Distributed Data Classification in Sensor Networks

Ittay Eyal
Technion, Israel Institute of Technology
Haifa 32000
Israel
ittay@tx.technion.ac.il

Idit Keidar
Technion, Israel Institute of Technology
Haifa 32000
Israel
idish@ee.technion.ac.il

Raphael Rom
Technion, Israel Institute of Technology
Haifa 32000
Israel
rom@ee.technion.ac.il

ABSTRACT

Low overhead analysis of large distributed data sets is necessary for current data centers and for future sensor networks. In such systems, each node holds some data value, e.g., a local sensor read, and a concise picture of the global system state needs to be obtained. In resource-constrained environments like sensor networks, this needs to be done without collecting all the data at any location, i.e., in a *distributed* manner. To this end, we define the *distributed classification problem*, in which numerous interconnected nodes compute a *classification* of their data, i.e., partition these values into multiple collections, and describe each collection concisely.

We present a *generic algorithm* that solves the distributed classification problem and may be implemented in various topologies, using different classification types. For example, the generic algorithm can be instantiated to classify values according to distance, like the famous k-means classification algorithm.

However, the distance criterion is often not sufficient to provide good classification results. We present an instantiation of the generic algorithm that describes the values as a *Gaussian Mixture* (a set of weighted normal distributions), and uses machine learning tools for classification decisions. Simulations show the robustness and speed of this algorithm.

We prove that any implementation of the generic algorithm converges over any connected topology, classification criterion and collection representation, in fully asynchronous settings.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: COMPUTER-COMMUNICATION NETWORKS—*Distributed Systems*

General Terms

Algorithms, Theory.

Keywords

Distributed classification, Gossip.

PODC'10, July 25–28, 2010, Zurich, Switzerland.

1. INTRODUCTION

To analyze large data sets, it is common practice to employ *classification* [6]: In classification, the data values are *partitioned* into several *collections*, and each collection is described concisely using a *summary*. This classical problem in machine learning is solved using various heuristic techniques, which typically base their decisions on a view of the complete data set, stored in some central database.

However, it is sometimes necessary to perform classification on data sets that are distributed among a large number of nodes. For example, in a grid computing system, load balancing can be implemented by having heavily loaded machines stop serving new requests. But this requires analysis of the load of all machines. If, e.g., half the machines have a load of about 10%, and the other half is 90% utilized, the system's state can be summarized by partitioning the machines into two collections — lightly loaded and heavily loaded. A machine with 60% load is associated with the heavily loaded collection, and should stop taking new requests. But, if the collection averages were instead 50% and 80%, it would have been associated with the former, i.e., lightly loaded, and would keep serving new requests. Another scenario is that of sensor networks with thousands of nodes monitoring conditions like seismic activity or temperature [1, 19].

In both of these examples, there are strict constraints on the resources devoted to the classification mechanism. Large-scale computation clouds allot only limited resources to monitoring, so as not to interfere with their main operation, and sensor networks use lightweight nodes with minimal hardware. These constraints render the collection of all data at a central location infeasible, and therefore rule out the use of centralized classification algorithms.

In this paper, we address the problem of *distributed classification*. To the best of our knowledge, this paper is the first to address this problem in a general fashion. A more detailed account of previous work appears in Section 2, and a formal definition of the problem appears in Section 3.

A solution to distributed classification ought to summarize data within the network. There exist distributed algorithms that calculate scalar aggregates, such as sum and average, of the entire data set [13, 10]. In contrast, a classification algorithm must partition the data into collections, and summarize each collection separately. In this case, it seems like we are facing a Catch-22 [12]: Had the nodes had the summaries, they would have been able to partition the values by associating each one with the summary it fits best. Alternatively, if each value was labeled a collection identifier, it

would have been possible to distributively calculate the summary of each collection separately, using the aforementioned aggregation algorithms.

