COMPUTING ACCURATE QUANTILES USING t-DIGESTS

TED DUNNING

Abstract. An on-line algorithm for computing approximations of rank-based statistics is presented that allows controllable accuracy. Moreover, this new algorithm can be used to compute hybrid statistics such as trimmed means in additional to computing arbitrary quantiles. An unusual property of the method is that it allows a quantile q to be computed with an accuracy relative to $q(1-q)$ rather than with an absolute accuracy as with most methods. This new algorithm is robust with respect to highly skewed distributions or highly ordered datasets and allows separately computed summaries to be combined with no loss in accuracy.

An open-source implementation of this algorithm from the author.

1. Introduction

Given a sequence of numbers, it is often desirable to compute rank-based statistics such as the median, 95-th percentile or trimmed means in an on-line fashion, keeping only a small data structure in memory. Traditionally, such statistics were often computed by sorting all of the data and then either finding the quantile of interest by interpolation or by processing all samples within particular quantiles. This sorting approach is infeasible for very large datasets which has led to interest in on-line approximate algorithms.

One early algorithm for computing on-line quantiles is described in [CLP00]. In that work specific quantiles were computed by incrementing or decrementing an estimate by a value proportional to the simultaneously estimated probability density at the desired quantile. This method is plagued by a circularity in that estimating density is only possible by estimating yet more quantiles. Moreover, this work did not allow the computation of hybrid quantities such as trimmed means.

An alternative approach is described in [SBAS04]. In this work, incoming values are assumed to be integers of fixed size. Such integers can trivially be arranged in a perfectly balanced binary tree where the leaves correspond to the integers and the interior nodes correspond to bit-wise prefixes. This tree forms the basis of the data structure known as a Q-digest. The idea behind a Q-digest is that in the uncompressed case, counts for various values are assigned to leaves of the tree. To compress this tree, sub-trees are collapsed and counts from the leaves are aggregated into a single node representing the sub-tree such that the maximum count for any collapsed sub-tree is less than a threshold that is a small fraction of the total number of integers seen so far. Any quantile can be computed by traversing the tree in left prefix order, adding up counts until the desired fraction of the total is reached. At that point, count for the last sub-tree traversed can be used to

interpolate to the desired quantile within a small and controllable error. The error is bounded because the count for each collapsed sub-tree is bounded.

The two problems with the Q-digest are that it depends on the tree structure being known ahead of time and that the error bounds do not necessarily apply if the algorithm is used in an on-line fashion. Adapting the Q-digest to use an balanced tree over arbitrary elements is difficult. This difficulty arises because rebalancing the tree involves sub-tree rotations and these rotations may require reapportionment of previously collapsed counts in complex ways. This reapportionment could have substantial effects on the accuracy of the algorithm and in any case make the implementation much more complex because the concerns of counting cannot be separated from the concerns of maintaining a balanced tree. Another problem with Q-digests is that if they are subjected to compression during building, it isn't entirely clear how to handle compressed counts that move high above the leaves, but which eventually have non-trivial counts at a lower level in the tree. The proof of correctness for Q-digests ignores this complication by only considering the case where counts are compressed after being collected on the leaves. It would be desirable to have error bounds that apply to a completely on-line data structure. These limitations do not apply in the original formulation of the Q-digest as a compression algorithm for quantile information, but current trends towards the use of on-line algorithms make these limitations awkward.

The work described here shows how the fundamental idea of a Q-digest can be easily extended to values in $\mathbb{R}$ without the complexity of apportioning counts during tree rebalancing. Indeed, this new data structure eliminates the idea of a tree for storing the original samples, maintaining only the concept of collapsing groups of observations in a way that preserves accuracy. This new algorithm, known as t-digest, has well-defined and easily proven error bounds and allows parallel on-line operation. A particularly novel aspect of the variant of t-digest described here is that accuracy for estimating the q quantile is relative to $q(1-q)$. This is in contrast to earlier algorithms which had errors independent of q. The relative error bound of the t-digest is convenient when computing quantiles for very small q or for q near 1. As with the Q-digest algorithm, the accuracy/size trade-off can be controlled by setting a single compression parameter.

The accuracy bounds are tight regardless of data ordering for the non-parallel on-line case. In the case of parallel execution, the error bound is somewhere between the constant error bound and the relative error bound except in the case of highly ordered input data. For randomly ordered data, parallel execution has expected error bounded by the same bound as for sequential execution.

