Data Stream Mining: the Bounded Rationality

João Gama
LIAAD-INESC TEC, and FEP, University of Porto
R. Ceuta 118-6, 4050-190 Porto, Portugal
E-mail: jgama@fep.up.pt

Keywords: data stream mining, bounded rationality

Received: October 12, 2012

The developments of information and communication technologies dramatically change the data collection and processing methods. Data mining is now moving to the era of bounded rationality. In this work we discuss the implications of the resource constraints impose by the data stream computational model in the design of learning algorithms. We analyze the behavior of stream mining algorithms and present future research directions including ubiquitous stream mining and self-adaption models.

Povzetek: V prispevku so predstavljeni algoritmi rudarjenja pretakanja podatkov.

1 Introduction

Herbert Simon, one of the AI pioneers, developed the idea of *bounded rationality* [28]:

> *in decision-making, rationality of individuals is limited by the information they have, the cognitive limitations of their minds, and the finite amount of time they have to make a decision. In complex situations, individuals who intend to make rational choices are bound to make satisfactory (rather than maximizing) choices.*

The developments of information and communication technologies dramatically change the data collection and processing methods. What distinguish current data sets from earlier ones are automatic data feeds. We do not just have people entering information into a computer. We have computers entering data into each other [24]. Moreover, advances in miniaturization and sensor technology lead to sensor networks, collecting high detailed spatio-temporal data about the environment. These scenarios impose strong constraints in the way we think and design data mining algorithms. In short, data mining algorithms must take into account that computational resources are limited, and the design of learning algorithms must be thought in the context of *bounded rationality*.

The computational model of data streams [24] requires *resource-aware* algorithms. Hulten and Domingos [16] identify desirable properties of learning systems for efficient mining continuous, high-volume, open-ended data streams:

- Require small constant time per data example;
- Use fix amount of main memory, irrespective to the total number of examples;
- Built a decision model using a single scan over the training data;
- Generating a anytime model independent from the order of the examples;
- Ability to deal with concept drift;
- For stationary data, ability to produce decision models that are nearly identical to the ones we would obtain using a batch learner.

In this paper we review the techniques used in learning from data streams related to the bounded rationality. We also discuss ubiquitous data mining contexts where both data sources and processing devices are distributed.

2 Data streams

In the data stream computational model examples are processed once, using restricted computational resources and storage capabilities. The goal of data stream mining consists of learning a decision model, under these constraints, from sequences of observations generated from environments with unknown dynamics. Most of the stream mining works focus on centralized approaches. The phenomenal growth of mobile and embedded devices coupled with their ever-increasing computational and communications capacity presents exciting new opportunities for real-time, distributed intelligent data analysis in ubiquitous environments. In domains like sensor networks, smart grids, social cars, ambient intelligence, etc. centralized approaches have limitations due to communication constraints, power consumption, and privacy concerns. Distributed online algorithms are highly needed to address the above concerns. These applications require distributed stream algorithms, highly scalable, computationally efficient and resource-aware. Table 1 summarizes the main differences between data base processing and data stream processing.

	Data bases	Data Streams
Data access	Random	Sequential
Number of passes	Multiple	Single
Processing Time	Unlimited	Restricted
Available Memory	Unlimited	Fixed
Result	Accurate	Approximate
Distributed	No	Yes

Table 1: Summary of the main differences between data base processing and data stream processing.

2.1 Approximation and randomization

Bounded rationality appears in a fundamental aspect in stream processing: the tradeoff between time and space required to solve a query and the accuracy of the answer. Data stream management systems developed a set of techniques that store compact stream summaries enough to approximately solve queries. These approaches require a trade-off between accuracy and the amount of memory used to store the summaries, with an additional constrain of small time to process data items [24, 2]. The most common problems end up to compute quantiles, frequent item sets, and to store frequent counts along with error bounds on their true frequency. The techniques developed are used in very high dimensions both in the number of examples and in the cardinality of the variables.

Many fundamental questions, like counting, require space linear in the input to obtain exact answers. Within data stream framework, *approximation* techniques, that is, answers that are correct within some small fraction ϵ of error; and *randomization* [23], that allows a small probability δ of failure, are used to obtain answers that with probability $1-\delta$ are in an interval of radius ϵ. Algorithms that use both approximation and randomization are referred to as (ϵ, δ) approximations. The base idea consists of mapping a very large input space to a small synopsis of size $O(\frac{1}{\epsilon^2} log(\frac{1}{\delta}))$.

Approximation and randomization techniques are the most illustrative examples of bounded rationality in data stream processing. They are used to solve problems like measuring the entropy of a stream [6], association rule mining frequent items [21], k-means clustering for distributed data streams using only local information [9], etc.