In Section 4 we present a generic distributed classification algorithm to solve this predicament. Our algorithm captures a large family of algorithms that solve various instantiations of the problem — with different approaches to classification, classifying values from any multidimensional domain and with different data distributions, using various summary representations, and running on arbitrary connected topologies. In our algorithm, all nodes obtain a classification of the complete data set without actually hearing all the data values. The double bind described above is overcome by implementing adaptive compression: A classification can be seen as a lossy compression of the data, where a collection of similar values can be described succinctly, whereas a concise summary of dissimilar values loses a lot of information. Our algorithm tries to distribute the values between the nodes. At the beginning, it uses minimal compression, since each node has only little information to store and send. Once a significant amount of information is obtained, a node may perform efficient compression, joining only similar values. A common approach to classification, taken for example by the k-means algorithm, summarizes each collection by its *centroid* (average of the values in the collection) and partitions the values based on distance. Such a solution is a possible implementation of our generic algorithm.

Since the summary of collections as centroids is often insufficient in real life, machine learning solutions typically also take the variance into account, and summarize values as a weighted set of Gaussians (normal distributions), which is called a *Gaussian Mixture (GM)* [18]. In Section 5, we present a novel distributed classification algorithm that employs this approach, also as an instance of our generic algorithm. The GM algorithm makes classification decisions using a popular machine learning heuristic, *Expectation Maximization (EM)* [5]. We present in Section 5.3 simulation results demonstrating the effectiveness of this approach. These results show that the algorithm converges with high speed. It can provide a rich description of multidimensional data sets. Additionally, it can detect and remove outlying erroneous values, thereby enabling robust calculation of the average.

The centroids and GM algorithms are but two examples of our generic algorithm; in all instances, nodes independently strive to estimate the classification of the data. This raises a question that has not been dealt with before: does this process converge? One of the main contributions of this paper, presented in Section 6, is a formal proof that indeed any implementation of our generic algorithm converges, s.t. all nodes in the system learn *the same* classification of the complete data set. We prove that convergence is ensured under a broad set of circumstances: arbitrary asynchrony, an arbitrary connected topology, and no assumptions on the distribution of the values.

Note that in the abstract settings of the generic algorithm, there is no sense in defining the destination classification the algorithm converges to precisely, or in arguing about its quality, since these are application-specific and usually heuristic in nature. Additionally, due to asynchrony and lack of constraints on topology, it is also impossible to bound the convergence time.

In summary, this paper makes the following contributions:

- It formally defines the problem of classification in a distributed environment (Section 3).
- It provides a generic algorithm that captures a range of algorithms solving this problem in a variety of settings (Section 4).
- It provides a novel distributed classification algorithm based on Gaussian Mixtures, which uses machine learning techniques to make classification decisions (Section 5).
- It proves that the generic algorithm converges in very broad circumstances, over any connected topology, using any classification criterion, in fully asynchronous settings (Section 6).

2. RELATED WORK

Kempe et al. [13] and Nath et al. [16] present approaches for calculating aggregates such as sums and means using gossip. These approaches cannot be directly used to perform classification, though this work draws ideas from [13], in particular the concept of weight diffusion, and the tracing of value weights.

In the field of machine learning, classification has been extensively studied for centrally available data sets (see [6] for a comprehensive survey). In this context, parallelization is sometimes used, where multiple processes classify partial data sets. Parallel classification differs from distributed classification in that all the data is available to all processes, or is carefully distributed among them, and communication is cheap.

Centralized classification solutions typically overcome the Catch-22 issue explained in the introduction by running multiple iterations. They first estimate a solution, and then try to improve it by re-partitioning the values to create a better classification. K-means [15] and Expectation Maximization [5] are examples of such algorithms. Datta et al. [4] implement the k-means algorithm distributively, whereby nodes simulate the centralized version of the algorithm. Kowalczyk and Vlassis [14] do the same for Gaussian Mixture estimation by having the nodes distributively simulate Expectation Maximization. These algorithms require multiple aggregation iterations, each similar in length to one complete run of our algorithm. The message size in these algorithms is similar to ours, dependent only on the parameters of the dataset, and not on the number of nodes. Finally, they demonstrate convergence through simulation only, but do not provide a convergence proof.