2. The Algorithm

The basic outline of the algorithm for constructing a t-digest is quite simple. An initially empty ordered list of centroids, $C = [c_1 \dots c_m]$ is kept. Each centroid consists of a mean and a count. To add a new value x_n with a weight w_n, the set of centroids is found that have minimum distance to x_n. This set is reduced by retaining only centroids with a count less than $4\delta q(1-q)n$ where δ controls the accuracy of quantile estimates and q is the

Algorithm 1: Construction of a t-Digest

Input: Ordered set of weighted centroids $C = \{\}$, sequence of real-valued, weighted points $X = \{(x_1, w_1), \ldots (x_N, w_N)\}$, and accuracy tolerance δ

Output: final set $C = [c_1 \ldots c_m]$ of weighted centroids

```
for (x_n, w_n) ∈ X :
    z = min |c_i.mean − x|;
    S = {c_i : |c_i.mean − x| = z ∧ c_i.count + 1 ≤ 4nδq(c_i)(1 − q(c_i))};
    while S ≠ {} ∧ w_n > 0 :
        Sample c_j ~ Uniform(S);
        Δw = min(4nδq(c_i)(1 − q(c_i)) − c_j.count, w_n);
        c_j.count ← c_j.count + Δw;
        c_j.mean ← c_j.mean + Δw(x_n − c_j.mean)/c_j.count;
        w_n ← w_n − Δw;
        S ← S − c_j
    if w_n > 0 :
        C ← C + (x_n, w_n);
    if |C| > K/δ :
        C ← cluster(permute(C));
C ← cluster(permute(C));
return C
```

estimated quantile for the mean of the centroid. If more than one centroid remains, one is selected at random. If a centroid is found, then (x_n, w_n) is added to that centroid. It may happen that the weight w_n is larger than can be added to the selected centroid. If so, as much weight as possible is allocated to the selected centroid and the selection is repeated with the remaining weight. If no satisfactory centroid is found or if there is additional weight to be added after all centroids with minimum distance are considered, then x_n is used to form a new centroid and the next point is considered. This procedure is shown more formally as Algorithm 1.

In this algorithm, a centroid object contains a mean and and a count. Updating such an object with a new data-point (x, w) is done using Welford's method [Wik, Knu98, Wel62].

The number of points assigned to each centroid is limited to $\max(1, \lfloor 4N\delta q(1-q) \rfloor)$ where q is the quantile for the approximate mean of the centroid and δ is a parameter chosen to limit the number of points that can be assigned to a centroid. Typically, this compression factor is described in terms of its inverse, $1/\delta$ in order to stay compatible with the conventions used in the Q-Digest. The algorithm approximates q for centroid c_i by summing the weights for all of the centroids ordered before c_i:

$$q(c_i) = \frac{c_i.\texttt{count}/2 + \sum_{j<i} c_j.\texttt{count}}{\sum_j c_j.\texttt{count}}$$

In order to compute this sum quickly, the centroids can be stored in a data structure such as a balanced binary tree that keeps sums of each sub-tree. For centroids with identical means, order of creation is used as a tie-breaker to allow an unambiguous ordering. Figure 1 shows actual centroid weights from multiple runs of this algorithm and for multiple distributions plotted against the ideal bound.

2.1. **Ordered Inputs.** The use of a bound on centroid size that becomes small for extreme values of q is useful because it allows relative error to be bounded very tightly, but this bound may be problematic for some inputs. If the values of X are in ascending or descending order, then C will contain as many centroids as values that have been observed. This will happen because each new value of X will always form a new centroid because $q(1-q) \approx 0$. To avoid this pathology, if the number of centroids becomes excessive, the set of centroids is collapsed by recursively applying the t-digest algorithm to the centroids themselves after randomizing the order of the centroids.

In all cases, after passing through all of the data points, the centroids are recursively clustered one additional time. This allows adjacent centroids to be merged if they do not violate the size bound. This final pass typically reduces the number of centroids by 20-40% with no apparent change in accuracy.

2.2. **Accuracy Considerations.** Initial versions of this algorithm tried to use the centroid index i as a surrogate for q, applying a correction to account for the fact that extreme centroids have less weight. Unfortunately, it was difficult to account for the fact that the distribution of centroid weights changes during the course of running the algorithm. Initially all weights are 1. Later, some weights become substantially larger. This means that the relationship between i and q changes from completely linear to highly non-linear in a stochastic way. This made it difficult to avoid too large or too small cutoff for centroids resulting in either too many centroids or poor accuracy.

The key property of this algorithm is that the final list of centroids C is very close to what would have been obtained by simply sorting and then grouping adjacent values in X into groups with the desired size bounds. Clearly, such groups could be used to compute all of the rank statistics of interest here and if there are bounds on the sizes of the groups, then we have comparable bounds on the accuracy of the rank statistics in question.

