A General Framework for Mining Massive Data Streams

Pedro Domingos **Geoff Hulten**
Department of Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98185-2350, U.S.A.
{pedrod,ghulten}@cs.washington.edu

Abstract

In many domains, data now arrives faster than we are able to mine it. To avoid wasting this data, we must switch from the traditional "one-shot" data mining approach to systems that are able to mine continuous, high-volume, open-ended data streams as they arrive. In this extended abstract we identify some desiderata for such systems, and outline our framework for realizing them. A key property of our approach is that it minimizes the time required to build a model on a stream, while guaranteeing (as long as the data is i.i.d.) that the model learned is effectively indistinguishable from the one that would be obtained using infinite data. Using this framework, we have successfully adapted several learning algorithms to massive data streams, including decision tree induction, Bayesian network learning, k-means clustering, and the EM algorithm for mixtures of Gaussians. These algorithms are able to process on the order of billions of examples per day using off-the-shelf hardware. Building on this, we are currently developing software primitives for scaling arbitrary learning algorithms to massive data streams with minimal effort.

1 The Problem

Many (or most) organizations today produce an electronic record of essentially every transaction they are involved in. When the organization is large, this results in tens or hundreds of millions of records being produced every day. For example, in a single day WalMart records 20 million sales transactions, Google handles 150 million searches, and AT&T produces 275 million call records. Scientific data collection (e.g., by earth sensing satellites or astronomical observatories) routinely produces gigabytes of data per day. Data rates of this level have significant consequences for data mining. For one, a few months' worth of data can easily add up to billions of records, and the entire history of transactions or observations can be in the hundreds of billions. Current algorithms for mining complex models from data (e.g., decision trees, sets of rules) cannot mine even a fraction of this data in useful time.

Further, mining a day's worth of data can take more than a day of CPU time, and so data accumulates faster than it can be mined. As a result, despite all our efforts in scaling up mining algorithms, in many areas the fraction of the available data that we are able to mine in useful time is rapidly dwindling towards zero. Overcoming this state of affairs requires a shift in our frame of mind from mining databases to mining data streams. In the traditional data mining process, the data to be mined is assumed to have been loaded into a stable, infrequently-updated database, and mining it can then take weeks or months, after which the results are deployed and a new cycle begins. In a process better suited to mining the high-volume, open-ended data streams we see today, the data mining system should be continuously on, processing records at the speed they arrive, incorporating them into the model it is building even if it never sees them again. A system capable of doing this needs to meet a number of stringent design criteria:

- It must require small constant time per record, otherwise it will inevitably fall behind the data, sooner or later.
- It must use only a fixed amount of main memory, irrespective of the total number of records it has seen.
- It must be able to build a model using at most one scan of the data, since it may not have time to revisit old records, and the data may not even all be available in secondary storage at a future point in time.
- It must make a usable model available at any point in time, as opposed to only when it is done processing the data, since it may never be done processing.
- Ideally, it should produce a model that is equivalent (or nearly identical) to the one that would be obtained by the corresponding ordinary database mining algorithm, operating without the above constraints.
- When the data-generating phenomenon is changing over time (i.e., when concept drift is present), the model at any time should be up-to-date, but also include all information from the past that has not become outdated.

At first sight, it may seem unlikely that all these constraints can be satisfied simultaneously. However, we have developed a general framework for mining massive data streams that satisfies all six [5]. Within this framework, we have designed and implemented massive-stream versions of decision tree induction [1, 6], Bayesian network learning [5], k-means clustering [2] and the EM algorithm for mixtures of Gaussians [3]. For example, our decision tree learner, called VFDT, is able to mine on the order of a billion examples per day using off-the-shelf hardware, while providing strong guarantees that its output is very similar to that of a "batch" decision tree learner with access to unlimited resources. We are currently developing a toolkit to allow implementation of arbitrary stream mining algorithms with no more effort than would be required to implement ordinary learners. The goal is to automatically achieve the six desiderata above by using the primitives we provide and following a few simple guidelines. More specifically, our framework helps to answer two key questions:

- How much data is enough? Even if we have (conceptually) infinite data available, it may be the case that we do not need all of it to obtain the best possible model of the type being mined. Assuming the data-generating process is stationary, is there some point at which we can "turn off" the stream and know that we will not lose predictive performance by ignoring further data? More precisely, how much data do we need at each step of the mining algorithm before we can go on to the next one?

- If the data-generating process is not stationary, how do we make the trade-off between being up-to-date and not losing past information that is still relevant? In the traditional method of mining a sliding window of data, a large window leads to slow adaptation, but a small one leads to loss of relevant information and overly-simple models. Can we overcome this trade-off?

In the remainder of this extended abstract we describe how our framework addresses these questions. Further aspects of the framework are described in Hulten and Domingos (2002).