Haridasan and van Renesse [11] and Sacha et al. [17] estimate distributions in sensor networks by estimating histograms. Unlike this paper, these solutions are limited to single dimensional data values. Additionally, both use multiple iterations to improve their estimations. While these algorithms are suitable for certain distributions, they are not applicable for classification, where, for example, small sets of distant values should not be merged with others. They also do not prove convergence.

3. MODEL AND PROBLEM DEFINITIONS

3.1 Network Model

The system consists of a set of n nodes, connected by communication channels, s.t. each node i has a set of neighbors $\texttt{neighbors}_i \subset \{1, \cdots, n\}$, to which it is connected. The channels form a static directed connected network. Communication channels are asynchronous but reliable links: A node may send messages on a link to a neighbor, and eventually every sent message reaches its destination. Messages are not duplicated and no spurious messages are created.

Time is discrete, and an execution is a series of events occurring at times $t = 0, 1, 2, \cdots$.

3.2 The Distributed Classification Problem

At time 0, each node i takes an input val_i — a value from a domain $\mathcal{D}$. In all the examples in this paper, $\mathcal{D}$ is a d-dimensional Cartesian space $\mathcal{D} = \mathbb{R}^d$ (with $d \in \mathbb{N}$). However, in general, $\mathcal{D}$ may be any domain.

A *weighted value* is a pair $\langle val, \alpha \rangle \in \mathcal{D} \times (0, 1]$, where α is a *weight* associated with a value *val*. We associate a weight of 1 to a whole value, so, for example, $\langle val_i, 1/2 \rangle$ is half of node i's value. A set of weighted values is called a *collection*:

Definition 1 (Collection). *A collection c is a set of weighted values with unique values. The collection's weight, $c.weight$, is the sum of the value weights:*

$$c.weight \triangleq \sum_{\langle val,\alpha \rangle \in c} \alpha \quad .$$

A collection may be *split* into two new collections, each consisting of the same values as the original collection, but associated with half their original weights. Similarly, multiple collections may be *merged* to form a new one, consisting of the union of their values, where each value is associated with the sum of its weights in the original collections.

A collection can be concisely described by a *summary* in a domain $\mathcal{S}$, using a function f that maps collections to their summaries: $f : (\mathcal{D} \times (0, 1])^* \to \mathcal{S}$. The domain $\mathcal{S}$ is a pseudo-metric space (like metric, except the distance between distinct points may be zero), with a distance function $d_S : \mathcal{S}^2 \to \mathbb{R}$.

A collection c may be partitioned into several collections, each holding a subset of its values and summarized separately[1]. The set of weighted summaries of these collections is called a *classification* of c. Weighted values in c may be split among collections, so that different collections contain portions of a given value. The sum of weights associated with a value *val* in all collections is equal to the sum of weights associated with *val* in c. Formally:

Definition 2 (Classification). *A Classification C of a collection c into J collections $\{c_j\}_{j=1}^{J}$ is the set of weighted summaries of these collections: $C = \{\langle f(c_j), c_j.weight \rangle\}_{j=1}^{J}$ s.t.*

$$\forall val: \sum_{\langle val,\alpha \rangle \in c} \alpha \;=\; \sum_{j=1}^{J} \left(\sum_{\langle val,\alpha \rangle \in c_j} \alpha \right).$$

A classification of a value set $\{val_j\}_{j=1}^{l}$ is a classification of the collection $\{\langle val_j, 1 \rangle\}_{j=1}^{l}$.

[1]Note that partitioning a collection is different from splitting it, because, when a collection is split, each part holds the same values.

The number of collections in a classification is bounded by a system parameter k.