That this algorithm does produce such an approximation is more difficult to prove rigorously, but an empirical examination is enlightening. Figure 2 shows the deviation of samples assigned to centroids for uniform and highly skewed distributions. These deviations are normalized by the half the distance between the adjacent two centroids. This relatively uniform distribution for deviations among the samples assigned to a centroid is found for uniformly distributed samples as well as for extremely skewed data. For instance, the $\Gamma(0.1, 0.1)$ distribution has a 0.01 quantile of 6.07×10^{-20}, a median of 0.006 and a mean of 1. This means that the distribution is very skewed. In spite of this, samples assigned to centroids near the first percentile are not noticeably skewed within the centroid. The impact of this uniform distribution is that linear interpolation allows accuracy considerably better than $q(1-q)/\delta$.

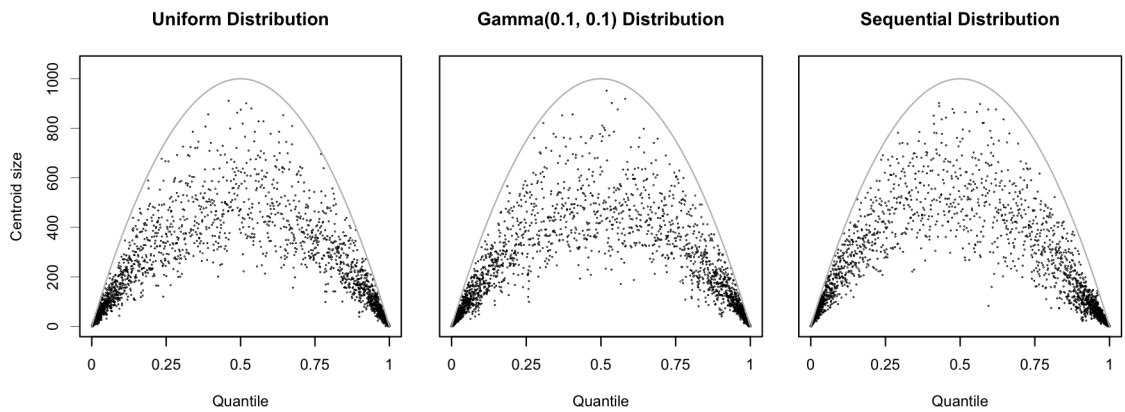

FIGURE 1. The accuracy of the size limit approximation used in the t-digest algorithm. The solid grey line indicates the size limit. These diagrams also shows actual centroid weights for 5 test runs on $100,000$ samples from a uniform, $\Gamma(0.1, 0.1)$ and sequential uniform distribution. In spite of the underlying distribution being skewed by roughly 30 orders of magnitude of difference in probability density for the Γ distribution, the centroid weight distribution is bounded and symmetric as intended. For the sequential uniform case, values are produced in sequential order with three passes through the $[0, 1]$ interval with no repeated values. As can be seen, this sequential presentation results in only a small asymmetry in the resulting size distribution.

2.3. **Finding the cumulative distribution at a point.** Algorithm 2 shows how to compute the cumulative distribution $P(x) = \int_{-\infty}^{x} p(\alpha)\, d\alpha$ for a given value of x by summing the contribution of uniform distributions centered at each the centroids. Each of the centroids is assumed to extend symmetrically around the mean for half the distance to the adjacent centroid.

For all centroids except one, this contribution will be either 0 or $c_i.\texttt{count}/N$ and the one centroid which straddles the desired value of x will have a *pro rata* contribution somewhere between 0 and $c_i.\texttt{count}/N$. Moreover, since each centroid has count at most δN the accuracy of q should be accurate on a scale of δ. Typically, the accuracy will be even better due to the interpolation scheme used in the algorithm and because the largest centroids are only for values of q near 0.5.

The empirical justification for using a uniform distribution for each centroid can be seen by referring to again to Figure 2.

2.4. **Inverting the cumulative distribution.** Computing an approximation of the q quantile of the data points seen so far can be done by ordering the centroids by ascending mean. The running sum of the centroid counts will range from 0 to $N = \sum c_i.\texttt{count}$. One

