# Summary Data Structures for Massive Data

Graham Cormode

Department of Computer Science, University of Warwick
graham@cormode.org

**Abstract.** Prompted by the need to compute holistic properties of increasingly large data sets, the notion of the "summary" data structure has emerged in recent years as an important concept. Summary structures can be built over large, distributed data, and provide guaranteed performance for a variety of data summarization tasks. Various types of summaries are known: summaries based on random sampling; summaries formed as linear sketches of the input data; and other summaries designed for a specific problem at hand.

## 1 Introduction

It is widely acknowledged in the business and scientific communities that "big data" holds both tremendous promise, and substantial challenges [10]. There is much potential for extracting useful intelligence and actionable information from the large quantities of data generated and captured by modern information processing systems. A chief problem presented by this scenario is the scale in terms of the so-called "three V's": volume, variety, and velocity. That is, big data challenges involve not only the sheer volume of the data, but the fact that it can represent a complex variety of entities and interactions between them, and new observations arrive, often across multiple locations, at high velocity.

Such sources of big data are becoming increasingly common, while the resources to deal with big data (chiefly, processor speed, fast memory and slower disk) are growing at a slower pace. The consequence of this trend is that we need more effort directed towards capturing and processing data in many critical applications. Careful planning and scalable architectures are needed to fulfill the requirements of analysis and information extraction on big data.

Some examples of applications that generate big data include:

*Physical Data.* The growing development of sensors and sensor deployments have led to settings where measurements of the physical world are available at very high dimensionality and at a great rate. Scientific measurements are the cutting edge of this trend. Astronomy data gathered from modern telescopes can easily generate terabytes of data in a single night. Aggregating large quantities of astronomical data provides a substantial big data challenge to support the study and discovery of new phenomena. Big data from particle physics experiments is also enormous: each experiment can generate many terabytes of readings, which can dwarf what is economically feasible to store for later comparison and investigation.

*Medical Data.* It is increasingly feasible to sequence entire genomes. A single human genome is not so large—it can be represented in under a gigabyte—but considering the entire genetic data of a large population represents a big data challenge. This may be accompanied by increasing growth in other forms of medical data, based on monitoring multiple vital signs for many patients at fine granularity. Collectively, this leads to the area of data-driven medicine, seeking better understanding of disease, and leading to new treatments and interventions, personalized for each individual patient.

*Activity Data.* Human activity data is increasingly being captured and stored. Online social networks record not just friendship relations but interactions, messages, photos and interests. Location data is also more available, due to mobile devices which can obtain GPS information. Other electronic activities, such as patterns of website visits, email messages and phone calls can be collected and analyzed. Collectively, this provides ever-larger collections of activity information. Service providers who can collect this data seek to make sense of it in order to identify patterns of behavior or signals of behavioral change, and opportunities for advertising and marketing.

*Business Data.* Businesses are increasingly able to capture more and complex data about their customers. Online stores can track millions of customers as they explore their site, and seek patterns in purchasing and interest, with the aim of providing better service, and anticipating future needs. The detail level of data is getting finer and finer. Previously, data would be limited to just the items purchased, but now extends to more detailed shopping and comparison activity, tracking the whole path to purchase.

Across all of these disparate settings, certain common themes emerge. The data in question is large, and growing. The applications seek to extract patterns, trends or descriptions of the data. Scalability and timeliness of response are vital in many of these applications.

In response to these needs, new computational paradigms are being adopted to deal with the challenge of big data. Large scale distributed computation is a central piece: the scope of the computation can exceed what is feasible on a single machine, and so clusters of machines work together in parallel. On top of these architectures, parallel algorithms are designed that can take the complex task and break it into independent pieces suitable for distribution over multiple machines.

A central challenge within any such system is how to compute and represent complex features of big data in a way that can be processed by many single machines in parallel. One answer is to be able to build and manipulate a compact summary of a large amount of data. This notion of a small summary is the subject of study of this article. The idea of a summary is a natural and familiar one. It should represent something large and complex in a compact fashion. Inevitably, a summary must dispense with some of the detail and nuance of the object which it is summarizing. However, it should also preserve some key features of the object in an accurate fashion.

