1
Introduction

> "Space," it says, "is big. Really big. You just won't believe how vastly, hugely, mindbogglingly big it is. I mean, you may think it's a long way down the road to the chemist's, but that's just peanuts to space, listen…"
>
> Douglas Adams, *The Hitchhiker's Guide to the Galaxy*

1.1 Small Summaries for Big Data

Data, to paraphrase Douglas Adams, is big. Really big. Moreover, it is getting bigger, due to increased abilities to measure and capture more information. Sources of big data are becoming increasingly common, while the resources to deal with big data (chiefly, processor power, fast memory, and slower disk) are growing at a slower pace. The consequence of this trend is that we need more effort in order to capture and process data in applications. Careful planning and scalable architectures are needed to fulfill the requirements of analysis and information extraction on big data. While the "big" in big data can be interpreted more broadly, to refer to the big potential of such data, or the wide variety of data, the focus of this volume is primarily on the scale of data.

Some examples of applications that generate large volumes of data include the following:

Physical Data. The growing development of sensors and sensor deployments has 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. The volume of 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 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. We commonly capture large amounts of human activity data. Online social networks record not just friendship relations but interactions, messages, photos, and interests. Location datasets are also more available, due to mobile devices that 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 such 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 datasets in question are 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, modeled as a mathematical object. This notion of a small summary is the subject of study of this work. 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 that it is summarizing. However, it should also preserve some key features of the object in an accurate fashion.

There is no single summary that accurately captures all properties of a dataset, 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 items arrive. Given that it is typically not feasible to load all the data into memory on one machine, we need summaries that can be constructed incrementally. That is, we seek summaries that can be built by observing each individual 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 dataset.

Note that the notion of summarization is distinct from that of *compression*. In general, 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 compactly storing the data, or reducing the data transmission time. However, in general, there is no guarantee of significant size reduction from compression. The compressed form is also typically difficult to analyze, 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 some limited loss of fidelity: typically, based on the human perception of multimedia data, such as audio or video. Summarization aims to provide only small loss of fidelity, but measured along other dimensions; summaries do not necessarily provide a way to reconstruct even an approximation 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 given by two numbers, we can extract the average temperature. This summary is easy to update incrementally, and can also be combined with a corresponding summary of different data 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.

This work aims to describe and explain the summaries that have been developed to deal with big data, and to compare summaries for similar goals in terms of the forms of data that they accept, and their flexibility of use. It follows a fairly technical approach, describing each summary in turn. It lists the type of data that can be summarized, and what operations can be performed on the summary to include more data in it, and to extract information about the summarized data. We assume some familiarity with mathematical and computer science concepts, but provide some necessary background in subsequent sections.

1.2 Preliminaries

This section lays down some of the basics of working with summaries: the kinds of data that they can take as inputs; the operations that may be performed on the summaries during their use; and the types of guarantees they provide over their output.

1.2.1 Data Models

In this volume, we focus on datasets that arise from the aggregation of many small pieces of data. That is, the challenge arises from the scale of billions or trillions of simple observations. This matches the motivating applications

described previously: high-frequency sensor readings, social network activities, transactions, and so on all have a moderate number of different types, but potentially huge quantities of each type. The summaries we describe will operate on a large number of "tuples" of a common type, which collectively describe a complex whole.

The types of data we consider are therefore each quite simple, and it is their scale that presents the challenge for summarization. We describe the types of data in somewhat abstract terms, with the understanding that these can be mapped onto the specific applications when needed.

Set Data. The simplest form of data we consider is a set of items. That is, the input forms a set A, as a subset of some universe of possible items U. For example, U could be the set of 64-bit integers (denoting, perhaps, serial numbers of items), and each item x in the data is then some particular 64-bit integer.

A very basic summary over set data is a random sample. A random sample is a quite general-purpose summary in the sense that it is useful for answering many possible questions about the underlying set A, although the accuracy may not be satisfactory. For example, a basic query that we may wish to pose on a set A is whether a particular item x is present in A, i.e., a *membership* query; or, for two sets A and B, how similar (the notion will be made more precise later) they are. Random samples can be used in place of the full datasets for answering these queries, but clearly will frequently make errors. The majority of the work on data summarization is thus devoted to constructing summaries targeted at certain specific queries, usually with (much) better accuracies than random samples.

Problems on sets often get more challenging if the same item may be fed into the summary multiple times, while A is still considered as a set, i.e., duplicates should be removed. In this case, even counting the cardinality of A becomes nontrivial, if we do not want the summary to store every distinct input item.