A classification algorithm strives to partition the samples into classes in a way that optimizes some criterion, for example, minimizes some distance metric among values assigned to the same class (as in k-means). In this paper, we are not concerned with the nature of this criterion, and leave it up to the application to specify the choice thereof

A classification algorithm maintains at every time t a classification $classification_i(t)$, yielding an infinite series of classifications. For such a series, we define convergence:

Definition 3 (Classification Convergence). *A series of classifications*

$$\left\{ \{\langle f(c_j(t)), c_j(t).weight \rangle\}_{j=1}^{J_t} \right\}_{t=1}^{\infty}$$

converges to a destination classification, which is a set of l collections $\{dest_x\}_{x=1}^{l}$, if for every $t \in 0, 1, 2, \cdots$ there exists a mapping ψ_t between the J_t collections at time t and the l collections in the destination classification $\psi_t : \{1, \cdots, J_t\} \to \{1, \cdots, l\}$, such that:

1. *The summaries converge to the collections to which they are mapped by ψ_t:*

$$\max_j \left\{ d_S(f(c_j(t)), f(dest_{\psi_t(j)})) \right\} \xrightarrow{t \to \infty} 0 \; .$$

2. *For each collection x in the destination classification, the relative amount of weight in all collections mapped to x converges to x's relative weight in the classification:*

$$\forall 1 \le x \le l : \frac{\sum_{\{j|\psi_t(j)=x\}} c_j(t).weight}{\sum_{j=1}^{J_t} c_j(t).weight} \xrightarrow{t \to \infty} \to \frac{dest_x.weight}{\sum_{y=1}^{l} dest_y.weight} \; .$$

We are now ready to define the problem addressed in this paper, where a set of nodes strive to learn a common classification of their inputs. As previous works on aggregation in sensor networks [13, 16, 2], we define a converging problem, where nodes continuously produce outputs, and these outputs converge to such a common classification.

Definition 4 (Distributed Classification). *Each node i takes an input val_i at time 0 and maintains a classification $\texttt{classification}_i(t)$ at each time t, s.t. there exists a classification of the input values $\{val_i\}_{i=1}^{n}$ to which the classifications in all nodes converge.*

4. GENERIC CLASSIFICATION ALGORITHM

We now present our generic algorithm that solves the Distributed Classification Problem. At each node, the algorithm builds a classification, which converges over time to one that describes all input values of all nodes. In order to avoid excessive bandwidth and storage consumption, the algorithm maintains classifications as weighted summaries of collections, and not the actual sets of weighted values. It begins with a classification of its own input value. It then periodically splits its classification into two new ones, which

Algorithm 1: Generic Distributed Data Classification Algorithm. Dashed frames show auxiliary code.

1 **state**
2 $classification_i$, initially $\{\langle$ `valToSummary` $(val_i), 1$ [, e_i] $\rangle\}$

3 Periodically do **atomically**
4 Choose $j \in neighbors_i$ `(Selection has to ensure fairness)`
5 $old \leftarrow classification_i$
6 $classification_i \leftarrow \bigcup_{c \in old}\{\langle c.\texttt{summary}, half(c.weight)$ [, $\frac{half(c.weight)}{c.weight} \cdot c.aux$] $\rangle\}$
7 send $(j, \bigcup_{c \in old}\{\langle c.\texttt{summary}, c.weight - half(c.weight)$ [, $\left(1 - \frac{half(c.weight)}{c.weight}\right) \cdot c.aux$] $\rangle\})$

8 Upon receipt of *incoming* do **atomically**
9 $bigSet \leftarrow classification_i \cup incoming$
10 $M =$ `partition` $(bigSet)$
11 $classification_i \leftarrow \bigcup_{x=1}^{|M|}\left\{\langle \texttt{mergeSet}\ \left(\bigcup_{c \in M_x}\{\langle c.\texttt{summary}, c.weight\rangle\}\right), \sum_{c \in M_x} c.weight$ [, $\sum_{c \in M_x} c.aux$] $\rangle\right\}$

12 **function** $half(\alpha)$
13 return the multiple of q which is closest to $\alpha/2$.

have the same summaries but half the weights of the originals; it sends one classification to a neighbor, and keeps the other. Upon receiving a classification from a neighbor, a node merges it with its own, according to an application-specific merge rule. The algorithm thus progresses as a series of merge and split operations.

In Section 4.1, we present the generic distributed classification algorithm. It is instantiated with a domain $\mathcal{S}$ of summaries used to describe collections, and with application-specific functions that manipulate summaries and make classification decisions. As an in-line example, we present an algorithm that summarizes collections as their *centroids* — the averages of their weighted values.