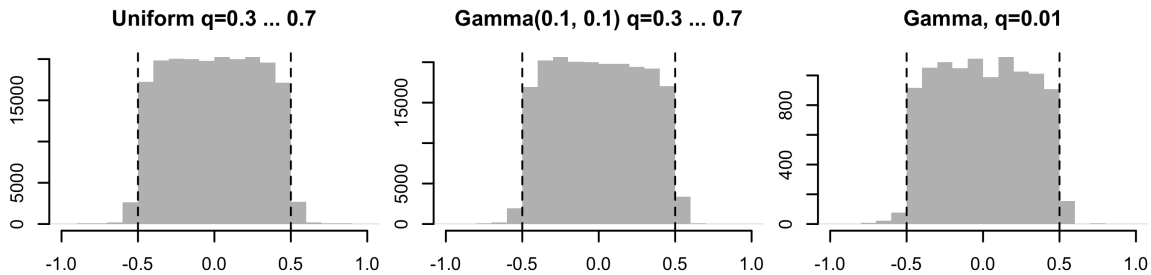

FIGURE 2. The deviation of samples assigned to a single centroid. The horizontal axis is scaled to the distance to the adjacent centroid so a value of 0.5 is half-way between the two centroids. There are two significant observations to be had here. The first is that relatively few points are assigned to a centroid that are beyond the midpoint to the next cluster. This bears on the accuracy of this algorithm. The second observation is that samples are distributed relatively uniformly between the boundaries of the cell. This affects the interpolation method to be chosen when working with quantiles. The three graphs show, respectively, centroids from $q \in [0.3, 0.7]$ from a uniform distribution, centroids from the same range of a highly skewed $\Gamma(0.1, 0.1)$ and centroids from $q \in [0.01, 0.015]$ in a $\Gamma(0.1, 0.1)$ distribution. This last range is in a region where skew on the order of 10^{22} is found.

particular centroid will have a count that straddles the desired quantile q and interpolation can be used to estimate a value of x. This is shown in Algorithm 3. Note that at the

Algorithm 2: Estimate quantile $C.\texttt{quantile}(x)$

Input: Centroids derived from distribution $p(x)$, $C = [\ldots[m_i, s_i, k_i]\ldots]$, value x
Output: Estimated value of $q = \int_{-\infty}^{x} p(\alpha)d\alpha$

```
t = 0, N = Σ_i k_i;
for i ∈ 1...m :
    if i < m :
        Δ ← (c_{i+1}.mean − c_i.mean)/2;
    else:
        Δ ← (c_i.mean − c_{i−1}.mean)/2;
    z = max(−1, (x − m_i)/Δ);
    if z < 1 :
        return (t + (k_i/N)(z+1)/2)
    t ← t + k_i;
return 1
```

extreme ends of the distribution as observed, each centroid will represent a single sample so maximum resolution in q will be retained.

2.5. **Computing the trimmed mean.** The trimmed mean of X for the quantile range $Q = [q_0, q_1]$ can be computed by computing a weighted average of the means of centroids that have quantiles in Q. For centroids at the edge of Q, a *pro rata* weight is used that is based on an interpolated estimate of the fraction of the centroid's samples that are in Q. This method is shown as Algorithm 4.

3. Empirical Assessment

3.1. **Accuracy of estimation for uniform and skewed distributions.** Figure 3 shows the error levels achieved with t-digest in estimating quantiles of 100,000 samples from a uniform and from a skewed distribution. In these experiments $\delta = 0.01$ was used since it provides a good compromise between accuracy and space. There is no visible difference in accuracy between the two underlying distributions in spite of the fact that the underlying densities differ by more roughly 30 orders of magnitude. The accuracy shown here is computed by comparing the computed quantiles to the actual empirical quantiles for the sample used for testing and is shown in terms of q rather than the underlying sample value. At extreme values of q, the actual samples are preserved as centroids with weight 1 so the observed for these extreme values is zero relative to the original data. For the data shown here, at $q = 0.001$, the maximum weight on a centroid is just above 4 and centroids in this range have all possible weights from 1 to 4. Errors are limited to, not surprisingly, just

Algorithm 3: Estimate value at given quantile $C.\texttt{icdf}(q)$

Input: Centroids derived from distribution $p(x)$, $C = [c_1 \dots c_m]$, value q
Output: Estimated value x such that $q = \int_{-\infty}^{x} p(\alpha) d\alpha$

```
t = 0, q ← q Σ c_i.count;
for i ∈ 1 ... m :
    k_i = c_i.count;
    if q < k_i :
        if i = 1 :
            Δ ← (c_{i+1}.mean − c_i.mean);
        elif i = m :
            Δ ← (c_i.mean − c_{i−1}.mean);
        else:
            Δ ← (c_{i+1}.mean − c_{i−1}.mean)/2;
        return m_i + ((q−t)/k_i − 1/2) Δ
    t ← t + k_i
return c_m.mean
```

a few parts per million or less. For more extreme quantiles, the centroids will have fewer samples and the results will typically be exact.

Obviously, with the relatively small numbers of samples such as are used in these experiments, the accuracy of t-digests for estimating quantiles of the underlying distribution cannot be better than the accuracy of these estimates computed using the sample data

Algorithm 4: Estimate trimmed mean. Note how centroids at the boundary are included on a *pro rata* basis.