Multiset Data. With set data, we typically assume the semantics that an item is either present or absent from the set. Under the multiset semantics, each item has a multiplicity. That is, we count the number of occurrences of each item. Again, the input is supported over a set U. Now, queries of interest relate to the multiplicity of items: how many occurrences of x are there in the data? Which items x occur most frequently?

It is sometimes convenient to think of multiset data as defining a vector of values, v. Then v_x denotes the multiplicity of item x in the input. Natural

queries over vectors include asking for the (Euclidean) norm of the vector, the distance between a pair of vectors, or the inner-product between two vectors. The accuracy of such estimators is often expressed in terms of the ℓ_p norm of the vector, $\|v\|_p$, where

$$\|v\|_p = \left(\sum_{i \in U} |v_i|^p\right)^{1/p}.$$

Important special cases include the Euclidean norm, $\|v\|_2$, and the Manhattan norm, $\|v\|_1$ (the sum of absolute values). We may also abuse notation and make reference to the ℓ_0 norm, sometimes called the Hamming norm, which is defined as $\|v\|_0 = |\{i : v_i \neq 0\}|$. This counts the number of nonzero entries in the vector v, i.e., the number of *distinct* items in the multiset. When dealing with skewed data, that is, where a few items have much larger count than others, we sometimes give bounds in terms of the residual ℓ_p norm. This is denoted as $\|v\|_p^{\mathrm{res}(k)}$, where, if we reindex v so that v_i is the ith largest (absolute) value, then

$$\|v\|_p^{\mathrm{res}(k)} = \left(\sum_{i=k+1}^{|U|} |v_i|^p\right)^{1/p}.$$

That is, the ℓ_p norm after removing the k largest entries of v.

Weighted Multiset Data. More generally, input describing a multiset may arrive with corresponding weights. This can represent, for example, a customer buying several instances of the same item in a single transaction. The multiplicity of the item across the whole input is the sum of all weights associated with it. The vector representation of the multiset naturally models the weighted case well, where v_i is the sum of weights of item i processed by the summary. The preceding queries all make sense over this style of input – to find the total weight for a given item, or the items with the largest total weights. Guarantees for summaries may be expressed in terms of vector norms such as $\|v\|_2$ or $\|v\|_1$. Different summaries can cope with different constraints on the weights: whether the weights should be integral, or can be arbitrary.

Of some concern is whether a summary allows *negative* weights. A negative weight corresponds to the removal of some copies of an item. Some summaries only tolerate nonnegative weights (the positive weights case), while others allow arbitrary positive and negative weights (which we call the general weights case). Lastly, a few summaries work in the "strict" case, where positive and negative weights are permitted, provided that the final weight

of every item is nonnegative when the summary is interrogated. By contrast, in the general case, we allow the multiplicity of an item to be negative. For the positive weights and strict cases, guarantees may be given in terms of $W = \|v\|_1$, the sum of the weights. Some summaries have guarantees in terms of $W^{\mathrm{res}(k)} = \|v\|_1^{\mathrm{res}(k)}$, the weight of the input (in the positive weight or strict case) after removing the k heaviest weights.

Matrices. Going beyond vectors, we may have data that can be thought of as many different vectors. These can be naturally collected together as large matrices. We are typically interested in $n \times d$ matrices M where both n and d are considerably large. In some cases, one or other of n and d is not so large, in which case we have a "short, fat matrix" or a "tall, skinny matrix," respectively.

As with the vector case, the constraints on the data can affect what is possible. Are the entries in the matrix integer or real valued? Is each entry in the matrix seen once only, or subject to multiple additive updates? Are entries seen in any particular order (say, a row at time), or without any order? Guarantees may be given in terms of a variety of matrix norms, including entrywise norms, such as the Frobenius norm,

$$\|M\|_F = \sqrt{\sum_{i,j} M_{i,j}^2},$$

or the p-norm, taken over unit norm vectors x,

$$\|M\|_p = \sup_{\|x\|_p=1} \|Mx\|_p.$$

Ordered Data. When U has a total order – namely, given any two items, we can compare them and determine which is the greater and which is the lesser under the order – we can formulate additional queries. For example, *how many occurrences of items in a given range are there (range queries)?*; *what is the median of the input?*; and more generally, *what does the data distribution look like on U?*

Some summaries manipulate items only by comparison, that is, given two items, checking whether one is greater or less than the other, or the two are equal. These summaries are said to be *comparison based*. They thus do not need to assume a fixed universe U beforehand, which is useful when dealing with, e.g., variable-length strings or user-defined data types.