2.2 Time windows

Most of the time, we are not interested in computing statistics over all the past, but only over the *recent* past. The assumption behind all these models is that most recent information is more relevant than historical data. The simplest situation uses *sliding windows* of fixed size. These types of windows are similar to *first in, first out* data structures. Whenever an element j is observed and inserted into the window, another element $j - w$, where w represents the window size, is forgotten. Computing statistics in the sliding window model requires storing all data inside the window. Exponential histograms [10] are used to approximately compute statistics over sliding windows requiring sublinear space in the size of the window.

Monitoring, analyzing and extracting knowledge from high speed streams might explore multiple levels of granularity, where the recent history is analyzed at fine levels of granularity and the need of precision decreases with the age of the data. As a consequence, the most recent data can be stored at the finest granularity, while more distant data at coarser granularity. This is called the *tilted* time window model. It might be implemented using exponential histograms [10].

2.3 Change detection

Sequence based windows is a general technique to deal with changes in the process that generates data. A reference algorithm is the `AdWin` – `AD`aptive sliding `WIN`dow presented by Bifet and Gavaldà [4]. `AdWin` keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis *there has been no change in the average value inside the window*. More precisely, an older fragment of the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window. This has two consequences: first, that change is reliably declared whenever the window shrinks; and second, that at any time the average over the existing window can be reliably taken as an estimate of the current average in the stream (barring a very small or very recent change that is still not statistically visible). `AdWin` does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique. This means that it keeps a window of length W using only $O(logW)$ memory and $O(logW)$ processing time per item.

2.4 Sampling

Sampling is a common practice for selecting a subset of data to be analyzed. Instead of dealing with an entire data stream, we select instances at periodic intervals. Sampling is used to compute statistics (expected values) of the stream. While sampling methods reduce the amount of data to process, and, by consequence, the computational costs, they can also be a source of errors, namely in monitoring applications that require to detect anomalies or extreme values.

The main problem is to obtain a *representative* sample, that is, a subset of data that has approximately the same properties of the original data. In statistics, most techniques require to know the length of the stream. For data streams, we need to modify these techniques. The simplest form of sampling is *random sampling*, where each element has equal probability of being selected [1]. The *reservoir sampling* technique [31] is the classic algorithm to maintain an online random sample. The base idea consists of maintaining a sample of size k, called the reservoir. As the stream

flows, every new element has a probability k/n, where n is the number of elements seen so far, of replacing an old element in the reservoir.

A similar technique, load shedding, drops sequences in the stream, when bursts cause bottlenecks in the processing capabilities. Tatbul et al. [29] discuss load shedding techniques in querying high-speed data streams.

2.5 Synopsis, sketches and summaries

Synopses are compact data structures that summarize data for further querying. Several methods have been used, including: wavelets [15], exponential histograms [10], frequency moments [3], etc. Data sketching via random projections is a tool for dimensionality reduction. Sketching uses random projections of data points with dimension d to a space of a subset of dimensions. It has been used for moment estimation [3], computing L-norms [24] and dot product of streams [1].

Cormode and Muthukrishnan [8] present a data stream summary, the so called *count-min* sketch, used for (ϵ, δ) approximations to solve *point queries*, *range queries*, and *inner product queries*. Consider an implicit vector $\mathbf{a}$ of dimension n that is incrementally updated over time. At each moment, the element a_i represents the counter associated with element i. A point-query is to estimate the value of an entry in the vector $\mathbf{a}$. The count-min sketch data structure, with parameters (ϵ, δ), is an array of $w \times d$ in size, where $d = log(1/\delta)$, and $w = 2/\epsilon$. For each incoming value of the stream, the algorithm use d hash functions to map entries to $[1, \ldots, w]$. The counters in each row are incremented to reflect the update. From this data structure, we can estimate at any time, the number of occurrences of any item j by taking $min_d CM[d, h_d(j)]$. Since the space used by the sketch CM is typically much smaller than that required to represent the vector $\mathbf{a}$ exactly, there is necessarily some approximation in the estimate. The estimate $\hat{a_j}$, has the following guarantees: $a_j \leq \hat{a_j}$, and, with probability at least $1 - \delta$, $\hat{a_i} \leq a_i + \epsilon ||a||_1$. The error of the estimate is at most ϵ with probability at least $1 - \delta$ in space $O(\frac{1}{\epsilon} log(\frac{1}{\delta}))$.

3 Algorithms for learning from data streams

Data Mining studies the automated acquisition of domain knowledge looking for the improvement of systems performance as result of experience. Data stream mining systems address the problems of data processing, modeling, prediction, clustering, and control in changing and evolving environments. They self-evolve their structure and knowledge on the environment [12]. In this section we address two challenging problems in stream mining that enforce the idea of bounded rationality.