In Section 4.2, we enumerate a set of requirements on the functions the algorithm is instantiated with. We then show that in any instantiation of the generic algorithm with functions that meet these requirements, the weighted summaries of collections are the same as those we would have obtained, had we applied the algorithm's operations on the original collections, and then summarized the results.

4.1 Algorithm

The algorithm maintains at each node an estimate of the classification of the complete set of values. Each collection c is stored as a weighted summary. By slight abuse of terminology, we refer by the term collection to both a set of weighted values c, and its summary–weight pair

$$\langle c.\texttt{summary}, c.weight\rangle.$$

The algorithm for node i is shown in Algorithm 1 (at this stage, we ignore the parts in dashed frames). The algorithm is generic, and it is instantiated with $\mathcal{S}$ and the functions `valToSummary` , `partition` and `mergeSet` . *The functions of the centroids example are given in Algorithm 2. The summary domain $\mathcal{S}$ in this case is the same as the value domain, i.e., $\mathbb{R}^d$.*

Initially, each node produces a classification with a single collection, based on the single value it has taken as input (Line 2). The weight of this collection is 1, and its summary is produced by the function `valToSummary` $: \mathcal{D} \rightarrow \mathcal{S}$, which encapsulates f. *In the centroids example, the initial summary is the input value (Algorithm 2,* `valToSummary` *function).*

A node occasionally sends data to a neighbor (Algorithm 1, Lines 3–7): It first splits its classification into two new ones. For each collection in the original classification, there is a matching collection in each of the new ones, with the same summary, but with approximately half the weight. Weight is quantized, limited to multiples of a system parameter q $(q, 2q, 3q, \cdots)$. This is done in order to avoid a scenario where it takes infinitely many transfers of infinitesimal weight to transfer a finite weight from one collection to another (Zeno effect). We assume q is small enough to avoid quantization errors: $q \ll \frac{1}{n}$. In order to respect the quantization requirement, the weight is not multiplied by exactly 0.5, but by the closest factor for which the resulting weight is a multiple of q (function *half* in Algorithm 1). One of the collections is attributed the result of *half* and the other is attributed the complement, so that the sum of weights is equal to the original, and system-wide conservation of weight is maintained.

The node then keeps one of the new classifications, replacing its original one (Line 6), and sends the other to some neighbor j (Line 7). The selection of neighbors has to ensure fairness in the sense that in an infinite run, each neighbor is chosen infinitely often; this can be achieved, e.g., using round robin. Alternatively, the node may implement gossip communication patterns: It may choose a random neighbor and send data to it (push), or ask it for data (pull), or perform a bilateral exchange (push-pull).

When a message with a neighbor's classification reaches the node, an event handler (Lines 8–11) is called. It first combines the two classifications of the nodes into a set *bigSet* (Line 9). Then, an application-specific function `partition` divides the collections in *bigSet* into sets $M = \{M_x\}_{x=1}^{|M|}$ (Line 10). The collections in each of the sets in M are merged into a single collection, together forming the new classification of the node (Line 11). The summary of each merged collection is calculated by another application-speci-

Algorithm 2: Centroid Functions

function `valToSummary` (*val*)
  return *val*

function `mergeSet` (`collections`)

return $\left(\sum_{\substack{\langle avg, m\rangle \in \\ \texttt{collections}}} m \right)^{-1} \times \sum_{\substack{\langle avg, m\rangle \in \\ \texttt{collections}}} m \cdot avg$

function `partition` (*bigSet*)
  $M = bigSet$
  if there are sets in M with weight q then merge them arbitrarily with others
  while $|M| > k$ **do**
    let M_x and M_y be the (different) elements of M whose centroids are closest
    $M = M \setminus \{M_x, M_y\} \cup (M_x \cup M_y)$
  return M

fic function, `mergeSet` , and its weight is the sum of weights of the merged collections.

To conform with the restrictions of k and q, the partition function must guarantee that (1) $|M| \leq k$; and (2) no M_x includes a single collection of weight q (that is, every collection of weight q is merged with at least one other collection).

