tdunning / t-digest Public

A new data structure for accurate on-line accumulation of rank-based statistics such as quantiles and trimmed means

Apache-2.0 license

1.8k stars 222 forks

Star Notifications

Code Issues 29 Pull requests 7 Actions Projects Wiki Security ...

main Go to file

gerashegalov and tdunning ... on Aug 26, 2022

View code

t-digest · build passing

A new data structure for accurate online accumulation of rank-based statistics such as quantiles and trimmed means. The t-digest algorithm is also very friendly to parallel programs making it useful in map-reduce and parallel streaming applications implemented using, say, Apache Spark.

The t-digest construction algorithm uses a variant of 1-dimensional k-means clustering to produce a very compact data structure that allows accurate estimation of quantiles. This t-digest data structure can be used to estimate quantiles, compute other rank statistics or even to estimate related measures like trimmed means. The advantage of the t-digest over previous digests for this purpose is that the *t*-digest handles data with full floating point resolution. With small changes, the *t*-digest can handle values from any ordered set for which we can compute something akin to a mean. The accuracy of quantile estimates produced by t-digests can be orders of magnitude more accurate than those produced by alternative digest algorithms in spite of the fact that t-digests are much more compact, particularly when serialized.

In summary, the particularly interesting characteristics of the t-digest are that it

- has smaller summaries when serialized
- works on double precision floating point as well as integers.
- provides part per million accuracy for extreme quantiles and typically <1000 ppm accuracy for middle quantiles
- is very fast (~ 140 ns per add)
- is very simple (~ 5000 lines of code total, <1000 for the most advanced implementation alone)
- has a reference implementation that has > 90% test coverage
- can be used with map-reduce very easily because digests can be merged
- requires no dynamic allocation after initial creation (`MergingDigest` only)

- has no runtime dependencies

Recent News

There is a new article (open access!) in Software Impacts on the t-digest, focussed particularly on this reference implementation.

Lots has happened in t-digest lately. Most recently, with the help of people posting their observations of subtle misbehavior over the last 2 years, I figured out that the sort in the `MergingDigest` really needed to be stable. This helps particularly with repeated values. Stabilizing the sort appears to have no negative impact on accuracy nor significant change in speed, but testing is continuing. As part of introducing this change to the sort, I made the core implementation pickier about enforcing the size invariants which forced updates to a number of tests.

The basic gist of other recent changes is that the core algorithms have been made much more rigorous and the associated papers in the docs directory have been updated to match the reality of the most advanced implementations. The general areas of improvement include substantial speedups, a new framework for dealing with scale functions, real proofs of size bounds and invariants for all current scale functions, much improved interpolation algorithms, better accuracy testing and splitting the entire distribution into parts for the core algorithms, quality testing, benchmarking and documentation.

I am working on a 4.0 release that incorporates all of these improvements. The remaining punch list for the release is roughly:

- ~~verify all tests are clean and not disabled~~ (done!)
- ~~integrate all scale functions into AVLTreeDigest~~ (done!)
- describe accuracy using the quality suite
- extend benchmarks to include `AVLTreeDigest` as first-class alternative
- measure merging performance
- consider issue #87
- review all outstanding issues (add unit tests if necessary or close if not)

Publication work is now centered around comparisons with the KLL digest (spoiler, the t-digest is much smaller and possibly 2 orders of magnitude more accurate than KLL). I would still like to see potential co-authors who could accelerate these submissions are encouraged to speak up! In the meantime, an

README.md

In research areas, there are some ideas being thrown around about how to bring strict guarantees similar to the GK or KLL algorithms to the t-digest. There is some promise here, but nothing real yet. If you are interested in a research project, this could be an interesting one.

Scale Functions

The idea of scale functions is the heart of the t-digest. But things don't quite work the way that we originally thought. Originally, it was presumed that accuracy should be proportional to the square of the size of a cluster. That isn't true in practice. That means that scale functions need to be much more aggressive about controlling cluster sizes near the tails. We now have 4 scale functions supported for both major digest forms (`MergingDigest` and `AVLTreeDigest`) to allow different trade-offs in terms of accuracy.

These scale functions now have associated proofs that they all preserve the key invariants necessary to build an accurate digest and that they all give tight bounds on the size of a digest. Having new scale functions means that we can get much better tail accuracy than before without losing much in terms of median accuracy. It also means that insertion into a `MergingDigest` is faster than before since we have been able to eliminate all fancy functions like sqrt, log or sin from the critical path (although sqrt *is* faster than you might think).

There are also suggestions that asymmetric scale functions would be useful. These would allow good single-tailed accuracy with (slightly) smaller digests. A paper has been submitted on this by the developer who came up with the idea and feedback from users about the utility of such scale functions would be welcome.

Better Interpolation

The better accuracy achieved by the new scale functions partly comes from the fact that the most extreme clusters near q=0 or q=1 are limited to only a single sample. Handling these singletons well makes a huge difference in the accuracy of tail estimates. Handling the transition to non-singletons is also very important.

Both cases are handled much better than before.

The better interpolation has been fully integrated and tested in both the `MergingDigest` and `AVLTreeDigest` with very good improvements in accuracy. The bug detected in the `AVLTreeDigest` that affected data with many repeated values has also been fixed.

