A benchmark for histogram building algorithms of numeric streams

Stefan Sebastian, 242

Contents

1 Introduction

Main goals Motivation

1.1 Histograms

Short description of histograms use + why streaming

//todo reformulate, benahim2.1 A histogram is a set of B pairs (called bins) of real numbers (p1,m1),...,(pB,mB), where B is a preset constant integer. The histogram is a compressed and approximate representation of a set S of real numbers

Paper structure

2 Algorithms

3 Numeric Histogram

3.1 Overview

The Numeric Histogram algorithm first appeared in the paper by Ben-Haim and Tom-Tov on building a streaming parallel decision tree algorithm[1]. The main focus was to describe a method for tree based classifiers for large data sets in a distributed environment, however they needed a data structure that can summarize large amounts of data accurately. Thus, they proposed a histogram data structure that can adapt to the requirements of a streaming environment.

The histogram maintains a fixed number of bins of the shape (p, m) where p represents a central value of the interval and m the number of points in it. Initially the histogram has some bins allocated but the values are not known. They are filled as data comes is added into the structure and once the allocated number of bins is reached the central bin values are updated depending on incoming values. In this regard, the algorithm is robust to changes in data patterns over a longer period of time, which is one of the main concerns of streaming analysis algorithms.

3.2 Procedures

The proposed data structure contains four procedures: update, merge, sum and uniform. But for the purpose of this benchmark only two were needed: update and sum. The update procedure adds a new point to the histogram data structure and is described in algorithm 1.

The sum procedure, depicted in algorithm 2, obtains the estimated number of points between $(-\infty, n]$ where n is the input value. In order to estimate the number of values in a range $[a, b]$ we can calculate $sum(b) - sum(a)$. The algorithm assumes that for each bin (p, m), there are $m/2$ points to the left of p and $m/2$ to the right. This means that the number of points in the interval $[p_i, i_{i+1}]$ is equal to $(m_i + m_{i+1})/2$, which is the area of the trapezoid $(p_i, 0), (p_i, m_i), (p_{i+1}, m_{i+1}), (p_{i+1}, 0)$, divided by $p_{i+1} - p_i$. Similarly, we can estimate the number of points in the interval $[p_i, p]$ by adding calculating its projection on the line from (p_i, m_i) to (p_{i+1}, m_{i+1}), then find the area of the new trapezoid and dividing again by $p_{i+1} - p_i$. The algorithm will not work properly if the point p is smaller than

Data: histogram $h = (p_1, m_1), .., (p_b, m_b)$, a point p
Result: a histogram that represents the set $S \bigcup \{p\}$
binary search for the closest p_i larger than p;
if $p_i = p$ **then**
   | $m_i = m_i + 1$;
else
   | add a new bin of shape (p, 1) to the histogram at the i-th position;
   | find the closest two bins by their p values;
   | merge those bins, moving p proportional to their m values;
end

<div align="center">

Algorithm 1: add procedure

</div>

p_0 or larger than p_b. For this reason the data structure should be initialized with a lower and upper bound into which all incoming data should fit.

Data: a histogram h, a point p such that $p_1 < p < p_b$
Result: estimated number of points in the interval $[-\infty, p]$
binary search to find i such that $p_i <= p <= p_{i+1}$;
set $s = (m_i + m_p) \cdot (b - p_i)/2 \cdot (p_{i+1} - p_i)$;
where $m_p = m_i + (m_{i+1} - m_i) \cdot (p - p_i)/(p_{i+1} - p_i)$;
for $j < i$ **do**
   | $s = s + m_j$;
end
$s = s + m_i/2$;

<div align="center">

Algorithm 2: sum procedure

</div>

3.3 In practice

This algorithm was adapted and implemented into the open source project Apache Hive[2], which is a datawarehouse solution built on top of Hadoop to provide data query and analysis methods. The project is used actively in industry, handling reporting tasks for large volumes of data, for e.g. data produced by 435M monthly users of Chitika[3]. The version of the algorithm presented in this report is based on the Hive implementation, called NumericHistogram[4]. The optimal number of bins is left as a choice to the user, however the suggested range is between 20 and 80.

3.4 Optimal Streaming Histograms

3.5 Overview

This algorithm was presented in a blog post[5] by an engineer from the Amplitude company. Therefore, it was developed with industry requirements in mind, and tested in realistic conditions before being published. The main challenge was to avoid storing gigantic amounts of streaming data and to uncover an optimal bucketing solution. The key requirements they identified for the bucket boundaries were: to be useful and reasonable for any range of data and to remain useful upon changes in data distribution.