Note that the parameter k forces lossy compression of the data, since merged values cannot later be separated. At the beginning, only a small number of data values is known to the node, so it performs only a few (easy) classification decisions. As the algorithm progresses, the number of values described by the node's classification increases. By then, it has enough knowledge of the data set, so as to perform correct classification decisions, and achieve a high compression ratio without losing valuable data.

In the centroids algorithm, the summary of the merged set is the weighted average of the summaries of the merged collections, calculated by the implementation of `mergeSet` *shown in Algorithm 2. Merging decisions are based on the distance between collection centroids. Intuitively, it is best to merge close centroids, and keep distant ones separated. This is done greedily by* `partition` *(shown in Algorithm 2) which repeatedly merges the closest sets, until the k bound is reached.*

4.2 Auxiliaries and Instantiation Requirements

4.2.1 Instantiation requirements

To phrase the requirements, we describe a collection in $\langle \mathcal{D}, (0,1] \rangle^*$ as a vector in the *Mixture Space* — the space $\mathbb{R}^n$ (n being the number of input values), where each coordinate represents one input value. A collection is described in this space as a vector whose j'th component is the weight associated with val_j in that collection. A vector in the mixture space precisely describes a collection. We can therefore view f as a mapping from mixture space vectors of collections to collection summaries, i.e., $f : \mathbb{R}^n \to \mathcal{S}$. From this point on, we use f in this manner.

We define the distance function $d_M : (\mathbb{R}^n)^2 \to \mathbb{R}$ between two vectors in the mixture space to be the angle between them. Collections consisting of similar weighted values are close in the mixture space (according to d_M). Their summaries should be close in the summary space (according to d_S), with some scaling factor ρ. Simply put — collections consisting of similar values (i.e., close in d_M) should have similar summaries (i.e., close in d_S). Formally:

R1 For any input value set,

$$\exists \rho : \forall v_1, v_2 \in (0,1]^n : d_S(f(v_1), f(v_2)) \leq \rho \cdot d_M(v_1, v_2).$$

In the centroids algorithm, we define d_S to be the L^2 distance between the centroids of collections. This fulfills requirement R1 (a proof can be found in [9]).

In addition, operations on summaries must preserve the relation to the collections they describe. Intuitively, this means that maintaining summaries is similar to performing the various operations on the value set, and then summarizing the results.

R2 Values are mapped by f to their summaries:

$$\forall i, 1 \leq i \leq n : \texttt{valToSummary}\ (val_i) = f(e_i).$$

R3 Summaries are oblivious to weight scaling:

$$\forall \alpha > 0, v \in (0,1]^n : f(v) = f(\alpha v).$$

R4 Merging a summarized description of collections is equivalent to merging these collections and then summarizing the result:

$$\texttt{mergeSet}\ \left(\bigcup_{v \in V} \langle \{ f(v), \|v\|_1 \rangle \} \right) = f\left(\sum_{v \in V} v \right).$$

4.2.2 Auxiliary correctness

We show that the weighted summaries maintained by the algorithm to describe collections that are merged and split, indeed do so. To do that, we define an auxiliary algorithm. This is an extension of Algorithm 1 with the auxiliary code in the dashed frames. Collections are now triplets, containing, in addition to the summary and weight, the collection's mixture space vector $c.aux$.

At initialization (Line 2), the auxiliary vector at node i is e_i (a unit vector whose i'th component is 1). When splitting a collection (Lines 6–7), the vector is factored by about 1/2 (the same ratio as the weight). When merging a set of collections, the mixture vector of the result is the sum of the original collections' vectors (Line 11).

The following lemma shows that, at all times, the summary maintained by the algorithm is indeed that of the collection described by its mixture vector:

LEMMA 1 (AUXILIARY CORRECTNESS). *The generic algorithm, instantiated by functions satisfying R2–R4, maintains the following invariant: For any collection c either in a node's classification ($c \in \texttt{classification}_i$) or in transit in a communication channel, the following two equations hold:*

$$f(c.aux) = c.\texttt{summary} \quad (1)$$

$$\|c.aux\|_1 = c.weight \quad (2)$$

PROOF. By induction on the global states of the system.