Input: Centroids derived from distribution $p(x)$, $C = [\ldots[m_i, s_i, k_i]\ldots]$, limit values q_0, q_2

Output: Estimate of mean of values $x \in [q_0, q_1]$

```
s = 0, k = 0;
t = 0, q_1 ← q_1 ∑ k_i, q_1 ← q_1 ∑ k_i;
for i ∈ 1 ... m :
    k_i = c_i.count;
    if q_1 < t + k_i :
        if i > 1 :
            Δ ← (c_{i+1}.mean − c_{i−1}.mean)/2;
        elif i < m :
            Δ ← (c_{i+1}.mean − c_i.mean);
        else:
            Δ ← (c_i.mean − c_{i−1}.mean);
        η = ((q−t)/k_i − 1/2) Δ;
        s ← s + η k_i c_i.mean;
        k ← k + η k_i;
    if q_2 < t + k_i :
        if i > 1 :
            Δ ← (c_{i+1}.mean − c_{i−1}.mean)/2;
        elif i < m :
            Δ ← (c_{i+1}.mean − c_i.mean);
        else:
            Δ ← (c_i.mean − c_{i−1}.mean);
        η = (1/2 − (q−t)/k_i) Δ;
        s ← s − η k_i c_i.mean;
        k ← k − η k_i;
    t ← t + k_i
return s/k
```

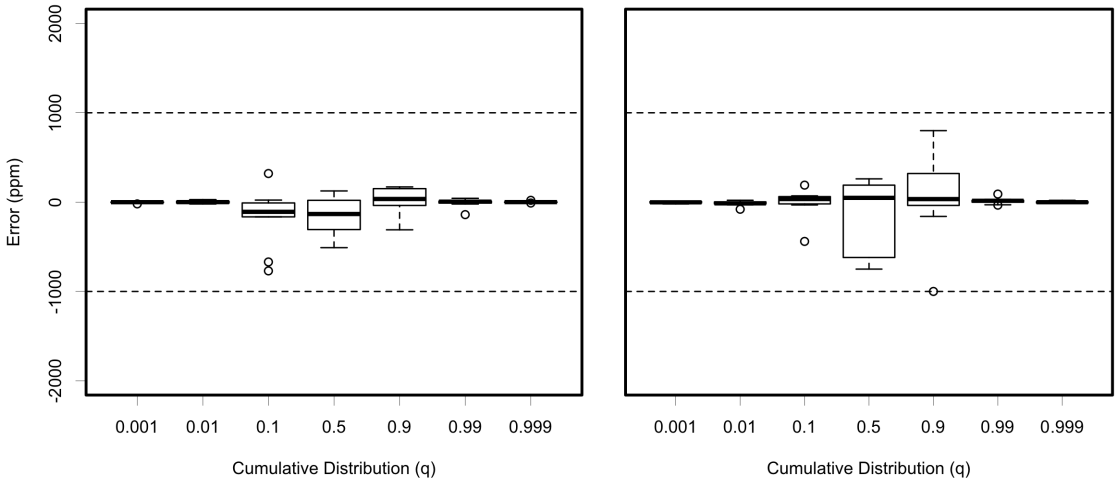

FIGURE 3. The absolute error of the estimate of the cumulative distribution function $q = \int_{-\infty}^{x} p(\alpha)\, d\alpha$ for the uniform distribution for 5 runs, each with 100,000 data points. As can be seen, the error is dramatically decreased for very high or very low quantiles (to a few parts per million). The precision setting used here, $1/\delta = 100$, would result in uniform error of 10,000 ppm without adaptive bin sizing and interpolation.

points themselves. For the experiments here, the errors due to sampling completely dominate the errors introduced by t-digests, especially at extreme values of q. For much larger sample sets of billions of samples or more, this would be less true and the errors shown here would represent the accuracy of approximating the underlying distribution.

It should be noted that using a Q-Digest implemented with long integers is only able to store data with no more than 20 significant decimal figures. The implementation in stream-lib only retains 48 bits of significants, allowing only about 16 significant figures. This means that such a Q-digest would be inherently unable to even estimate the quantiles of the Γ distribution tested here.

3.2. **Persisting t-digests.** For the accuracy setting and test data used in these experiments, the t-digest contained $820 - 860$ centroids. The results of t-digest can thus be stored by storing this many centroid means and weights. If centroids are kept as double precision floating point numbers and counts kept as 4-byte integers, the t-digest resulting from from the accuracy tests described here would require about 10 kilobytes of storage for any of the distributions tested.