2 The Framework

A number of well-known results in statistics provide probabilistic bounds on the difference between the true value of a parameter and its empirical estimate from finite data. For example, consider a real-valued random variable x whose range is R. Suppose we have made n independent observations of this variable, and computed their mean $\bar{x}$. The Hoeffding bound [4] (also known as additive Chernoff bound) states that, with probability at least $1-\delta$, and irrespective of the true distribution of x, the true mean of the variable is within ϵ of $\bar{x}$, where

$$\epsilon = \sqrt{\frac{R^2 \ln(2/\delta)}{2n}}$$

Put another way, this result says that, if we only care about determining x to within ϵ of its true value, and are willing to accept a probability of δ of failing to do so, we need gather only $n = \frac{1}{2}(R/\epsilon)^2 \log(2/\delta)$ samples of x. More samples (up to infinity) produce in essence an equivalent result. The key idea underlying our framework is to "bootstrap" these results, which apply to individual parameters, to similar guarantees on the difference (loss) between the whole complex model mined from finite data and the model that would be obtained from infinite data in infinite time. The high-level approach we use consists of three steps:

1. Derive an upper bound on the time complexity of the mining algorithm, as a function of the number of samples used in each step.

2. Derive a upper bound on the relative loss between the finite-data and infinite-data models, as a function of the number of samples used in each step of the finite-data algorithm.

3. Minimize the time bound (via the number of samples used in each step) subject to user-defined limits on the loss.

Where successful, this approach effectively allows us to mine infinite data in finite time, "keeping up" with the data no matter how much of it arrives. At each step of the algorithm, we use only as much data from the stream as required to preserve the desired global loss guarantees. Thus the model is built as fast as possible, subject to the loss targets. The tighter the loss bounds used, the more efficient the resulting algorithm will be. (In practice, normal bounds yield faster results than Hoeffding bounds, and their general use is justifiable by the central limit theorem.) Each data point is used at most once, typically to update the sufficient statistics used by the algorithm. The number of such statistics is generally only a function of the model class being considered, and is independent of the quantity of data already seen. Thus the memory required to store them, and the time required to update them with a single example, are also independent of the data size.

When estimating models with continuous parameters (e.g., mixtures of Gaussians), the above procedure yields a probabilistic bound on the difference between the parameters estimated with finite and infinite data. (By "probabilistic," we mean a bound that holds with some confidence $1 - \delta^*$, where δ^* is user-specified. The lower the δ^*, the more data is required.) When building models based on discrete decisions (e.g., decision trees, Bayesian network structures), a simple general bound can be obtained as follows. At each search step (e.g., each choice of split in a decision tree), use enough data to ensure that the probability of making the wrong choice is at most δ. If at most d decisions are made during the search, each among at most b alternatives, and c checks for the winner are made during each step, by the union bound the probability that the total model produced differs from what would be produced with infinite data is at most $\delta^* = bcd\delta$. For specific algorithms and with additional assumptions, it may be possible to obtain tighter bounds (see, for example, Domingos & Hulten (2000)).

3 Time-Changing Data

The framework just described assumes that examples are i.i.d. (independent and identically distributed). However, in many data streams of interest this is not the case; rather, the data-generating process evolves over time. Our framework handles time-changing phenomena by allowing examples to be forgotten as well as remembered. Forgetting an example involves subtracting it from the sufficient statistics it was previously used to compute. When there is no drift, new examples are statistically equivalent to the old ones and the mined model does not change, but if there is drift a new best decision at some search point may surface. For example, in the case of decision tree induction, an alternate split may now be best. In this case we begin to grow an alternative subtree using the new best split, and replace the old subtree with the new one when the latter becomes more accurate on new data. Replacing the old subtree with the new node right away would produce a result similar to windowing, but at a constant cost per new example, as opposed to $O(w)$, where w is the size of the window. Waiting until the new subtree becomes more accurate ensures that past information continues to be used for as long as it is useful, and to some degree overcomes the trade-off implicit in the choice of window size. However, for very rapidly changing data the pure windowing method may still produce better results (assuming it has time to compute them before they become outdated, which may not be the case). An open direction of research that we are

beginning to pursue is to allow the "equivalent window size" (i.e., the number of time steps that an example is remembered for) to be controlled by an external variable or function that the user believes correlates with the speed of change of the underlying phenomenon. As the speed of change increases the window shrinks, and vice-versa. Further research involves explicitly modeling different types of drift (e.g., cyclical phenomena, or effects of the order in which data is gathered), and identifying optimal model updating and management policies for them. Example weighting (instead of "all or none" windowing) and subsampling methods that approximate it are also relevant areas for research.

4 Conclusion

In many domains, the massive data streams available today make it possible to build more intricate (and thus potentially more accurate) models than ever before, but this is precluded by the sheer computational cost of model-building; paradoxically, only the simplest models are mined from these streams, because only they can be mined fast enough. Alternatively, complex methods are applied to small subsets of the data. The result (we suspect) is often wasted data and outdated models. In this extended abstract we outlined some desiderata for data mining systems that able to "keep up" with these massive data streams, and some elements of our framework for achieving them. A more complete description of our approach can be found in the references below.

References

[1] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, Boston, MA, 2000. ACM Press.

[2] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 106–113, Williamstown, MA, 2001. Morgan Kaufmann.

[3] P. Domingos and G. Hulten. Learning from infinite data in finite time. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 673–680. MIT Press, Cambridge, MA, 2002.

[4] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[5] G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 525–531, Edmonton, Canada, 2002. ACM Press.

[6] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001. ACM Press.