Second International Conference
Modelling and Development of Intelligent Systems
Sibiu - Romania, September 29 - October 02, 2011

A technique for constructing training sets in data stream mining: kSiEved Window Training Set

Sabina Surdu

Abstract

One of the challenges that data mining is facing today is applying its techniques to dynamic data, *i.e.* data streams. Data streams are produced over time by data sources and systems that process them run continuous queries in a perpetual manner, in order to produce streamed results. New data management and query processing paradigms emerged in this context. Applying data mining algorithms on data streams requires adapting the classical methods of data mining to the novel processing paradigms. One of the main challenges in data stream processing is represented by limited system resources. Our goal is to provide a resource-aware technique that constructs training sets for a data stream mining algorithm, namely the kSiEved Window Training Set technique.

1 Introduction

Data mining has been around for a while, in a great number of fields, ranging from science to business domains. The technology enables hidden patterns discovery and extraction from large data sets and predicting future values for data elements based on some known information. But the data on which one might apply data mining is not always static. One of the current challenges in the field is to perform data mining on rapid, continuous streams of data.

Consider the case of an astronomic application, that receives data from a large number of celestial bodies, which emanate radio waves. Processing such data could aim at discovering similar planets in neighbouring parts of the galaxy. A surveillance system continuously processes streams of information from multiple monitoring devices, to detect unauthorized access. ATM transactions also generate streams of continuous data, recording clients operations, to detect fraudulent activity. Monitoring clickstreams in electronic commerce can reveal various patterns in web sites usage, which can lead to better design solutions, serving both the customer and the business.

In an increasing number of domains, continuous data processing has become a stringent requirement. Novel paradigms are developed for data management and query processing in this context. Although stream processing technology has yet to reach maturity, data mining requirements are already formulated for streaming applications, as the above examples show.

In this paper we introduce a novel, resource-aware technique that builds training sets for a data stream mining algorithm (the design of such an algorithm is out of the scope of our paper). The kSiEved Window Training Set (kSEWT) technique constructs training records by sifting streams in a parameterized manner, depending on the value of the sieve parameter k and specified error bounds.

Our objective in this paper is two-fold. Apart from introducing kSEWT, we also provide a high-level description of stream processing models and paradigms. We motivate the need to perform data mining on continuous data. We highlight the issues encountered when processing data streams, which represent core challenges in data stream mining.

The paper is structured as follows. In section 2 we discuss basic aspects concerning data mining. Section 3 takes a more detailed look at data stream processing. Exemplifying with specific Data Stream Management Systems, we show the main difficulties and challenges encountered in designing such systems, highlighting problems that need to be tackled when applying data mining techniques. In Section 4 we present our kSiEved Window Training Set technique. We provide an informal description as well as the formalized kSEWT model. Section 5 discusses the kSEWT strategy and presents experimental results. Section 6 provides future research directions. Section 7 concludes on the results of our work.

2 Data mining

2.1 Data proliferation

We are moving through the Information Age at a constantly increasing pace. The quantities of produced information expand on a daily basis. [14] identifies *data overload* as one of the main challenges raised by the digital era. Nowadays databases are growing in two dimensions: number of fields and number of objects they contain. Moreover, recently emerged data streams are infinite.

Data proliferation is a result of both computer hardware and software advancements. The cost of storing large volumes of data is perpetually decreasing, whereas storage capacities are increasing. Processing computing power is also advancing on a daily basis. Constant improvements in transmission speed enable larger quantities of data to be transported over networks. Fast progress in data capture also plays an important role, as input data can come from a generous palette of devices: sensors, ATMs, RFID tags, hand-held devices, GPS receivers, etc.

We are heading towards the Internet of Things, an ubiquitous network society composed of interconnected devices, that provide useful information to the user, whatever his or her location [23]. The interconnectivity of smart, communicating devices enables data flow paradigms that haven't been encountered before. [16] describes a pervasive environment, where one can query classical data (like data from databases), data streams and functionalities in a consistent, generalized and declarative framework. Such an enviromnent, queryable in an abstract, easily read by humans manner, will lead to even greater quantities of data than those we need to cope with today. Scientific databases (*e.g.* radio astronomy databases) are expected to reach exascale sizes in the years to come [9]. [19] results show that in 2007 the world's storage capacity was 290 exabytes (1 exabyte = 10^{18} bytes), the humankind communicated 2 zettabytes (1 zettabyte = 10^{21} bytes) and the computing power of general-purpose computers was 6.4 exa instructions per second.