3.1 Mining infinite data

Hulten and Domingos [16] present a general method to learn from arbitrarily large databases. The method consists of deriving an upper bound for the learner's loss as a function of the number of examples used in each step of the algorithm. Then use this to minimize the number of examples required at each step, while guaranteeing that the model produced does not differ significantly from the one that would be obtained with infinite data. This methodology has been successfully applied in k-means clustering [16], hierarchical clustering of variables [26], decision trees [11, 17], regression trees [18], decision rules [14], etc.

The most representative algorithm in this line is the `Very Fast Decision Tree` [11]. `VFDT` is a decision-tree learning algorithm that dynamically adjusts its bias accordingly to the availability of data. In decision tree induction, the main issue is the decision of when to expand the tree, installing a splitting-test and generating new leaves. The basic idea consists of using a small set of examples to select the splitting-test to incorporate in a decision tree node. If after seeing a set of examples, the difference of the merit between the two best splitting-tests does not satisfy a statistical test (the Hoeffding bound), `VFDT` proceeds by examining more examples. It only makes a decision (i.e., adds a splitting-test in that node), when there is enough statistical evidence in favor of a particular test. This strategy guarantees model stability (low variance) and controls overfiting – examples are processed once without the need of model regularization (pruning). This profile is quite different from the standard decision trees model using greedy search and static data sets. Using static datasets, the decisions in deeper nodes of the tree are based on less and less examples. Statistical support decreases as the tree grows, and regularization is mandatory. In VFDT like algorithms the number of examples needed to grow a node is only defined by the statistical significance of the difference between the two best alternatives. Deeper nodes of the tree might require more examples than those used in the root of the tree!

3.2 Mining ubiquitous streams

The phenomenal growth of mobile and embedded devices coupled with their ever-increasing computational and communications capacity presents an exciting new opportunity for real-time, distributed intelligent data analysis in ubiquitous environments. Learning from distributed data, require efficient methods in minimizing the communication overheads between nodes [27]. The strong limitations of centralized solutions is discussed in depth in [20]. The authors point out *a mismatch between the architecture of most off-the-shelf data mining algorithms and the needs of mining systems for distributed applications*. Such mismatch may cause a bottleneck in many emerging applications, namely hardware limitations related to the limited bandwidth channels. Most important, in applications like monitoring, centralized solutions introduce delays in event detection and

reaction, that can make mining systems useless.

Ubiquitous data stream mining implies new requirements to the bounded rationality to be considered [22]: *i)* the algorithms will process local information and *ii)* need to communicate with other agents in order to infer the global learning context; *iii)* the budget for communications (bandwidth, battery) are limited.

Typical applications include clustering sensor networks [25], autonomous vehicles [30], analysis of large social networks [19], multiple classification models exploring distributed processing [7], etc.

4 Concluding remarks

There is a fundamental difference between learning from small datasets and streaming data: mining data streams is a continuous process. This observation opens the ability to monitor the evolution of the learning process; to use change detection mechanisms to self-diagnosis the evolution of this process, and to react and repair decision models [13]. Continuous learning, forgetting, self-adaptation, and self-reaction are main characteristics of any intelligent system. They are characteristic properties of stream learning algorithms. From a bias-variance analysis there is a fundamental difference [5]: while learning from small data sets requires an emphasis in variance reduction, learning from large data sets is more effective when using algorithms that place greater emphasis on bias management.

Bounded rationality is in the core of next generation of data mining systems. Learning algorithms must be able to adapt continuously to changing environmental conditions (including their own condition) and evolving user needs. Learning must consider the real-time constrains of limited computing power and communication resources.

Acknowledgment

This work is funded by the ERDF through the Programme COMPETE PEst-C/SAU/UI0753/2011 and by FCT project PTDC/EIA/098355/2008, *Knowledge Discovery from Ubiquitous Data Streams*.

References

[1] Aggarwal, C. (2006). On biased reservoir sampling in the presence of stream evolution. In *Proceedings International Conference on Very Large Data Bases*, Seoul, Korea, pp. 607–618. ACM.

[2] Aggarwal, C. (Ed.) (2007). *Data Streams – Models and Algorithms*. Springer.

[3] Alon, N., Y. Matias, and M. Szegedy (1999). The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences 58*, 137–147.

[4] Bifet, A. and R. Gavaldà (2006). Kalman filters and adaptive windows for learning in data streams. In *Proceedings of the 9th Discovery Science*, Volume 4265 of *Lecture Notes Artificial Intelligence*, Barcelona, Spain, pp. 29–40. Springer.