There is no single summary which accurately captures all properties of a data set, even approximately. Thus, at the heart of the study of small summaries are the questions of *what should be preserved?* and *how accurately can it be preserved?*. The answer to the first question determines which of many different possible summary types may be appropriate, or indeed whether any compact summary even exists. The answer to the second question can determine the size and processing cost of working with the summary in question.

Another important question about summaries for big data is how they can be constructed and maintained as new data arrives. Given that it is typically not feasible to load all the data into memory on one machine, then we need summaries which can be constructed incrementally. That is, we seek summaries that can be built by observing each data item in turn, and updating the partial summary. Or, more strongly, we seek summaries such that summaries of different subsets of data built on different machines can be combined together to obtain a single summary that accurately represents the full data set.

Note that the notion of summarization is distinct from traditional ideas of *compression*. Lossless compression is concerned with identifying regularity and redundancy in datasets to provide a more compact exact representation of the data. This is done for the purpose of archiving, or reducing transmission time. However, in general, there is no guarantee of significant size reduction from compression. The compressed form is also typically difficult to utilize, and decompression is required in order to work with the data. In contrast, summarization is intended to provide a very significant reduction in the size of the data (sometimes several orders of magnitude), but does not promise to reconstruct the original data, only to capture certain key properties. Lossy compression methods fall in between, as they can provide guaranteed size reductions. They also aim to allow an approximate reconstruction of the original data with only small loss of fidelity based on some measure of loss: typically, human perception of multimedia data, such as audio or video. Summarization aims to provide only small loss of fidelity, but on other dimensions; summaries do not necessarily provide a way to make even an approximate reconstruction of the original input.

As a first example of summarization, consider a data set consisting of a large collection of temperature readings over time. A suitable summary might be to keep the sum of all the temperatures seen, and the count. From this summary, we can extract the average temperature. This summary is easy to update incrementally, and can also be combined with a corresponding summary by computing the overall sum and count. A different summary retains only the maximum and minimum temperature observed so far. From this, we can extract the range of temperatures observed. This too is straightforward to maintain under updates, and to merge across multiple subsets. However, neither summary is good at retrieving the median temperature, or some other properties of the statistical distribution of temperatures. Instead, more complex summaries and maintenance procedures are required.

## 2 Operations on Summaries

While different summaries capture different functions and properties of the data, we abstract a set of four basic operations that summaries should support. These basic operations are INITIALIZE, UPDATE, MERGE and QUERY.

INITIALIZE. The INITIALIZE operation for a summary is to initialize a new instance of the summary. Typically, this is quite simple, just creating empty data structures for the summary to use. For summaries that use randomization, this can also involve drawing the random values that will be used throughout the operation of the summary.

UPDATE. The UPDATE operation takes a new data item, and updates the summary to reflect this. The time to do this UPDATE should be quite fast, since we want to process a large input. Ideally, this is faster than reading the whole summary. Since UPDATE takes a single item at a time, the summary can process a stream of items one at a time, and only retain the current state of the summary at each step.

MERGE. When faced with a large amount of data to summarize, we would like to distribute the computation over multiple machines. Performing a sequence of UPDATE operations does not guarantee that we can parallelize the action of the summary, so we also need the ability to MERGE together a pair of summaries to obtain a summary of the union of their inputs. This is possible in the majority of cases, although a few summaries only provide an UPDATE operation and not a MERGE. MERGE is often a generalization of UPDATE: applying MERGE when one of the input summaries consists of just a single item will reduce to the UPDATE operation. In general a MERGE operation is slower than UPDATE, since it requires reading through both summaries in full.

QUERY. At various points we want to use the summary to learn something about the data that is summarized. We abstract this as QUERY, with the understanding that the meaning of QUERY depends on the summary: different summaries capture different properties of the data. In some cases, QUERY takes parameters, while for other summaries, there is a single QUERY operation. Some summaries can be used to answer several different types of query. In this presentation, we pick one primary query to answer with the QUERY operation.