2.2 Motivation

In this context, organizations, research communities and other entities have started to base their decisions chiefly on the data they collect [18]. In some cases, raw data needs to be mined in order to extract useful information for a decision maker.

Take for instance the case of a chocolate company that sells three times as much chocolate Santa Clauses during Christmas than it usually does. The company will naturally supply the retailers with increased quantities of chocolate Santa Clauses right before Christmas. Further data analysis can reveal that 85% of the people who buy chocolate Santa Clauses also buy chocolate ornaments for the tree. These patterns are called association rules with chocolate Santa Clause as antecedent and chocolate ornament as consequent, having a confidence factor of 85% [3]. The chocolate company can maximize its revenue during Christmas, by supplying increased quantities of both chocolate Santa Clauses and ornaments. This technique is usually referred to as market basket analysis. Another tactic than can be employed is arranging products that are part of the same association rule next to each other on shelves.

In a financial context, a bank may want to predict a new customer's probability to default a loan, based on existing information it has on its current customers. A monitoring application in radio astronomy could aim at automatically classifying the monitored objects, based on radio waves values. Marketing campaigns could target new customers that are more likely to respond to the offers, instead of randomly choosing potential clients. And the examples can go on.

In the past analysis was performed by human specialists. But on data sets with millions of records and hundreds or thousands of fields, direct hands-on analysis is not possible any more. The process of extracting useful information needs to be performed by computer, in an automated manner.

2.3 Data mining

Data mining is the automated analysis of large data sets in order to find unsuspected relationships and to summarize the data in novel, understandable and useful ways [17].

[14] emphasizes the fact that data mining is a step in the Knowledge Discovery in Databases (KDD) process. KDD refers to the whole process of mapping low-level, raw data into high-level information. Apart from data mining, KDD also include steps like data collection, cleaning, integration, selection and transformation, and model evaluation and validation.

Numerous techniques can be applied in the data mining step. [22] enumerates, among the most important ones: artificial neural networks, decision trees, genetic algorithms, nearest neighbour method and rule induction.

Data mining can be driven by two types of goals: verification and discovery [14]. Verification restricts the data mining task to the process of verifying previously formulated user hypotheses. Discovery allows data mining to find unknown patterns in data. Furthermore, discovery goals can be classified as either descriptive (find patterns that describe data) or predictive (find patterns that predict future behaviour).

New challenges arise when considering data streams as input to data mining algorithms. Another dimension is to be considered in the analysis process: time. Consider a stream that contains records representing customers transactions. And let a transactional database represent, at each time instant, a snapshot of the stream (*i.e.* a large window). If the temporal variables are not taken into account, a mining algorithm can provide association rules such as $BuyProductA \Longrightarrow BuyProductB$. However, if time is also considered, association rules like these can be discovered: $BuyProductA \Longrightarrow BuyProductB$ jumps to 70% at the weekend, during 6pm to 10pm [24]. These temporal rules allow retailers to make more efficient, time-aware promotion strategies.

3 Data stream processing

3.1 Data streams and continuous queries

Data streams are sequences of values produced in a continuous fashion by data sources. In our paper we consider streams of structured data, which contain elements or records, similar to those found in relational databases. From now on we will use the terms record, tuple, element and event interchangeably, to denote the basic unit of data that arrives on a stream. Consider the example of Saturn's radio emissions. The planet is the data source that provides a stream of radio data. Figure 1 depicts three tuples that arrive on the stream displaying Saturn's emissions, one every 30 seconds (the values are normalized).

In a data stream, every element has a timestamp, which represents the time (logical or physical) when that element arrived on the stream. If a patient in a hospital has a monitoring device attached, which tracks the blood pressure, one can tell the value of the monitored parameter at any time. Traditional data from classical databases doesn't encompass the notion of time. This apparently minor add-on has a significant influence on the data processing paradigm.

Applications that handle data streams are called monitoring applications, because they usually scan or monitor a number of data streams. These applications are best served by dedicated systems that process data streams, called Data Stream Management Systems (DSMSs). The main feature of DSMSs that distinguishes them from traditional Database Management Systems (DBMSs) is the fact that they run continuous queries over input data streams. Queries in traditional DBMSs are executed once against stored data relations, in a finite amount of time (these are called one-shot queries [5]). Queries in DSMSs are executed continuously against transient input data streams. Experiments in [6] show that a DSMS outperforms a DBMS when processing high load data streams and executing both continuous and one-shot queries.