This size can be substantially decreased, however. One simple option is to store differences between centroid means and to use a variable byte encoding such as zig-zag encoding

to store the cluster size. The differences between successive means are at least three orders of magnitude smaller than the means themselves so using single precision floating point to store these differences can allow the t-digest from the tests described here to be stored in about 4.6 kilobytes while still regaining nearly 10 significant figures of accuracy in the means. This is roughly equivalent to the precision possible with a Q-digest operating on 32 bit integers, but the dynamic range of t-digests will be considerably higher and the accuracy is considerably better.

3.3. **Space/Accuracy Trade-off.** Not surprisingly, there is a strong trade-off between the size of the t-digest as controlled by the compression parameter $1/\delta$ and the accuracy which which quantiles are estimated. Quantiles at 0.999 and above or 0.001 or below were estimated to within a small fraction of 1% regardless of digest size. Accurate estimates of the median require substantially larger digests. Figure 4 shows this basic trade-off.

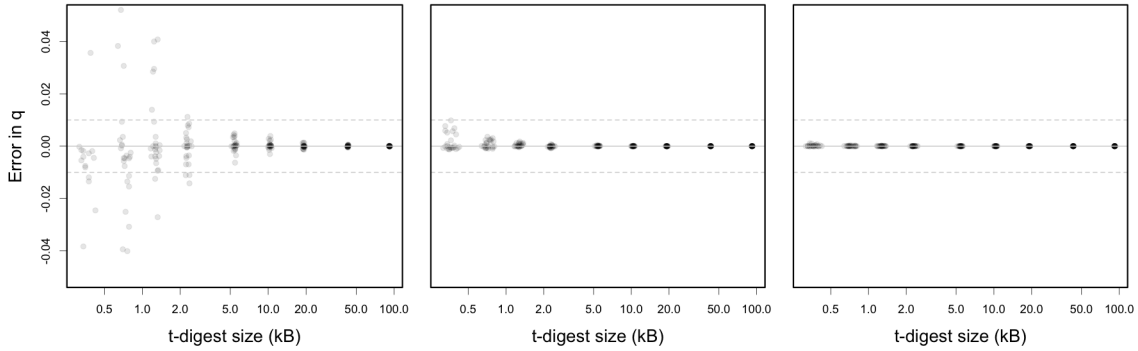

FIGURE 4. Accuracy is good for extreme quantiles regardless of digest size. From left to right, these panels show accuracy of estimates for $q = 0.5, 0.01$ and 0.001 as a function the serialized size of the t-digest. Due to symmetry, these are equivalent to accuracy for $q = 0.5, 0.99$, and 0.999 as well. For mid quantiles such as the median ($q = 0.5$), moderate digest sizes of a few kilobytes suffice to get better than 1% accuracy, but a digest must be 20kB or more to reliably achieve 0.1% accuracy. In contrast, accuracy for the 0.1%-ile (or 99.9%-ile) reaches a few parts per million for digests larger than about 5 kB. Note that errors for $q = 0.5$ and digests smaller than 1 kB are off the scale shown here at nearly 10%. All panels were computed using 100 runs with 100,000 samples. Compression parameter ($1/\delta$) was varied from 2 to 1000 in order to vary the size of the resulting digest. Sizes shown were encoded using 4 byte floating point delta encoding for the centroid means and variable byte length integer encoding.

The size of the resulting digest depends strongly on the compression parameter $1/\delta$ as shown in the left panel of Figure 5. Size of the digest also grows roughly with the log of the

number of samples observed, at least in the range of 10,000 to 10,000,000 samples shown in the right panel of Figure 5.