## 3 Three Examples of Summaries

This section describes three examples of different constructions of summaries, and some of their applications and uses. The aim of the presentation is to convey the basic ideas; for further details the interested reader is directed to more detailed surveys [8, 18].

### 3.1 Samples

The notion of random sampling is a familiar and convenient one. Given a large number of data items, we randomly pick a subset, so that the probability of any subset being picked as the sample is uniform. Then, we can estimate the answer to a query by evaluating the query over the sample, and scaling appropriately. For certain queries, strong error guarantees are known for this process.

Working with a sample of size $k$ can fit neatly into the summary framework. To INITIALIZE a sample, we create an empty set. To UPDATE a sample, we determine (randomly) whether to include the new item in the sample, and to MERGE two samples, we will determine which subset of items to maintain. One natural way to accomplish these steps is to maintain the "weight" of each sample, as the number of input items represented by the sample. To MERGE two samples of weight $n_1$ and $n_2$ respectively, we pick items from the first with probability $n_1/(n_1+n_2)$, and from the second with probability $n_2/(n_1+n_2)$. An UPDATE operation is then a MERGE where one of the input samples has size 1. Lastly, to QUERY a sample to estimate the fraction of input items satisfying a property $P$, we can report the fraction of items in the sample satisfying $P$.

Tools such as the Chernoff-Hoeffding inequality [16] can be brought to bear to analyze the accuracy of answering a query from samples. First, we observe that the result of a combination of MERGE and UPDATE operations generates a uniform random sample of size $k$ – the probability of any item being included in the sample is uniform, and independent of the inclusion probabilities of other items. Then we can consider the random variable corresponding to the fraction of items in the sample that satisfy property $P$. Let $X_i$ be the random variable indicating if the $i$th sampled item satisfies $P$: we have $\Pr[X_i = 1] = p$, where $p$ is the global proportion we are trying to estimate. Applying the Chernoff-Hoeffding inequality, we have

$$\Pr[|\frac{1}{k}\sum_{i=1}^{k} X_i - p| > \varepsilon] \leq 2\exp(-2\varepsilon^2 k)$$

Consequently, setting the sample size $k = O(\frac{1}{\varepsilon^2}\ln 1/\delta)$ is sufficient to ensure that the error is at most $\varepsilon$ with probability at least $1-\delta$.

*Samples: pros and cons.* Due to their flexibility and simplicity, samples have been used in a variety of applications. For example, many routers in the internet sample information about flows for network traffic monitoring [17]. It is often convenient to work with samples, since we just have to apply the desired computation on the sample instead of on the full data (in contrast, other summaries require different, complex procedures to be performed on the summary data). However, they have some limitations. Samples do not work well for problems not based on raw counts – for example, samples are not helpful for estimating the number of distinct items [5]. The accuracy bounds in terms of $O(1/\varepsilon^2)$ can be high in some applications where $\varepsilon$ is very small. In some cases, other summary techniques achieve a smaller cost, proportional to $O(1/\varepsilon)$, which can make a big difference in practice.

### 3.2 Sketches

The term 'sketches' refers generally to a class of summary that can be expressed as a linear transformation of the input data. That is, if we can express the input as a vector, then the sketch is the result of multiplying the vector by some matrix $S$. We are most interested in cases where this matrix $S$ has a compact, implicit representation, so that we do not have to explicitly store it in full. A simple example is the Count-Min sketch, which summarizes a large vector so that individual entries of the vector can be estimated [9]. Entries of the vector are mapped down to a smaller domain by a hash function, and the sum of entries mapping to each location are maintained; this is repeated for a small number of different hash functions with the same range.