Geometric Data. Multidimensional geometric data naturally arise in big data analytics. Any point on earth is characterized by latitude and longitude; a point

in space has three coordinates. More importantly, many types of multidimensional data can be interpreted and analyzed geometrically, although they are not inherently geometric by nature. For example, we may see readings which include temperature, pressure, and humidity. In data mining, various features can be extracted from an object, which map it to a high-dimensional point.

Over such data, the summary may support range queries, which could generalize one-dimensional ranges in different ways such as axis-parallel rectangles, half-spaces, or simplexes. Moreover, one could ask for many interesting geometric properties to be preserved by the summary, for example, the diameter, the convex hull, the minimum enclosing ball, pairwise distances, and various clusterings.

Graph Data. A graph represents a different kind of multidimensional data, where each input item describes an edge in a graph. Typically, the set of possible nodes V is known upfront, and each edge is a member of $V \times V$. However, in some cases V is defined implicitly from the set of edges that arrive. Over graphs, typical queries supported by summaries may be to approximate the distance between a pair of nodes, determine the number of connected components in the graph, or count the number of a particular subgraph, such as counting the number of triangles.

1.2.2 Operations on Summaries

For uniformity of presentation, each summary we describe typically supports the same set of basic operations, although these have different meanings for each summary. These basic operations are INITIALIZE, UPDATE, MERGE, and QUERY. Some summaries additionally have methods to CONSTRUCT and COMPRESS them.

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 formed of many data items. Ideally, this should be 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.

Many summaries described in this book support not only adding a new item to the summary, but also deleting a previously inserted item. To maintain uniformity, we treat a deletion as an UPDATE operation with a negative multiplicity. Examples include the Count-Min Sketch (Section 3.4), Count Sketch (Section 3.5), and the AMS Sketch (Section 3.6). This usually follows from the fact the summary is a linear transformation of the multiplicity vector representing the input, and such summaries are often called *linear sketches*. This concept is discussed in more detail toward the end of the book (Section 9.3.4).

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 usually reduces 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 are 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 typically pick one primary question to answer with the QUERY operation, and then discuss the other ways in which the summary can be used.

CONSTRUCT. We can always construct a summary by adding items one by one into the summary using the UPDATE and MERGE operations. However, for a few summaries, UPDATE is expensive, complicated, or even impossible. In these cases, we will describe how to CONSTRUCT the summary from the given input in an offline setting.

Compress. Some summaries also provide an additional operation which seeks to Compress the data structure. This is the case when the effect of Update and Merge operations allows the size of the summary to grow. In this case, Compress will aim to reduce the size of the summary as much as possible, while retaining an accurate representation. However, since the time cost for this operation may be higher than Update, it is not performed with every Update operation, but on a slower schedule, say after some number of Update operations have been performed.

A Simple Example: Counts, Sums, Means, Variances. We give an illustration of how these operations apply to the simple case of keeping counts. These give a first example of a summary allowing us to track the number of events that have been observed. Counters also easily allow us to track the sum of a sequence of weights, find their mean, and compute the observed variance/standard deviation.

We will illustrate the use of a counter c, and a sum of weights w, as well as a sum of squared weights s. The Initialize operation sets all of these to zero. Given an update of an item i, with a possible weight w_i, we can Update c by incrementing it: $c \leftarrow c + 1$. The sum of weights is updated as $w \leftarrow w + w_i$, and the sum of squared weights as $s \leftarrow s + w_i^2$. To Merge together two counter summaries, we can simply sum the corresponding values: the merge of c_1 and c_2 is $c_1 + c_2$, the merge of w_1 and w_2 is $w_1 + w_2$, and the merge of s_1 and s_2 is $s_1 + s_2$. We can apply different Query operations to obtain different aggregates: the total count of all the updates and the total sum of all the weights are simply the final values of c and w, respectively. The mean weight is given by w/c, and the variance of the weights is $s/w - (w/c)^2$.

1.2.3 Models of Computation

Traditionally, computer science has focused on the random access machine (RAM) model of computation to study algorithms and data structures. This abstraction is a good match for single-threaded computation on a single machine, but other models are required to fit computation on large volumes of data. The summaries that we describe are flexible and can be implemented in a variety of different settings.

The Streaming Model. The streaming model of computation considers data that arrive as a massive sequence of discrete observations, which collectively describe the data. For example, we might think of the data as describing a