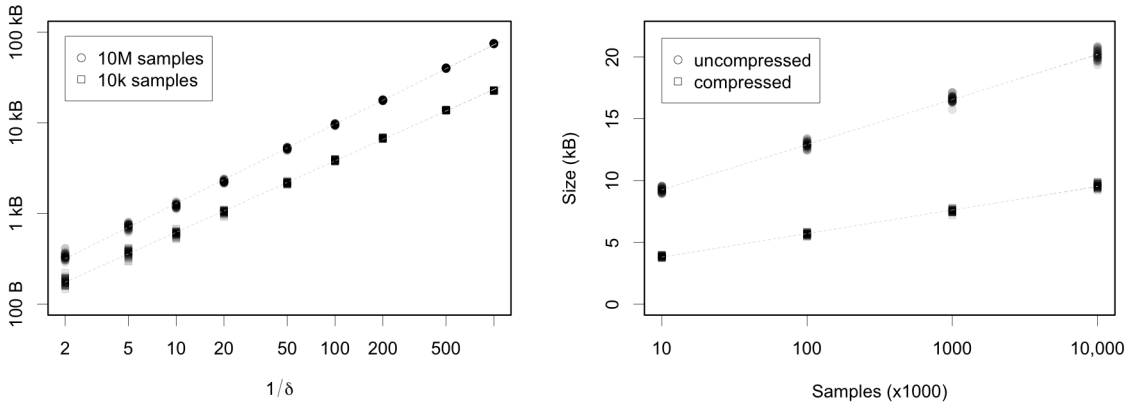

FIGURE 5. Size of the digest scales sub-linearly with compression parameter ($\alpha \approx 0.7 \ldots 0.9$) for fixed number of points. Size scales approximately logarithmically with number of points for fixed compression parameter. The graph shown here is for $1/\delta = 100$. The dashed lines show best-fit log-linear models.

3.4. **Computing t-digests in parallel.** With large scale computations, it is important to be able to compute aggregates like the t-digest on portions of the input and then combine those aggregates.

For example, in a map-reduce framework such as Hadoop, a combiner function can compute the t-digest for the output of each mapper and a single reducer can be used to compute the t-digest for the entire data set.

Another example can be found in certain databases such as Couchbase or Druid which maintain tree structured indices and allow the programmer to specify that particular aggregates of the data being stored can be kept at interior nodes of the index. The benefit of this is that aggregation can be done almost instantaneously over any contiguous sub-range of the index. The cost is quite modest with only a $O(\log(N))$ total increase in effort over keeping a running aggregate of all data. In many practical cases, the tree can contain only two or three levels and still provide fast enough response. For instance, if it is desired to be able to compute quantiles for any period up to years in 30 second increments, simply keeping higher level t-digests at the level of 30 seconds and days is likely to be satisfactory because at most about 10,000 digests are likely to need merging even for particularly odd intervals. If almost all queries over intervals longer than a few weeks are day aligned, the number of digests that must be merged drops to a few thousand.

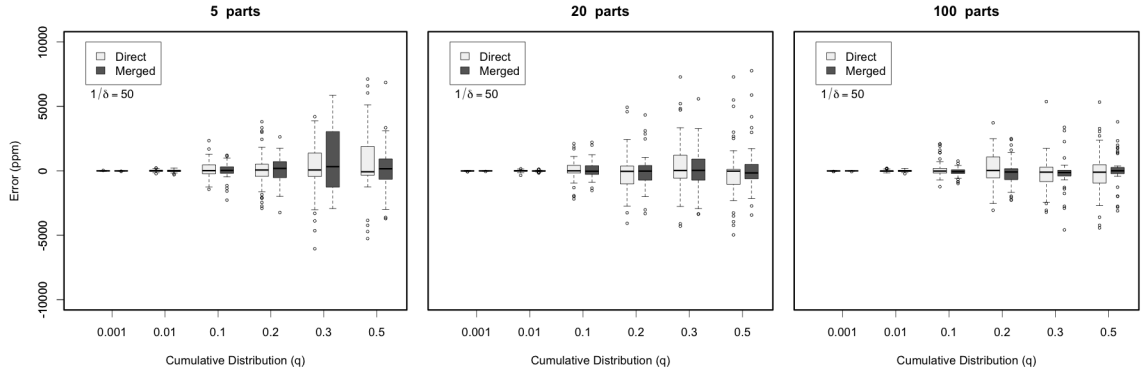

FIGURE 6. Accuracy of a t-digest accumulated directly is essentially the same as when the digest is computed by combining digests from 5, 20 or 100 equal sized portions of the data. This allows efficient use of the $t$digest in map-reduce and database applications. Of particular interest is the fact that accuracy actually improves when the input data is broken in to many parts as is shown in the right hand panel. All panels were computed by 40 repetitions of aggregating 100,000 values. Accuracy for directly accumulated digests is shown on the left of each pair with the white bar and the digest of digest accuracy is shown on the right of each pair by the dark gray bar.

Merging t-digests can be done many ways. The algorithm whose results are shown here consisted of simply making a list of all centroids from the t-digests being merged, shuffling that list, and then adding these centroids, preserving their weights, to a new t-digest.

3.5. **Comparison with Q-digest.** The prototype implementation of the t-digest completely dominates the implementation of the Q-digest from the popular stream-lib package [Thi] when size of the resulting digest is controlled. This is shown in Figure 7. In the left panel, the relationship between the effect of the compression parameter for Q-digest is compared to the similar parameter $1/\delta$ for the t-digest. For the same value of compression parameter, the sizes of the two digests is always within a factor of 2 for practical uses. The middle and right panel show accuracies for uniform and Γ distributions.