Using the terminology of summaries, to INITIALIZE a Count-Min sketch, we first determine the parameters: $d$, the number of hash functions to apply, and $w$, the range of the hash functions. We then create an array $C$ of size $d \times w$ to summarize the data, and pick $d$ hash functions $h_j$. To UPDATE the sketch with a positive increment of $u$ to vector location $i$, we map $i$ into $C$ by the hash functions, and compute $C[j, h_j(i)] \leftarrow C[j, h_j(i)] + u$ for all $1 \leq j \leq d$. This is a linear update function. To MERGE two sketches that share the same parameters ($d, w$ and $h_j$), we sum the two arrays. Lastly, to QUERY the estimated value of a location $i$, we find $\min_j C[j, h_j(i)]$, the smallest value in the array of the locations where location $i$ was mapped to: assuming positive weights for all item, this indicates the estimate with the least error from colliding items.

From this description it is reasonably straightforward to see that the sketch is a linear transform of its input vector $x$, so it is the same, irrespective of the ordering of the UPDATE and MERGE operations. It is less immediate to see why the sketch should be accurate, given that $w$ is typically chosen to be much smaller than the size of the vector being summarized. The intuition behind the sketch is that the hash functions work to spread out the different items, so on average the error in the estimates cannot be too high. Then the use of different hash functions ensures that the chance of getting an estimate that is much more inaccurate than average is quite small. In particular, in any given row, the estimate for $i$ is expected to have error proportional to $\|x\|_1/w$, where $\|x\|_1$ denotes the $L_1$ norm of the input vector $x$. Consequently, there is constant probability ($\frac{1}{2}$) that the error is at most $2\|x\|_1/w$, by the Markov inequality. Taking the minimum of $d$ repetitions amplifies this to a probability of $2^{-d}$. Equivalently, to guarantee an error of at most $\varepsilon\|x\|_1$ with probability $1 - \delta$, we set the parameters to $w = 2/\varepsilon$ and $d = \log_2 1/\delta$.

*Sketches: pros and cons.* Sketch techniques have been used widely for log data analysis in large systems, such as Sawzall [19] and CMON [22]. Their linearity offers extra flexibility: it means that we can obtain a sketch of the sums, differences and scaled versions of inputs by applying the corresponding transformation to the sketch data structure. This makes them useful for applying linear forecasting models to large data [7]. They have also been used as the basis for building machine learning models over very large feature domains [23]. Sketch constructions

are known for a variety of input types, including count distinct and set sizes [11], set membership [4], vector operations such as dot-product [3] and matrix computations [6]. However, they have their limitations: some sketch structures can be large and very slow to update, limiting their practicality. Sketch ideas do not naturally apply to settings with unbounded domains, such as sets of strings. Lastly, they do not support arbitrary complex operations: one sketch addresses only one or two defined problems, and its is hard to compose sketches to address complex aggregates.

### 3.3 Special-Purpose Summaries

Other summaries take the form of various special-purpose structures, which capture various aspects of the data. The properties of these vary instance by instance, and require special analysis to accompany their implementation. However, they can offer very good performance in terms of their space and time costs.

A simple example is the MG summary, named for the initials of its inventors. This was originally proposed in the context of streams of data [15], and later generalized to allow merging of summaries [1] for summarizing a collection of weights. The MG summary stores a collection of pairs: items drawn from the input $x$ and associated weights $w_x$. To INITIALIZE an empty MG summary, we create an empty set of tuples, with space for $k$ pairs. The MERGE of two MG summaries takes two summaries constructed using the same parameter $k$. We first merge the component tuples in the natural way. If $x$ is stored in both summaries, its merged weight is the sum of the weights in each input summary. If $x$ is stored in only one of the summaries, it is also placed in the merged summary with the same weight. We then sort the tuples by weight, and find the $k+1$st largest weight, $w_{k+1}$. This weight is subtracted from *all* tuples, and only those with positive weight are retained. At most $k$ tuples can now have weight above zero: the tuple with $k+1$st largest weight, and all tuples with smaller weight, will now have weight 0 or below, and so can be discarded from the summary. The UPDATE procedure is the special case of MERGE where one of the summaries contains just a single item. To QUERY for the estimated weight of an item $x$, we look up whether $x$ is stored in the summary. If so, QUERY reports the associated weight $w_x$ as the estimate, otherwise the weight is assumed to be 0.