Two-level Merging

We now have a trick for the `MergingDigest` that uses a higher value of the compression parameter (delta) while we are accumulating a t-digest and a lower value when we are about to store or display a t-digest. This two-level merging has a small (negative) effect on speed, but a substantial (positive) effect on accuracy because clusters are ordered more strictly. This better ordering of clusters means that the effects of the improved interpolation are much easier to observe.

Extending this to `AVLTreeDigest` is theoretically possible, but it isn't clear the effect it will have.

Repo Reorg

The t-digest repository is now split into different functional areas. This is important because it simplifies the code used in production by extracting the (slow) code that generates data for accuracy testing, but also because it lets us avoid any dependencies on GPL code (notably the jmh benchmarking tools) in the released artifacts.

The major areas are

- core - this is where the t-digest and unit tests live
- docs - the main paper and auxiliary proofs live here
- benchmarks - this is the code that tests the speed of the digest algos
- quality - this is the code that generates and analyzes accuracy information

Within the docs sub-directory, proofs of invariant preservation and size bounds are moved to `docs/proofs` and all figures in `docs/t-digest-paper` are collected into a single directory to avoid cluster.

LogHistogram and FloatHistogram

This package also has an implementation of `FloatHistogram` which is another way to look at distributions where all measurements are positive and where you want relative accuracy in the measurement space instead of accuracy defined in quantiles. This `FloatHistogram` makes use of the floating point hardware to implement variable width bins so that adding data is very fast (5ns/data point in benchmarks) and the resulting sketch is small for reasonable accuracy levels. For instance, if you require dynamic range of a million and are OK with about bins being about ±10%, then you only need 80 counters.

Since the bins for `FloatHistogram` 's are static rather than adaptive, they can be combined very easily. Thus you can store a histogram for short periods of time and combined them at query time if you are looking at metrics for your system. You can also reweight histograms to avoid errors due to structured omission.

Another class called `LogHistogram` is also available in `t-digest` . The `LogHistogram` is very much like the `FloatHistogram` , but it incorporates a clever quadratic update step (thanks to Otmar Ertl) so that the bucket widths vary more precisely and thus the number of buckets can be decreased by about 40% while getting the same accuracy. This is particularly important when you are maintaining only modest accuracy and want small histograms.

In the future, I will incorporate some of the interpolation tricks from the main *t*-digest into the `LogHistogram` implementation.

Compile and Test

You have to have Java 1.8 to compile and run this code. You will also need maven (3+ preferred) to compile and test this software. In order to build the figures that go into the theory paper, you will need R. In order to format the paper, you will need latex. Pre-built pdf versions of all figures and papers are provided so you won't need latex if you don't need to make changes to these documents.

On Ubuntu, you can get the necessary pre-requisites for compiling the code with the following:

```
sudo apt-get install  openjdk-8-jdk git maven
```

Once you have these installed, use this to build and test the software:

```
cd t-digest; mvn test
```

Most of the very slow tests are in the `quality` module so if you just run the tests in `core` module, you can save considerable time.

Testing Accuracy and Comparing to Q-digest

The normal test suite produces a number of diagnostics that describe the scaling and accuracy characteristics of t-digests. In order to produce nice visualizations of these properties, you need to have many more samples. To get this enhanced view, run the tests in the `quality` module by running the full test suite once or, subsequently, by running just the tests in the quality sub-directory.

```
cd quality; mvn test
```

The data from these tests are stored in a variety of data files in the `quality` directory. Some of these files are quite large.

I have prepared detailed instructions on producing all of the figures used in the main paper.

Most of these scripts will complete almost instantaneously; one or two will take a few tens of seconds.

The output of these scripts are a collection of PDF files that can be viewed with any suitable viewer such as Preview on a Mac. Many of these images are used as figures in the main t-digest paper.

Implementations in Other Languages

The t-digest algorithm has been ported to other languages:

- Python: tdigest
- Go: github.com/spenczar/tdigest github.com/influxdata/tdigest
- JavaScript: tdigest
- C++: CPP TDigest, FB's Folly Implementation (high performance)
- C++: TDigest as part of Apache Arrow
- CUDA C++: tdigest.cu as part of `libcudf` in RAPIDS powering the `approx_percentile` and `percentile_approx` expressions in Spark SQL with RAPIDS Accelerator for Apache Spark
- Rust: t-digest and its modified version in Apache Arrow Datafusion
- Scala: TDigest.scala
- C: tdigestc (w/ bindings to Go, Java, Python, JS via wasm)
- C: t-digest-c as part of RedisBloom
- Clojure: t-digest for Clojure
- C#: t-digest-csharp (.NET Core)
- Kotlin multiplatform: tdigest_kotlin_multiplatform
- OCaml: tdigest. Purely functional, can also compile to JS via js_of_ocaml.

Continuous Integration

The t-digest project makes use of Travis integration with Github for testing whenever a change is made.

You can see the reports at:

```
https://travis-ci.org/tdunning/t-digest
```

travis update

Installation

The t-Digest library Jars are released via Maven Central Repository. The current version is 3.3.

```
<dependency>
    <groupId>com.tdunning</groupId>
    <artifactId>t-digest</artifactId>
    <version>3.3</version>
</dependency>
```

Releases

8 tags

Packages

No packages published

Used by 582

+ 574

Contributors 38

+ 27 contributors

Languages

- Java 92.3%
- R 7.7%