As expected, the t-digest has very good accuracy for extreme quantiles while the Q-digest has constant error across the range. Interestingly, the accuracy of the Q-digest is at best roughly an order of magnitude worse than the accuracy of the t-digest even. At worse, with extreme values of q, accuracy is several orders of magnitude worse. This situation is even worse with a highly skewed distribution such as with the $\Gamma(0.1, 0.1)$ shown in the right panel. Here, the very high dynamic range introduces severe quantization errors into the results. This quantization is inherent in the use of integers in the Q-digest.

For higher compression parameter values, the size of the Q-digest becomes up to two times smaller than the t-digest, but no improvement in the error rates is observed.

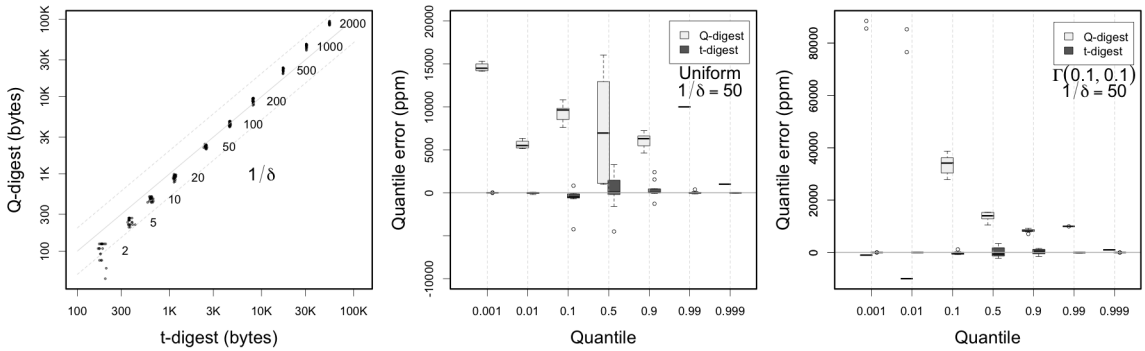

FIGURE 7. The left panel shows the size of a serialized Q-digest versus the size of a serialized t-digest for various values of $1/\delta$ from 2 to 100,000. The sizes for the two kinds of digest are within a factor of 2 for all compression levels. The middle and right panels show the accuracy for a particular setting of $1/\delta$ for Q-digest and t-digest. For each quantile, the Q-digest accuracy is shown as the left bar in a pair and the t-digest accuracy is shown as the right bar in a pair. Note that the vertical scale in these diagrams are one or two orders of magnitude larger than in the previous accuracy graphs and that in all cases, the accuracy of the t-digest is dramatically better than that of the Q-digest even though the serialized size of the each is within 10% of the other. Note that the errors in the right hand panel are systematic quantization errors introduced by the use of integers in the Q-digest algorithm. Any distribution with very large dynamic range will show the same problems.

3.6. **Speed.** The current implementation has been primarily optimized for ease of development, not execution speed. As it is, running on a single core of a 2.3 GHz Intel Core i5, it typically takes 2-3 microseconds to process each point after JIT optimization has come into effect. It is to be expected that a substantial improvement in speed could be had by profiling and cleaning up the prototype code.

4. CONCLUSION

The t-digest is a novel algorithm that dominates the previously state-of-the-art Q-digest in terms of accuracy and size. The t-digest provides accurate on-line estimates of a variety of of rank-based statistics including quantiles and trimmed mean. The algorithm is simple and empirically demonstrated to exhibit accuracy as predicted on theoretical grounds. Moreover, the t-digest algorithm is inherently on-line while the Q-digest is an on-line adaptation of an off-line algorithm.

The t-digest algorithm is available in the form of an open source, well-tested implementation from the author. It has already been adopted by the Apache Mahout project and is likely to be adopted by other projects as well.

References

[CLP00] Fei Chen, Diane Lambert, and Jos C. Pinheiro. Incremental quantile estimation for massive tracking. In *In Proceedings of KDD*, pages 516–522, 2000.

[Knu98] Donald E. Knuth. *The Art of Computer Programming, volume 2: Seminumerical Algorithms*, page 232. Addison-Wesley, Boston, 3 edition, 1998.

[SBAS04] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: New aggregation techniques for sensor networks. pages 239–249. ACM Press, 2004.

[Thi] Add This. Algorithms for calculating variance, online algorithm. `https://github.com/addthis/stream-lib`. [Online; accessed 28-November-2013].

[Wel62] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, pages 419–420, 1962.

[Wik] Wikipedia. Algorithms for calculating variance, online algorithm. `http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#Online_algorithm`. [Online; accessed 19-October-2013].

E-mail address: `ted.dunning@gmail.com`