

MINING HIGH-SPEED DATA STREAMS

Authors:

(1) Pedro Domingos
University of Washington
Seattle, WA 98195-2350, U.S.A.

(2) Geoff Hulten
University of Washington
Seattle, WA 98195-2350, U.S.A

Students:

- (1) Fredrick Bihan**
- (2) Lars Eriksson**
- (3) Vinod Kulkarni**

Abstract:

Many organizations such as Wall Mart, K Mart, Sears etc. have very large databases that grow without limit at a rate of several million records per day. Mining these continuous data streams brings unique opportunities but also new challenges. This paper talks about Very Fast Decision Trees (VFDT) that builds decision trees that can incorporate tens of thousands of examples per second using off-the shelf hardware!

Introduction:

Knowledge based systems are constrained by three main limited resources:

- Time
- Memory
- Sample Size

In traditional applications of machine learning and statistics, sample size tends to be the dominant limitation. Computational resources for a massive search are available, but carrying out such a search over the small samples available often leads to “Over Fitting”.

In today’s data mining applications the bottleneck is time and memory, not examples. The latter is typically in over supply and it is impossible with current KDD (Knowledge based discovery and Data mining) systems to make use of all of them within the available computational resources. As a result most of the examples go unused resulting in “Under Fitting”.

Problems:

The most efficient algorithms available today, concentrate on making it possible to mine databases that do not fit in main memory by only requiring sequential scans of the disk. But even these algorithms have only been tested on up to a few million examples. In many cases this is less than a days worth of data. For e.g. Everyday Wall Marts across the country record millions of transactions! Other examples would involve telecommunications companies, banks, credit card companies, etc.

When the source of examples is an open-ended data stream, the notion of mining a database with fixed size itself becomes questionable.

Solutions:

Ideally, we would like present day KDD systems to operate continuously and indefinitely, incorporating examples as they arrive. Such desiderata are fulfilled by “Incremental Learning Methods”. However, the available algorithms of this type do not guarantee that the model learned would be similar to the one obtained by learning on the same data in “Batch Mode”. Even if they do, it would be at a high cost in efficiency.

Proposal:

The authors propose Hoeffding trees, a decision-tree learning method that overcomes this trade-off. Hoeffding trees can be learned in constant time per example, while being nearly identical to the trees a conventional batch learner would produce, given enough examples.

The probability that the Hoeffding and conventional tree learners will choose different tests at any given node decreases exponentially with the number of examples. VFDT, a decision-tree learning system based on Hoeffding trees, mines examples in less time than it takes to input them from the disk! It does not store any examples in the main memory, requiring space proportional to the size of the tree and associated statistics.

HOEFFDING Trees:

(1) Definitions:

A classification problem is generally defined as follows:

- i. \underline{N} is a set of training examples of the form (x, y) , where
- ii. \underline{y} is a discrete class label
- iii. \underline{x} is a vector of \underline{d} attributes.

(2) Goal:

To produce from the examples a model $y = f(x)$ that will predict the classes y for future examples x with high accuracy.

(3) Why Statistical Rules?

1. Classical Decision tree learners like ID3, C4.5, CART etc. assume that all training examples can be stored simultaneously in main memory and are thus severely limited in the number of examples they can learn from.
2. Disk based Decision tree learners like SLIQ and SPRINT assume examples stored on the disk and learn by repeatedly reading them in sequentially. While this increases the size of the training sets, it can become prohibitively expensive when learning complex trees and fails when datasets are too large to fit in the available disk space.
3. Hence the goal is to design a decision tree learner from extremely large (potentially infinite) datasets. This learner should require each example to be read at most once and only a small constant time to process it. This makes it possible to mine online data sources and to build potentially very complex trees with acceptable computational costs.

Hoeffding Bound:

In order to find the best attribute to test at a given node, it may be sufficient to consider only a small subset of training examples that pass through the node. Thus, given a stream of examples, the first ones will be used to choose the root test; once the root attribute is chosen, the succeeding examples will be passed down to the corresponding leaves and used to choose the appropriate attributes there and so on recursively.

Using a statistical method solves the difficult problem of deciding exactly how many examples are necessary at each node. This method is called the “Hoeffding Bound”.