[5] Brain, D. and G. Webb (2002). The need for low bias algorithms in classification learning from large data sets. In *Principles of Data Mining and Knowledge Discovery PKDD*, Volume 2431 of *Lecture Notes in Artificial Intelligence*, Helsinki, Finland, pp. 62–73. Springer.

[6] Chakrabarti, A., K. D. Ba, and S. Muthukrishnan (2006). Estimating entropy and entropy norm on data streams. In *STACS: 23rd Annual Symposium on Theoretical Aspects of Computer Science*, Marseille, France, pp. 196–205.

[7] Chen, R., K. Sivakumar, and H. Kargupta (2004). Collective mining of Bayesian networks from heterogeneous data. *Knowledge and Information Systems Journal 6*(2), 164–187.

[8] Cormode, G. and S. Muthukrishnan (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms 55*(1), 58–75.

[9] Cormode, G., S. Muthukrishnan, and W. Zhuang (2007). Conquering the divide: Continuous clustering of distributed data streams. In *ICDE: Proceedings of the International Conference on Data Engineering*, Istanbul, Turkey, pp. 1036–1045.

[10] Datar, M., A. Gionis, P. Indyk, and R. Motwani (2002). Maintaining stream statistics over sliding windows. In *Proceedings of Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, USA, pp. 635–644. Society for Industrial and Applied Mathematics.

[11] Domingos, P. and G. Hulten (2000). Mining High-Speed Data Streams. In *Proceedings of the ACM Sixth International Conference on Knowledge Discovery and Data Mining*, Boston, USA, pp. 71–80. ACM Press.

[12] Gama, J. (2010). *KnowledgeDiscovery from Data Streams*. Data Mining and Knowledge Discovery. Atlanta, US: Chapman & Hall CRC Press.

[13] Gama, J. and P. Kosina (2011a). Learning about the learning process. In *Advances in Intelligent Data Analysis*, Volume 7014 of *Lecture Notes in Computer Science*, pp. 162–172. Springer.

[14] Gama, J. and P. Kosina (2011b). Learning decision rules from data streams. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, pp. 1255–1260.

[15] Gilbert, A. C., Y. Kotidis, S. Muthukrishnan, and M. Strauss (2001). Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, Rome, Italy, pp. 79–88.

[16] Hulten, G. and P. Domingos (2001). Catching up with the data: research issues in mining data streams. In *Proc. of Workshop on Research Issues in Data Mining and Knowledge Discovery*, Santa Barbara, USA.

[17] Hulten, G., L. Spencer, and P. Domingos (2001). Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, pp. 97–106. ACM Press.

[18] Ikonomovska, E., J. Gama, and S. Dzeroski (2011). Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery 23*, 128–168.

[19] Kang, U., C. E. Tsourakakis, and C. Faloutsos (2011). Pegasus: mining peta-scale graphs. *Knowl. Inf. Syst. 27*(2), 303–325.

[20] Kargupta, H., A. Joshi, K. Sivakumar, and Y. Yesha (2004). *Data Mining: Next Generation Challenges and Future Directions*. AAAI Press and MIT Press.

[21] Manku, G. S. and R. Motwani (2002). Approximate frequency counts over data streams. In *Proceedings of 28th International Conference on Very Large Data Bases*, Hong Kong, pp. 346–357. Morgan Kaufmann.

[22] May, M. and L. Saitta (Eds.) (2010). *Ubiquitous Knowledge Discovery*. LNAI 6202, Springer.

[23] Motwani, R. and P. Raghavan (1997). *Randomized Algorithms*. Cambridge University Press.

[24] Muthukrishnan, S. (2005). *Data Streams: Algorithms and Applications*. Now Publishers.

[25] Rodrigues, P. P., J. Gama, and L. Lopes (2009). Knowledge discovery for sensor network comprehension. In *Intelligent Techniques for Warehousing and Mining Sensor Network Data*, pp. 118–134. Information Science.

[26] Rodrigues, P. P., J. Gama, and J. P. Pedroso (2008). Hierarchical clustering of time series data streams. *IEEE Transactions on Knowledge and Data Engineering 20*(5), 615–627.

[27] Sharfman, I., A. Schuster, and D. Keren (2007). A geometric approach to monitoring threshold functions over distributed data streams. *ACM Transactions Database Systems 32*(4), 301–312.

[28] Simon, H. (1957). *Models of Man*. John Wiley.

[29] Tatbul, N., U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker (2003). Load shedding in a data stream manager. In *Proceedings of the International Conference on Very Large Data Bases*, Berlin, Germany, pp. 309–320. VLDB Endowment.

[30] Thrun, S. (2010). Toward robotic cars. *Communications ACM 53*(4), 99–106.

[31] Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software 11*(1), 37–57.