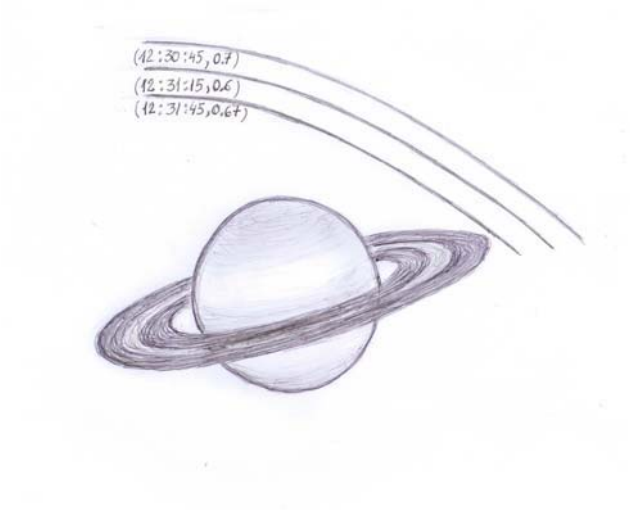

Figure 1: Stream of Saturn's radio emissions

The results of continuous queries are provided in real time, with a specified low latency requirement. For instance, in a financial application, one might want to be alerted whenever a certain stock exceeds a threshold. This kind of answer needs to be produced immediately, it is useless if it is delivered later in the future. Moreover, data is usually processed in a one-pass manner, *i.e.* each tuple that arrives on the stream is processed and then discarded. On the fly processing is a chief trait of DSMSs. Some history of input data can be stored, but in the end it is deleted.

The temporal dimension must be taken into account by any data stream mining algorithm. Furthermore, training sets can be constructed using continuous queries.

3.2 Data Stream Management Systems

We describe the main challenges in processing data streams with three DSMSs, that we consider to be highly relevant in this field.

Aurora is a DSMS developed at Brandeis University, Brown University and M.I.T [1], that allows users to express queries via a visual programming language. Queries are created by dragging operators on a GUI and appear in the form of network diagrams, where boxes represent operations and arcs represent data streams.

STREAM is a general purpose DSMS developed at Stanford [5]. STREAM's continuous queries are written in an SQL-like declarative interface, using a specialized query language: CQL (Continuous Query Language). Both STREAM and Aurora are centralized DSMSs.

Borealis is a second generation stream processing engine, which inherits its stream functionality from Aurora [2]. It expresses queries using the boxes-and-arrows paradigm from Aurora.

For each of the described systems, a data stream is a sequence of elements produced by a data source, where each element is attached to a timestamp [7]. This stream is append-only for Aurora and STREAM (an element once arrived on a stream cannot be deleted or updated). Borealis however encompasses a useful mechanism of revision processing, which means previously arrived tuples on a stream can be corrected (and so can the results of queries that take into account such data) [4].

As we can see, up to this point there are no standards for a data model (revision processing is not present in all systems) or a query language over data streams. One of the main difficulties for applying data mining in this context is coping with different data models and different ways of specifying continuous queries.

3.3 Improving performance

One way to improve performance in DSMSs is by scheduling query operators in a way that reduces memory and CPU usage. Aurora schedules operators using a train superbox technique [13]. Multiple tuples are pushed as long trains through multiple boxes, in order to reduce box call overhead, improve box optimization and avoid the cost of going to disk. STREAMs scheduling approach is adaptive. If the stream tuple rate is uniform, the system uses a FIFO approach, which aims at minimizing state. If the arrival tuple rate is bursty, STREAM uses its Chain algorithm [8]. Taking into account Quality of Service values, Borealis can schedule operators in a way that maximizes the chances of obtaining important values results [2].

In order to cope with high input data rates, the DSMSs sometimes perform load shedding, *i.e.* they drop tuples, saving both memory and CPU usage. In this manner, even if the data rate is bursty, the DSMS can still provide answers to its continuous queries in a timely fashion. Aurora and Borealis perform load shedding taking into consideration QoS values. Each output value is associated with a QoS function which indicates the quality of the output (in terms of latency, value, etc). Aurora performs two types of load shedding: random and semantic [1]. STREAM performs load shedding by dropping input tuples or minimizing state when the system is overloaded.

Aurora and Borealis base their load shedding and operator scheduling decisions on QoS information, a notion we don't encounter in STREAM. A performant data mining algorithm should also take into consideration different approaches to operator scheduling and load shedding, in order to improve performance, by reducing resource usage.

In this section we provided a general overview of data stream processing aspects. In [21] we give a more detailed presentation of data streams and DSMSs.

4 kSiEved Window Training Set technique

4.1 Mining data streams in a resource-constrained setting

We identify one of the core difficulties in data stream mining to be resource usage. Data mining on multiple, high load data streams poses problems concerning memory usage as well as CPU time.

