 10th period forth, the 300 oldest events are discarded whenever a period begins. From periods 11 to 20, the events characterizing a plane change \( D \), which increases to 2.0 accordingly. Finally, when the last 4,000 events begin to be processed, \( D \) evolves to stabilize close to 3.0, following the behavior of the changing data, as expected.

The Spot-Exrates dataset [10] contains work-daily spot prices (foreign currency in dollars) of 12 currencies over the period of 10/SEP/86 to 8/SEP/96. In our experiments, we considered each currency as an attribute (\( E = 12 \)) and the daily prices of the 12 currencies as an event (\( N = 2,567 \)). We chose to use the events of (approximately) a year to define each counting period, thus allowing further analyzes on the behavior of the stream from year to year.

In Figure 8 we can observe that the variation of \( D \) is more significant during the first 4 years, indicating that the fluctuation of the spot prices in this period was higher than that in the following years. Moreover, we can say there are many correlated currencies, as \( D \) is much lower than the embedding dimension \( E \) over time. Recall from Section 2 that if the attributes of a dataset are completely uncorrelated \( D \) closely approaches \( E \); otherwise, if there are correlated attributes, \( D \) is lower than \( E \) accordingly.

Figure 7: Variation of \( D \) in Synthetic3D.

Figure 8: Variation of \( D \) in Spot-Exrates.

The Wind dataset [10] contains daily average wind speeds at 12 synoptic meteorological stations in the Republic of Ireland, from year 1961 to 1978. The original data is composed of 15 attributes: year, month and day of the measurements, and average wind speed at each of the 12 stations. We used only the 12 values of wind speed, and defined each period based on the events of a year, as done on the previous experiment.

Figure 9: Variation of \( D \) in Wind.

In Figure 9, the increase in \( D \) during the first 10 years indicates a meaningful variation on the behavior of the data, that is, the values of wind speed measured at the stations varied such that the correlations between the attributes also changed, increasing the value of \( D \) from 5.6 to 6.9. The dataset Evaporator [10] has 3 attributes measured at the input and 3 at the output of an industrial four-stage evaporator system. The inputs measure the air flow and the vapor flow to the first evaporator stage, and the cooling water flow, respectively. The outputs measure the dry mat-
The description of Evaporator does not include information about sampling time. Thus, we performed the experiment using approximately 1/10 of the stream in each counting period ($n_i = 630$), dividing the sliding window into 5 periods. Figure 10 shows that outputs of the system are somehow correlated to the inputs, as $D$ remains significantly lower than $E$. We can also observe that there is no meaningful change on the behavior of the stream, as the highest and the lowest values of $D$ are very close.

![Figure 10: Variation of $D$ in Evaporator.](image)

5. CONCLUSIONS

From the perspective of a database, one of the largest differences between a dataset that can be updated at any time and a data stream is that the dataset always has a current “state”, where queries can be asked. In a data stream there is no such state, and the questions regarding the behavior of the data are made over the last period of time. To be able to answer queries about a data stream, it is needed to store parameters about the data as the events flow. Thus, these parameters can be used to answer a query whenever it is posed, as the events cannot be retrieved anymore. One of the most important properties of a set of data, be it a traditional one or a data stream, is its intrinsic dimension $D$. This property has a large impact over the way many queries are executed, and its knowledge can lead to significant performance differences when processing a query, as $D$ is usually lower than the embedding dimension $E$. In particular, as it has already been done in traditional database management, the value $D$ can be explored in stream management tasks such as cost and selectivity estimation, query optimization and forecasting. Without loss of generality, such tasks can use the changing $D$ to represent the behavior of the data, instead of using the number of attributes $E$ as a parameter.

Our work proposes a method to continuously measure $D$ in data streams. The SID-meter module performs such calculations with the following interesting properties:

- It can track $D$ even when the behavior of the data stream skews intensely over time.
- After a short start-up period, $D$ can be asked at any time.
- Its complexity is linear on the number of events inside a sliding window, and independent of the number of attributes of the data stream.
- Its precision depends on the main memory availability, and it can self-adjust to achieve better precision if more memory becomes available.

6. REFERENCES


Acknowledgement

This research has been partially supported by CNPq under grants 52.1685/98-6, 860.068/00-7 and 35.0852/94-4, and by FAPESP under grant 01/03370-6. The authors are grateful to Eamonn Keogh for providing real datasets [10].