Basis Initialization puts at time 0, at every node i the auxiliary vector e_i, a weight of 1, and the summary `valToSummary` of value i. Requirement R2 thus ensures that Equation 1 holds in the initial state, and Equation 2 holds since $\|e_i\| = 1$. Communication channels are empty.

Assumption At time $j-1$ the invariant holds.

Step Transition j may be either send or receive. Each of them removes collections from the set, and produces a collection or two. To prove that at time j the invariant holds, we need only show that in both cases the new collection(s) maintain the invariant.

Send We show that the mapping holds for the kept collection c_{keep}. A similar proof holds for the sent one c_{send}. Proof of Equation 1:

$$\begin{aligned}
&c_{\text{keep}}.\texttt{summary} \overset{\text{line 6}}{=} \\
&= c.\texttt{summary} \overset{\text{induction assumption}}{=} \\
&= f(c.aux) \overset{\text{R3}}{=} \\
&= f\left(\frac{half(c.weight)}{c.weight} \cdot c.aux\right) \overset{\text{auxiliary line 6}}{=} \\
&= f(c_{\text{keep}}.aux)
\end{aligned}$$

Proof of Equation 2:

$$\begin{aligned}
&c_{\text{keep}}.weight \overset{\text{line 6}}{=} \\
&= half(c.weight) = \\
&= \frac{half(c.weight)}{c.weight} \cdot c.weight \overset{\text{induction assumption}}{=} \\
&= \frac{half(c.weight)}{c.weight} \cdot \|c.aux\|_1 = \\
&= \left\| \frac{half(c.weight)}{c.weight} \cdot c.aux \right\|_1 \overset{\text{auxiliary line 6}}{=} \\
&= \|c_{\text{keep}}.aux\|_1
\end{aligned}$$

Receive We prove that the mapping holds for each of the m produced collections. Each collection c_x is derived from a set M_x. Proof of Equation 1:

$$\begin{aligned}
&c_x.\texttt{summary} \overset{\text{line 11}}{=} \\
&= \texttt{mergeSet}\left(\bigcup_{c\in M_x} \{\langle c.\texttt{summary}, c.weight\rangle\}\right) \overset{\text{induction assumption}}{=} \\
&= \texttt{mergeSet}\left(\bigcup_{c\in M_x} \{\langle f(c.aux), \|c.aux\|_1\rangle\}\right) \overset{\text{R4}}{=} \\
&= f\left(\sum_{c\in M_x} c.aux\right) \overset{\text{auxiliary line 11}}{=} \\
&= f\left(c_x.aux\right)
\end{aligned}$$

Proof of Equation 2:

$$\begin{aligned}
&c_x.weight \overset{\text{line 11}}{=} \\
&= \sum_{c\in M_x} c.weight \overset{\text{induction assumption}}{=} \\
&= \sum_{c\in M_x} \|c.aux\|_1 = \\
&= \left\| \sum_{c\in M_x} c.aux \right\|_1 \overset{\text{auxiliary line 11}}{=} \\
&= \|c_x.aux\|_1
\end{aligned}$$

□

5. GAUSSIAN CLASSIFICATION

When classifying value sets from a metric space, the centroids solution is seldom sufficient. Consider the example shown in Figure 1, where we need to associate a new value with one of two existing collections. Figure 1a shows the information that the centroids algorithm has for collections A and B, and a new value. The algorithm would associate the new value to collection A, on account of it being closer to its centroid. However, Figure 1b shows the set of values that produced the two collections. We see that it is more likely that the new value in fact belongs to collection B, since it has a much larger variance.

The field of machine learning suggests the heuristic of classifying data using a Gaussian Mixture (a weighted set of normal distributions), allowing for a rich and accurate description of multivariate data. Figure 1b illustrates the summary employed by GM: An ellipse depicts an equidensity line of the Gaussian summary of each collection. Given these Gaussians, one can easily classify the new value correctly.