Consider a random variable “a” whose range is “R” (e.g. for a probability the range is one). Suppose we have made “n” observations of this variable, and computed their mean \bar{a} . The Hoeffding bound states that, with probability $1 - \delta$, the true mean of the variable is at least $\bar{a} - \epsilon$, where

$$\epsilon = R * \sqrt{(\ln (1/\delta) / 2n)}$$

Let $G(X_i)$ be the heuristic measure used to choose test attributes (e.g. the measure could be Information Gain). The goal is to ensure that, with a high probability, that attribute chosen using n examples (where n is as small as possible) is the same that would be chosen using infinite examples.

Let X_a be the attribute with the highest observed \hat{G} after seeing n examples and, and X_b be the second highest attribute.

Let $\Delta \hat{G} = \hat{G}(X_a) - \hat{G}(X_b) \geq 0$ be the difference between the observed heuristic values. Then, given a desired δ , the Hoeffding bound guarantees that X_a is the correct choice with probability $1 - \delta$ if n examples have been seen at this node and

$$\Delta \hat{G} > \epsilon^2 \quad \text{----- (1)}$$

In other words, if the observed

$$\Delta \hat{G} > \epsilon \quad \text{----- (2)}$$

then the Hoeffding bound guarantees that the true $\Delta G \geq \Delta \hat{G} - \epsilon > 0$ with probability $1 - \delta$, and therefore that X_a is indeed the best attribute with probability $1 - \delta$.

Thus a node needs to accumulate examples from the stream until ϵ becomes smaller than $\Delta \hat{G}$. (ϵ is a monotonically decreasing function of n.) At this point the node can be split using the current best attribute, and succeeding examples will be passed to the new leaves.

Algorithm:

The Hoeffding tree algorithm is based on the Decision Tree (ID3) algorithm. For the pseudo code see the appendix.

Pre- pruning is carried out by considering at each node a NULL attribute X_0 that consists of not splitting the node. Thus a split will only be made if, with confidence $1 - \delta$, the best split found is better according to G than not splitting.

X_0 will determine the leaf nodes.

The algorithm constructs the tree using the same procedure as ID3. It calculates the information gain for the attributes and determines the best two attributes. At each node it

checks for condition 2 (above). If the condition is satisfied then it creates child nodes based on the test at the node. If not it streams in more training examples and carries out the calculations till it satisfies the condition.

If d is the number of attributes, v is the maximum number of values per attribute, and c is the number of classes, the Hoeffding tree algorithm requires $O(dvc)$ memory to store the necessary counts at each leaf. If l is the number of leaves in the tree, the total memory required is $O(ldvc)$.

VFDT is a decision tree learning system based on the Hoeffding tree algorithm.

The VFDT system

The VFDT system is based on the Hoeffding Tree algorithm seen above : it uses either the information gain or Gini index as attribute evaluation measure and allow the user to specify some parameters as :

Ties : Two attributes with very close G ' s will lead to examination of a large number of examples to determine the best one. So the user can define a threshold under which a split will occur.

G computation : This computation is the most time consuming part of the algorithm and it makes sense that just one example will not dramatically change the G . So, the user can specify a number n_{\min} of new examples before recompilation.

Memory and poor attribute : The VFDT system leads to minimize memory usage using two techniques :

- deactivation of non-promising leaf
- dropping of non-promising attribute.

This allows the system to keep memory available for new leaf.

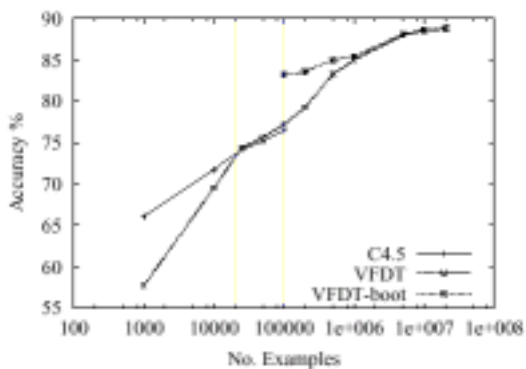
Initialization : The system can be initialized with a previously computed tree. It allows it to begin with comparable accuracy as classic decision tree learner.