In [20] we addressed the problem of resource usage in a continuous setting, by assessing the sizing window effect over continuous queries results. We will borrow some specifications from this model in the kSEWT technique.

[15] describes data-based subset oriented data stream mining solutions as techniques that choose a subset of the input data stream to be subject to data analysis. Examples include sampling (probabilistic choice of a data tuple to be processed or not), load shedding (dropping fractions of input data streams) and sketching (vertically project the data stream, by omitting some fields in its schema). We propose a data-based subset oriented approach to analysing data streams. The basic components of such a technique are represented by the data stream training sets, constructed in an efficient manner, using load shedding mechanisms.

4.2 Motivation

Stream data rates can be very high. [11] enumerates several requirements that must be met by a stream mining algorithm, among which we are interested in two: the algorithm must use a limited amount of memory and work in a limited amount of time. Under heavy load input streams, it is difficult to meet these expectations [12]. Hence we will build training sets that are to be supplied to an algorithm in a resource-aware manner, using queries over sliding windows.

With existing computational and memory capabilities, it is physically impossible to compute a continuous query on an entire stream of data, since the stream is infinite. Therefore queries must take into account portions of the streams, *i.e.* sliding windows. A sliding window represents a finite sequence of data from a stream. As new elements arrive on the stream, the window slides over them, continuously refreshing itself (oldest elements are removed from the window as new elements get inserted into the window). The result of executing a query against an entire stream of data would be the ideal result.

When running the query against a sliding window, one obtains an approximation of the ideal result. A sliding window has a fixed size, expressed as number of time instants (time-based sliding window) or number of elements it contains (tuple-based sliding window) [5]. In the rest of the paper we will consider time-based sliding windows.

In time series analysis, a common solution for predicting future values is to use the nearest neighbour technique [10]. If a stream contains temporal stock prices, one approach is to construct training records encompassing 10 consecutive stock prices (the first 9 values represent the predictor, whilst the 10th element is the prediction value). Training records built in this manner are in fact sliding windows.

But even these approximate answers can be computationally difficult to compute. Consider for instance a training set that is the result of a multiple attribute join between three 60 seconds sliding windows (training records) over three data streams, each of which has a data rate of over 1000 tuples / second. Submitting training sets like these to a data mining algorithm would put a great pressure on resource usage.

To address the problem of constructing training sets over streams with high input data rates in a limited resource setting, we designed the kSiEved Window Training Set technique.

4.3 Informal description

We provide an informal description of our technique, based on an example from astronomy. We consider that the result of a query on a sliding window is the correct result. The correct result is good enough for our requirements and is an approximation of the ideal result, which could be obtained if we could execute the query against the entire stream.

Consider a data stream that gets tuples from multiple radio telescopes, which monitor different portions of the observable universe. A data mining algorithm could aim at discovering hidden patterns in the stream, indicating for instance the "traffic" of celestial bodies in the monitored areas. An end user might be interested in predicting the number of meteorites passing by whenever a given event occurs. In such a context one would use a specific aggregation class of queries, that mainly performs counting operations.

We intend to build training sets in a way that is dependent on the end user requirements. If the mining algorithm mainly uses aggregation queries that perform certain tasks, we integrate this class of queries in the process of building resource-aware training sets. We do not limit the goal of data mining to only encompass verification. The end user doesn't formulate a rigid hypothesis that needs to be verified. Instead, he or she simply provides a narrowing of the hypotheses characteristics that can be found in the mining process.

We are provided with an error bound, *i.e.* if the error of the approximation of a correct result doesn't exceed this value, then the approximation is good. We superpose sieves with decreasing number of holes over a sliding window. The first sieve has the same number of holes as the number of time instants in the sliding window. The sieve sifts the window's elements, generating a sieved window, that contains only the elements in the initial sliding window that passed through the sieve's holes. The result of executing a continuous query against this sieved window is, at any time instant, the correct result. Subsequently we choose another sieve, with a lower number of holes. When superposing this sieve against the window, it sifts windows elements having positions in the window corresponding to the sieve's holes. We constantly decrease the number of holes in the sieve. We compute queries over each of the chosen sieved windows, thus obtaining approximate answers.

We apply these steps at every time instant. Subsequently we assess the correctness of our results with respect to the error bound. We find the sieve for which the error of the approximation is just below the error bound. That sieve will provide us with training records for a training set. We build training sets composed of a variable number of training records. The number of training sets and the number of training records in a training set both depend on the queries used in the process.