The algorithm was also designed with data visualization needs in mind. For this reason the buckets size and spacing have been thought out to look intuitive in a chart and be easy to interpret.

3.6 Iterations

In order to reach a satisfying solution they went through multiple iterations of the algorithm. The first one was to save the first 1000 distinct values on the stream and then make evenly spaced buckets using them. However, this method behaves poorly when distribution changes over time because the buckets are fixed. Also, this might have resolution issues when the distribution is skewed, for example: a lot of points in a small range of values.

The second technique was iterative merging of the closest values into buckets. This was done in order to solve the resolution problem. Basically, among the first 1000 values, the 2 closest ones are merged into a bucket. This process is repeated until only 50 buckets are left. This improves resolution, however it causes difficulties in interpreting the histograms as all buckets have different widths and the spacing seems arbitrary.

3.7 Algorithm

The final form of the algorithm is to pre-create buckets on a logarithmic scale. This requires the user to know the boundaries of the incoming data, and then, using those bounds, to create buckets that have 10% increments for every order of magnitude. For example, for the range 1 - 10, there will be buckets of size 0.1, for the range 10 - 100, of size 1 and so forth. This solves the resolution issues because each value will be in a bucket in a 10% range of its true value. The spacing issues are also solved, and data visualization is easier and more intuitive. The fixed size of the buckets should not be a problem in this scheme, which should work with a variety of data set types.

3.8 Implementation

The blog post did not provide an implementation so a variant will be proposed in this report. The initialization step is done with a given lower and upper bound. The smallest power of 10 larger than the upper bound and the largest power of 10 smaller than the lower bound are found and for each value p between them 90 buckets of width $p/10$ are created.

For the add procedure, the bucket is found using binary search which provides a logarithmic complexity. The bin is the incremented by 1 for each addition. In order to estimate3 the number of values in a range $[a, b]$ two binary searches are performed, for the bin of a and the bin of b. We need to estimate the number of values in the range $[a, bUb]$, where bUb is the upper bound of the bucket which contains a, and in the range $[bLb, b]$ where bLb is the lower bound of the bucket which contains b. The approach is similar for both intervals, so only one will be described. In order to estimate the number of values in $[a, bUb]$, a ratio is calculated between the distance from a to the upper bound and the size of the bucket. This ratio is then multiplied with the number of values in the bucket. The values for the buckets contained in the range $[bUb, bLb]$ are added as they are.

Data: a histogram h, two points a, b
Result: estimated number of points in the interval $[a, b]$
estimate = 0;
bin1 = getBin(a), bin2 = getBin(b);
$lowerBound = bin1.val$, $upperBound = lowerBound \cdot 10$;
$ratio = (upperBound - a)/(upperBound - lowerBound)$;
$estimate = estimate + ratio \cdot bin1.count$;
for *each bin from bin1 to bin2* **do**
  | $estimate = estimate + bin.count$
end
$lowerBound = bin2.val$, $upperBound = lowerBound \cdot 10$;
$ratio = (b - lowerbound)/(upperBound - lowerBound)$;
$estimate = estimate + ratio \cdot bin2.count$;

Algorithm 3: Estimated number of points in range

4 Benchmark

TODO describe benchmark methodology tests results

References

[1] Y. Ben-Haim and E. Tom-Tov, "A streaming parallel decision tree algorithm," *J. Mach. Learn. Res.*, vol. 11, pp. 849–872, Mar. 2010.

[2] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: A warehousing solution over a map-reduce framework," *Proc. VLDB Endow.*, vol. 2, pp. 1626–1629, Aug. 2009.

[3] "Poweredby." `https://cwiki.apache.org/confluence/display/Hive/PoweredBy`. Accessed: 2019-05-01.

[4] "Numerichistogram." `https://github.com/apache/hive/blob/f37c5de6c32b9395d1b34fa3c02ed06d1bfbf6eb/ql/src/java/org/apache/hadoop/hive/ql/udf/generic/NumericHistogram.java`. Accessed: 2019-05-01.

[5] "Optimal streaming histograms." `https://amplitude.com/blog/2014/08/06/optimal-streaming-histograms`. Accessed: 2019-05-01.