Comparing the approximate answer given by QUERY, and the true weight of $x$ (the sum of all weights associated with $x$ in the input), the approximate answer is never more than the true answer. This is because the weight associated with $x$ in the summary is the sum of all weights for $x$ in the input, less the various decreases due to MERGE and UPDATE operations. The MG summary also ensures that this estimated weight is not more than $\varepsilon W$ below the true answer, where $W$ is the sum of all input weights.

*Special-purpose summaries: pros and cons.* Summaries such as the MG summary have been used for quite a wide variety of purposes, such as web clickstream mining [14]. They tend to work very well for their target domain but, as indicated

by their name, do not obviously generalize to other problems. Other special-purpose summaries include summaries for order statistics of distributions, such as the Greenwald-Khanna summary [12], and the q-digest [21]; $\varepsilon$-approximations and $\varepsilon$-nets for geometric data [13]; and summaries for graph distances and connectivity [2].

## 4 Summaries Summary

As noted in the preceding discussion, there are numerous, growing applications for summary data structures, from scaling up machine learning and data mining, to applications in privacy and security (e.g. [20]). For more information, see various tutorials and surveys [8]. Many directions for further research are open.

Natural questions surround improving on the current summaries, either in terms of their parameters (reducing the space and update time), or extending their generality (allowing them to be applied in more general update models). While summaries for computing over data that can be thought of in terms of vectors are by now quite well-understood, the situation for other forms of data – in particular, matrices and graphs – is less complete. New styles of summary are required in the context of verification of computation: these are summaries that allow a "verifier" to check the computation performed by a more powerful "prover" on a stream of data. A list of specific open problems is maintained at `http://sublinear.info/`.

## References


1. P. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei, and K. Yi. Mergeable summaries. In *ACM Principles of Database Systems*, 2012.
2. K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *ACM-SIAM Symposium on Discrete Algorithms*, 2012.
3. N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *ACM Principles of Database Systems*, pages 10–20, 1999.
4. B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.
5. M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya. Towards estimation error guarantees for distinct values. In *ACM Principles of Database Systems*, pages 268–279, 2000.
6. K. L. Clarkson and D. P. Woodruff. Numerical linear algebra in the streaming model. In *ACM Symposium on Theory of Computing*, pages 205–214, 2009.
7. G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *International Conference on Very Large Data Bases*, 2005.
8. G. Cormode, M. Garofalakis, P. Haas, and C. Jermaine. *Synposes for Massive Data: Samples, Histograms, Wavelets and Sketches*. Foundations and Trends in Databases. *NOW publishers*, 2012.
9. G. Cormode and S. Muthukrishnan. An improved data stream summary: The Count-Min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

10. K. Cukier. Data, data everywhere. *The Economist*, Feb. 2010.
11. P. Flajolet and G. N. Martin. Probabilistic counting algorithms for database applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.
12. M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD International Conference on Management of Data*, 2001.
13. S. Har-Peled and S. Mazumdar. Coresets for $k$-means and $k$-median clustering and their applications. In *ACM Symposium on Theory of Computing*, pages 291–300, 2004.
14. A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient computation of frequent and top-$k$ elements in data streams. In *International Conference on Database Theory*, 2005.
15. J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2:143–152, 1982.
16. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
17. Cisco NetFlow. More details at `http://www.cisco.com/warp/public/732/Tech/netflow/`.
18. F. Olken. *Random Sampling from Databases*. PhD thesis, Berkeley, 1997.
19. R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with sawzall. *Dynamic Grids and Worldwide Computing*, 13(4):277–298, 2005.
20. S. Schechter, C. Herley, and M. Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of HotNets*, 2010.
21. N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *ACM SenSys*, 2004.
22. K. To, T. Ye, and S. Bhattacharyya. CMON: A general purpose continuous IP backbone traffic analysis platform. Technical Report RR04-ATL-110309, Sprint ATL, 2004.
23. K. Q. Weinberger, A. Dasgupta, J. Langford, A. J. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *International Conference on Machine Learning (ICML)*, 2009.