4.4 kSEWT model

4.4.1 Notion of time

In order to cope with temporal data, we adopt a slightly modified version of the discrete time model described in [16]. $t_i \in T, i \in \mathbb{N}^*$ represents a time instant and we have that: $\forall i, j \in \mathbb{N}^*, i < j \Rightarrow t_i < t_j$.

T is an infinite ordered set of time instants, which goes from past to future instants. For simplicity we are going to choose the set of positive natural numbers to represent it: $T = \mathbb{N}^* = \{1, ..., +\infty\}$.

4.4.2 Data model

We borrowed basic notations for our model from [16]: A represents the countable infinite set of attribute names and D is the countable infinite set of constants.

A stream S has a schema (a named set of attributes): $schema(S) \subset A$. A tuple s on stream S is an element of $D^{|schema(S)|}$.

We formally define a stream S as follows:

$$S = \{(s,t) | s \in D^{|schema(S)|}, t \in T\}. \tag{1}$$

We formally define a sliding window over stream S:

$$SW(S, [t_i, t_j]) = \{(s,t) \mid (s,t) \in S, t \in [t_i, t_j]\}. \tag{2}$$

We will use $SW_{ij}(S)$ as an equivalent notation to $SW(S, [t_i, t_j])$, to simplify our equations. t_i is the starting point of the window and t_j is the endpoint of the window. A window slides over its stream with one time instant. When the stream S is obvious, we will denote a sliding window over it by SW_{ij}.

We define the size of a sliding window to be the number of time instants it contains: $Size(SW_{ij}(S)) = t_j - t_i + 1$. Since this definition is trivial, when referring to the size of a given sliding window, we will denote it by σ.

We first established these definitions of the stream, sliding window and sliding window size in [20].

4.4.3 kSiEved Window

We denote by $SW(S)$ the set of all sliding windows over stream S. In the following, we consider a sliding window SW_{ij} over stream S, of size σ, therefore we omit S in our definitions, to provide a clearer focus on the sieved window.

We define a function that gives the position of a timestamp t inside its window ($t \in [t_i, t_j]$):

$$position : T \times SW(S) \to \mathbb{N}^*, position(t, SW_{ij}) = t - (i - 1). \tag{3}$$

If one applies this function to each timestamp in the window, starting from the earliest, one obtains the set of positions $\{1, 2, ..., \sigma\}$. We will call this set $Positions$. A window with size σ has σ timestamps and function $position$ maps each of these timestamps to their corresponding ordered positions in the set $Positions$:

$$Positions = position(T_{ij}, SW_{ij}), \text{where } T_{ij} = [t_i, t_j], \text{ for a given window } SW_{ij}. \tag{4}$$

In order to apply a sieve over the given window, we define a function that extracts a subset of positions from $Positions$, in a parameterized manner:

$$kposition : \mathcal{P}(N^*) \times N^* \to \mathcal{P}(N^*), kposition(Positions, k) = \{kp \in Positions \mid kp \equiv 1 \pmod{k}\}. \tag{5}$$

Given a parameter k, we choose the holes in our sieve to be represented by all values from $Positions$ which belong to the congruence class of 1 modulo k. We denote by $kPositions$ the subset from $Positions$ obtained when applying function $kposition$ with parameter k.

We define a kSiEved Window over the sliding window, by a value k ($kPositions$ is computed for window SW_{ij}, as described above):

$$SEW(SW_{ij}, k) = \{(s,t) \mid (s,t) \in SW_{ij}, position(t, SW_{ij}) \in kPositions\}. \tag{6}$$

We denote this window by $SEW_{ij}(k)$. This window has the same temporal boundaries t_i and t_j as the sliding window it was computed from. It however only contains elements whose positions are filtered by the *kposition* function.

4.4.4 Continuous queries

Let Q be a continuous query. We denote the result of evaluating Q at time t against a sliding window by $Q(SW_{it}, t)$ (similar to the notation we used in [20]). This is a correct result. We denote the result of evaluating Q at time t against a kSiEved Window by $Q(SEW_{it}(k), t)$. This is an approximation of the correct result.

We fill formally express the notion of query equivalence. We say that $Q(SW_{it}, t)$ is equivalent to $Q(SEW_{it}(k), t)$ at time t and write:

$$Q(SW_{it}, t) \equiv Q(SEW_{it}(k), t) \text{ wrt } \epsilon \iff distance(Q(SW_{it}, t), Q(SEW_{it}(k), t)) \leq \epsilon. \tag{7}$$

In the following we will define our *distance* function.