Study and comparison

To be interesting, VFDT should at least give results comparable to conventional decision tree learners and then should take advantage of all the extra examples it can use to go further than other algorithm.

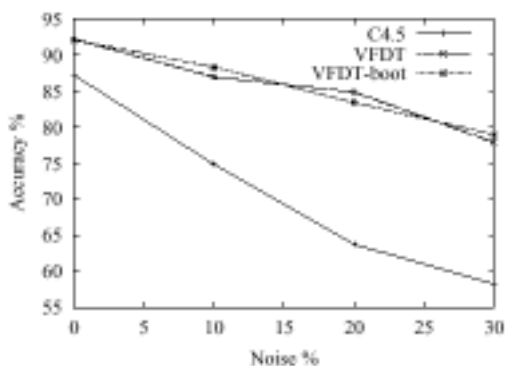
This is the reason why, VFDT has been compared to C4.5 release 8 on a series of generated datasets. These datasets have been created by sampling random trees (depth=18 and between 2.2k to 61k leaves) and adding noise, from 0 to 30%, to it.

This study will so compare C4.5, VFDT and VFDT-boot, a VFDT system bootstrapped with an over-pruned tree produced by C4.5.



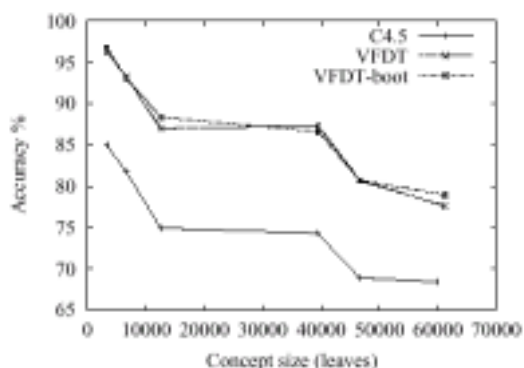
This figure shows here accuracy as a function of the number of examples.

It appears here, that C4.5 is more accurate than VFDT up to 25k examples while they are similar between 25k and 100k. C4.5 cannot process examples after 100k and this is where VFDT take a real advantage to increase significantly its accuracy (88.8 % for VFDT against 76.5% for C4.5).



The following figure shows the response to the different systems to noise.

Here, the difference between C4.5 and the VFDT based systems increase as the noise level increase : this result is another proof that the Hoeffding bound is an effective pruning method.



Another important aspect is the accuracy as a function of the concept size, which is illustrated by this figure. In this case the computation has been made with 100k examples for C4.5 and 20 millions for VFDT based systems.

It appears here that the difference between the systems remain the same but VFDT induces trees with a constant number of node regardless of the size which leads to think that VFDT should take advantages of even more examples.

Parameters importance

Another series have been run to evaluate the importance of the different parameters in the VFDT system.

- The first one was testing the Tie feature and it appears that for VFDT as well as for VFDT-boot, the accuracy was higher with ties but it also generate much more nodes.
- Testing n , the number of new examples before to recomputed G , the two VFDT based systems took 4 times longer to run for a gain of 1.1% for VFDT and a loss of 0.9% for VFDT-boot.
- The last two runs were made changing δ only. These two runs shown that with a lower δ , the system generate 30% more node for a gain of 1 to 2.3%

A real world example

The authors have made a real world study. They tried to mine all the web page requests that were made from the University of Washington main campus during a week in May 1999. The estimated population of the University is 50,000 people (students, faculty and staff), during this week they registered 23,000 active Internet clients. The traced requests summed up to 82.2 million in the end of the week and the peak rate of which they were received was 17,400 per minute. The size of the trace file was around 20 GB.

To be able to make a decision tree from this kind of data each request was tagged with an anonymous ID, representing one of the 170 organizations on campus (colleges, departments, etc.). They then split the request log into a series of equally long time slice T_0, \dots, T_t, \dots the time slices were set to an hour. Each organization was labeled O_1, \dots, O_{170} and each of the 244,000 hosts that were requested H_1, \dots, H_{244k} . Count was kept for how many times an organization requested a host in the time slice, C_{ijt} . These counts were the generalized into four buckets, “no request”, “1-12 requests”, “13-25 requests” and “26 or more requests”. Then for each time slice and host accessed in the time slice (T_t, H_j) an example with attributes $t \bmod 24$, $C_{1,jt}, \dots, C_{jt}, \dots, C_{170,jt}$ was generated. Class 1 if H_j is requested in time slice T_{t+1} and 0 if it is not.