4.4.5 Aggregated value distance

We will consider the result of Q at time t is an aggregated value. As a future direction, we also want to define a distance function that takes into account collection results. For the time being, we will only consider the case when the queries issue aggregated results.

Consider t_c to be the current timestamp.

Let R_{sw} be the result of executing Q against window SW_{ic} at time t_c:

$$R_{sw} = Q(SW_{ic}, t_c), R_{sw} \in \mathbb{R}. \tag{8}$$

Consider a kSiEved Window, computed from the sliding window, $SEW_{ic}(k)$. Let R_{sew} be the result of executing Q against SEW:

$$R_{sew} = Q(SEW_{ic}(k), t_c), R_{sew} \in \mathbb{R}. \tag{9}$$

Similar to [20], we define the distance function between two aggregated value results as follows:

$$distance : \mathbb{R} \times \mathbb{R} \to \mathbb{R}, distance(R_{sw}, R_{sew}) = |R_{sw} - R_{sew}|. \tag{10}$$

5 kSEWT strategy

The purpose of our work is to construct a training set encompassing training records in a manner that considers the limited available resources. We save both memory and CPU time by sifting windows elements through sieves. As the number of elements to be processed decreases, both memory and CPU time are saved. We require less memory and less CPU time to store and to process respectively a smaller number of elements. Some general directions are borrowed from our paper [20], but in this case we describe an algorithmic process that is applied during data mining training sets construction.

Let H be a set of hypotheses constraints specified by an end user. H doesn't establish formulated hypotheses, but some limitations on the models that are to be discovered in the mining process. For our astronomical example, H contains an element that simply states that the user is interested in finding

celestial traffic correlations and / or predictions, using aggregated count queries. So we will construct our training sets with a query that asks: "How many objects were seen by all the telescopes in a given timeframe?".

Let Q be this query. Remember that $t_c \in T$ is the current timestamp and consider $t_i \in T$ to be a timestamp that marks a window's starting point in time. Also note that c actually denotes timestamp t_c, for simplicity, where necessary. As t_c slides on the time axis, its value changes. We denote by $CrtTS$ the finite set of all values from T that t_c, the current timestamp, will take in our experiments. Subsequently:

1. Choose a threshold, δ. An approximate answer should not differ by more than δ from the correct answer. This means the query that provides the correct answer is equivalent to the query that provides the approximate answer wrt δ (according to (7)).

2. Compute the correct result of the query executed against a sliding window of size σ:

$$R_{sw_c} = Q(SW_{ic}, t_c) \tag{11}$$

3. Construct sieves over the sliding window. Start with k = 1 and increase k by one unit until it reaches σ. Compute results of Q over kSiEved Windows $SEW_{ic}(k)$:

$$R_{sew_{c_k}} = Q(SEW_{ic}(k), t_c) \tag{12}$$

and compute distances between the correct result and the result of the query against each of the kSiEved Windows:

$$distance_{c_k} = \mid R_{sw_c} - R_{sewc_k} \mid \tag{13}$$

Note: in our experiments, we will normalize the distance function results, so that a distance will take values from the interval [0,1].

4. Perform steps 2 and 3 for all current timestamps t_c from $CrtTS$ (as the current timestamp changes).

5. Once t_c has taken all the values from $CrtTS$, compute the average of distance results over time, for every k value parameter:

$$AvgDistance(k) = \frac{\sum\limits_{c \in CrtTS} d_{c_k}}{|CrtTS|} \tag{14}$$

6. Choose the k value for which $AvgDistance(k)$ doesn't exceed δ, such that $AvgDistance(k+1)$ exceeds δ. This k will provide the kSiEved Window Training Set, that is the set of all kSiEved Windows obtained in the experiment. These windows are the training records that compose the actual training set, that is to be submitted to a generic stream mining algorithm.

The results from step 5 can be plotted on a two-dimensional graph, that will describe how the average distance from the correct result evolves as parameter k increases.

5.1 Experimental results

We conducted our experiments on random data with uniform distribution. The simplified data schema looks like this: (TimeStamp, TelescopeID, IN/OUT), where IN/OUT represents an object entering into or exiting from a monitored area. We considered windows of fixed size $\sigma = 60$ time instants. For such a window, the function that gives the cardinality of $kPositions$ is:

$$NoKPositions : \mathbb{N}^* \to \mathbb{N}^*, NoKPositions(k) = \begin{cases} \sigma/k & \text{if } \sigma \mod k = 0 \\ [\sigma/k] + 1 & \text{if } \sigma \mod k \neq 0 \end{cases}$$

This function gives the number of holes in a sieve that is to be applied on a sliding window of size σ, given parameter k. Figure 2 displays the evolution of the number of holes in the sieve as the k parameter increases from 1 to 60, according to our function $NoKPositions$.

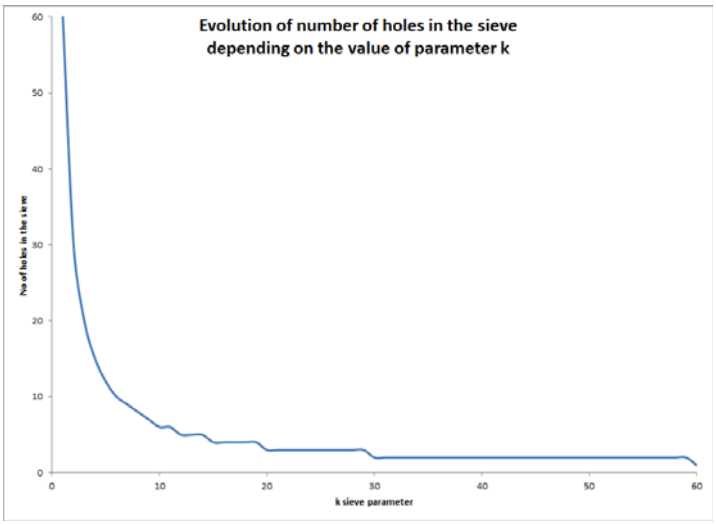

Figure 2: The number of holes in the sieve, depending on k

We followed the strategy described above in our experiments. We plotted the results from step 5 on the graph in Figure 3. If we choose $\delta = 0.5$ from this graph it follows we can apply sieves with parameter k = 2 when constructing kSEWT. This means we are dropping half of the input tuples, which translates to significant resource saving. Different data distributions could lead to greater values for k (we leave this as future work).

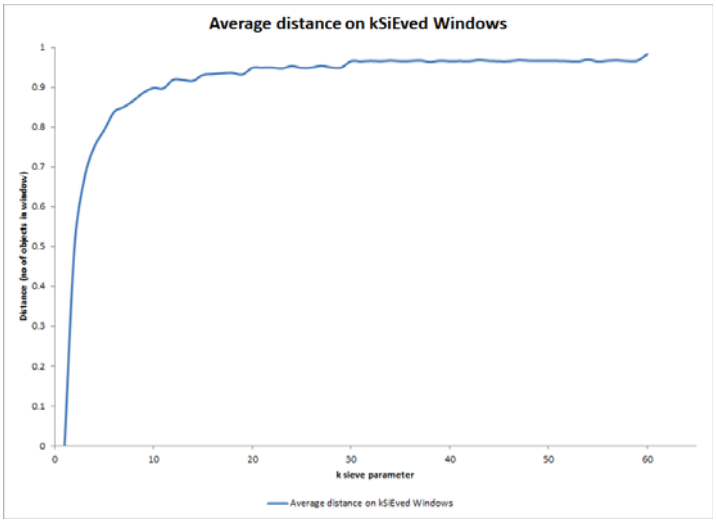

Figure 3: Average distance on kSiEved Windows

6 Future research directions

We are working on a language that allows a rigorous specification of the hypotheses constraints set H. Having such a formalization will allow us to define quantifiable relationships between the mining algorithm goals and the construction of the training sets, via specific continuous queries.

We intend to quantify resource consumption, in terms of memory and CPU usage. The kSEWT

technique will also take into account, apart from the error bound, a resource consumption bound. This will allow the solution to work on different hardware and software platforms, in a highly configurable manner.

In constructing training sets, the technique currently uses a distance function that measures the error of an answer with respect to expanded versions of the training records, *i.e.* the sliding windows. A future goal is to define an improved version of the distance function. This version of the function will take into account models provided from the mining algorithm the training sets are to be submitted to. This way we will greatly improve the accuracy of our results.

7 Conclusion

Data stream processing is still in its infancy. Standards for data models and query processing have yet to be defined, as the field has a long way until it reaches maturity. In this context, data mining tasks must be applied in a heterogeneous environment, where streams are represented in different ways, depending on the system and queries are specified and executed in different manners. We proposed a resource-aware, generalized approach to constructing training sets for a generic data stream mining algorithm, the kSiEved Window Training Set technique. We highlighted the main difficulties we encountered and our solutions to some of these challenges. We also outlined future research directions generated by our work.

Acknowledgement: This paper is supported by the Sectoral Operational Programme Human Resources Development (SOP HRD) 2007 - 2013, co-financed by the European Social Fund, under the contract number POSDRU/6/1.5/S/3.