This example generation can be carried out in real time by only keeping statistics on the last an current time slice, C_{t-1} and C_t , in memory. Then outputting C_{t-1} as soon as C_t is complete. By using this procedure the authors could obtain a dataset of 1.89 million examples of which 61.1% were labeled with the most common class (that the host was not requested in the next time slice).

Testing was carried out on the last day’s log. The VFDT was run on 1.61 million examples and took 1277 second to learn a decision stump (a decision tree with only one node). They also ran the C4.5 algorithm, but then they could only use 74.5k examples (what fits in 40 MB of memory). It took the C4.5 2975 seconds to learn the tree. To see that the difference in performance would differ even more with larger datasets they used a machine with 1 GB of RAM. With this machine they could fit the 1.61 million examples in the memory to run it with the C4.5. The run time now increased to 24 hours.

We can clearly see that the VFDT much faster than the C4.5 and that it can achieve similar accuracy in a fraction of time. Mining on larger log data then one week is not possible on the C4.5 while the VFDT can incorporate infinite data.

The next step would be to apply VFDT to predict page request from a given host.

Related work

There has been quite a lot of previous work related to mining databases using sub sampling.

Maron and Moore used Hoeffding bounds to speed selection of instance-based regression models via cross-validation.

Gratch's sequential ID3 used another statistical method to minimize the number of examples needed to choose each split in a decision tree. However, this method's guarantees of similarity were much looser than those derived from the Hoeffding trees.

The system most closely related to the VFDT is Utgoff's ID5R. The ID5R learns the same tree as the ID3, by reconstructing sub trees as needed. The problem with this system is the fact that it is much slower than the ID3 on already "nice" data. Presumably, noise data would aggravate this already long run time. Thus ID5R is not suitable for learning from high-speed data streams.

The VFDT combines the best of both worlds, accessing data sequentially and using subsampling to potentially require much less than one scan of the data. Additionally, VFDT has the advantage of being incremental and anytime. This meaning; new examples can be quickly incorporated as they arrive. Further more, a usable model is available after the first few examples and then progressively refined.

Future work

There are still many things to do that can improve the VFDT's efficiency. One of the most obvious is to optimize the computation of the Hoeffding bound. In other words, to have the program only re-computing the G's exactly when the current example may cause the Hoeffding bound to be reached. Comparisons with ID5R and other incremental algorithms are another thing that needs to be done.

Conclusion

Many organizations today have more than very large databases; they have databases that grow without limit at a rate of several million records per day.

This paper introduces Hoeffding trees and VFDT-system. VFDT is an anytime system that builds decision trees using constant memory and constant time per example.

It uses Hoeffding bounds to guarantee that its output is asymptotically nearly identical to that of a conventional learner. Further more, empirical studies show its effectiveness in learning from massive and continuous streams of data.

VFDT is currently being applied to mining the continuous stream of Web access data from the whole University of Washington main campus.

Opinion

We have decided to grade the paper in two aspects, the paper as a whole and the algorithm. We think that the algorithm is very good. Results clearly show that this is the fastest way of producing asymptotically similar decision trees. The fact that the authors have implemented the algorithm and made a data mining system out of it is also very good. That shows us that this works in practice and not only in theory. On a scale from 1-10 we rate the algorithm 8.

The paper is good. However, we are missing some parts. First of all we think that an inclusion of an example would be a good thing in order to get the reader to more easily understand the algorithm. However, we can also understand the problem with this since it will take quite a few examples to actually come to the point where the first split is carried out.

Secondly, the authors take the reader's knowledge in statistics for granted. The thing we would have liked them to explain more thoroughly is Hoeffding Bound. They have, of course, references to where this information can be obtained, but since this is such an important part of the paper we think it needs some more attention. The nature of the measure is not explained anywhere. To sum it up, we give the paper a 7 on the scale 1-10.