References

[1] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik, Aurora: a new model and architecture for data stream management. *The VLDB Journal*, vol. 12, n° 2, p. 120-139, 2003.

[2] D.J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.H. Hwang, W. Lindner, A.S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, S. Zdonik, The Design of the Borealis Stream Processing Engine. *Proceedings of the 2005 Conference on Innovative Data Systems Research (CIDR)*, p. 277-289, 2005.

[3] R. Agrawal, T. Imielinski, A. Swami, Mining Association Rules between Sets of Items in Large Databases. *Proceedings of the 1993 ACM SIGMOD Conference*, p. 207-216, 1993.

[4] Y. Ahmad, B. Berg, U. Cetintemel, M. Humphrey, J.H. Hwang, A. Jhingran, A. Maskey, O. Papaemmanouil, A. Rasin, N. Tatbul, W. Xing, Y. Xing, S. Zdonik, Distributed Operation in the Borealis Stream Processing Engine (demo). *ACM SIGMOD Conference*, p. 882-884, 2005.

[5] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom, STREAM: The Stanford Data Stream Management System. *Technical Report*, Stanford InfoLab, 2004.

[6] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A.S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbetts, Linear Road: A Stream Data Management Benchmark. *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB '04)*, p. 480-491, 2004.

[7] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and Issues in Data Stream Systems. *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '02)*, p.1-16, 2002.

[8] B. Babcock, S. Babu, M. Datar, R. Motwani, Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. *Proceedings of Special Interest Group on Management of Data Conference 2003 (SIGMOD'03)*, p. 253-264, 2003.

[9] J. Becla, K.T. Lim, D.L. Wang, Report from the 4th Workshop on Extremely Large Databases. *Data Science Journal*, vol. 9, pp. MR1-MR8, 2011.

[10] A. Berson, S. Smith, K. Thearling, Building Data Mining Applications for CRM. *McGraw-Hill Companies*, 1999.

[11] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, Moa Data Stream Mining. A Practical Approach. *Center for Open Software Innovation, The University of Waikato*, 2011.

[12] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, T. Seidl, Moa: Massive Online Analysis, a Framework for Stream Classification and Clustering. *JMLR: Workshop and Conference Proceedings*, vol. 11, p. 44-50, 2010.

[13] D. Carney, U. Cetintemel, A. Rasin, S. Zdonik, M. Cherniack, M. Stonebraker, Operator Scheduling in a Data Stream Manager. *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB '03)*, p. 838-849, 2003.

[14] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, vol. 17, n˚ 3, p. 37-54, 1996.

[15] M.M. Gaber, A. Zaslavsky, S. Krishnaswamy, Mining Data Streams: A Review. *ACM SIGMOD Record*, vol. 34, n˚ 2, p.18-26, 2005.

[16] Y. Gripay, *A Declarative Approach for Pervasive Environments: Model and Implementation*. Ph.D. Thesis, Institut National des Sciences Appliquées de Lyon, 2009.

[17] D. Hand, H. Mannila, P. Smyth, *Principles of Data Mining*. The MIT Press, Cambridge, MA, 2001.

[18] H. H.O. Nasereddin, Stream Data Mining. *International Journal of Web Applications*, vol. 1, n˚ 4, 2009.

[19] M. Hilbert, P. Lopez, The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, vol. 332, n˚ 6025, p. 60-65, 2011.

[20] S. Surdu, V. M. Scuturici, Addressing resource usage in stream processing systems: sizing window effect. *The International Database Engineering and Applications Symposium ACM International Conference Proceeding Series*, p. 247-248, 2011.

[21] S. Surdu, Data stream management systems: a response to large scale scientific data requirements. *Annals of the University of Craiova, Mathematics and Computer Science Series*, vol. 38, n˚ 3, p. 66-75, 2011.

[22] K. Thearling, An Introduction to Data Mining. *URL: http://www.thearling.com/text/dmwhite/dmwhite.htm* .

[23] International Telecommunication Union, The Internet of Things. *ITU Internet Reports*, 2005.

[24] Q. Zhao, S. S. Bhowmick, Sequential Pattern Mining: A Survey. *Technical Report, CAIS, Nanyang Technological University, Singapore*, 2003.

Sabina Surdu, Ph.D. Student
Babes-Bolyai University
Faculty of Mathematics and Computer Science
Str. Mihail Kogalniceanu nr. 1, RO-400084, Cluj-Napoca
ROMANIA
E-mail: *surdusabina@yahoo.com*