We present in Section 5.1 the GM algorithm — a new distributed classification algorithm, implementing the generic one by representing collections as Gaussians, and classifications as Gaussian Mixtures. Contrary to the classical machine learning algorithms, ours performs the classification without collecting the data in a central location. Nodes use the popular machine learning tool of Expectation Maximization to make classification decisions (Section 5.2). A taste of the results achieved by our GM algorithm is given in Section 5.3 via simulation. It demonstrates the classification of multidimensional data and more. Note that due to the heuristic nature of EM, the only possible evaluation of our algorithm's quality is empirical.

5.1 Gaussian Mixture Classification

The summary of a collection is the tuple $\langle \mu, \sigma \rangle$, comprised of the average of the weighted values in the collection $\mu \in \mathbb{R}^d$ (where $\mathcal{D} = \mathbb{R}^d$ the value space), and their covariance matrix $\sigma \in \mathbb{R}^{d\times d}$. Together with the weight, a collection is described by a weighted Gaussian, and a classification consists of a weighted set of Gaussians, or a *Gaussian Mixture*.

The mapping f of a vector is the tuple $\langle \mu, \sigma \rangle$ of the average and covariance matrix of the collection represented by the normalized vector. We define d_S as in the centroids algorithm. The function `valToSummary` (*val*) returns a collection with an average equal to *val*, a zero covariance matrix, and a weight of 1. The function `mergeSet` takes a set

(a) Centroids and a new value (b) Gaussians and a new value

Figure 1: Associating a new value when collections are summarized (a) as centroids and (b) as Gaussians.

of weighted summaries and calculates the summary of the merged set. A complete description of the functions and a proof that they conform to R1–R4are deferred to the complete version of this paper [9].

5.2 Expectation Maximization Partitioning

To complete the description of the GM algorithm, we now explain the `partition` function. When a node has accumulated more than k collections, it needs to merge some of them. In principle, it would be best to choose collections to merge according to Maximum Likelihood, which is defined in this case as follows: We denote a Gaussian Mixture of x Gaussians *x-GM*. Given a too large set of $l \geq k$ collections, an *l-GM*, the algorithm tries to find the *k-GM* probability distribution for which the *l-GM* has the maximal likelihood. However, computing Maximum Likelihood is NP-hard. We therefore instead follow common practice and approximate it with the *Expectation Maximization* algorithm [15].

5.3 Simulation Results

Due to the heuristic nature of the Gaussian Mixture classification and of EM, the quality of their results is typically evaluated experimentally. In this section, we briefly demonstrate the effectiveness of our GM algorithm through simulation. First, we demonstrate the algorithm's ability to classify multidimensional data, which could be produced by a sensor network. Then, we demonstrate a possible application using the algorithm to calculate the average while removing outliers and coping with node failures. This result also demonstrates the convergence speed of the algorithm.

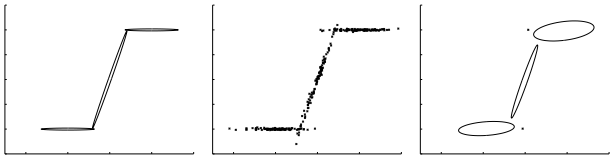
In both cases, we simulate a fully connected network of 1,000 nodes. Like previous works [7, 11], we measure progress in rounds, where in each round each node sends a classification to one neighbor. Nodes that receive classifications from multiple neighbors accumulate all the received collections and run EM once for the entire set.

More simulation results and analysis can be found in [8].

5.3.1 Multidimensional data classification

As an example input, we use data generated from a set of three Gaussians in $\mathbb{R}^2$. Values are generated according to the distribution shown in Figure 2a, where the ellipses are equidensity contours of normal distributions. This input might describe temperature readings taken by a set of sensors positioned on a fence located by the woods, and whose right side is close to a fire outbreak. Each value is comprised of the sensor's location x and the recorded temperature y. The generated input values are shown in Figure 2b. We run the GM algorithm with this input until its convergence; $k = 7$ and q is set by floating point accuracy.

The result is shown in Figure 2c. The ellipses are equidensity contours, and the x's are singleton collections (with zero variance). This result is visibly a usable estimation of the input data.



(a) Distribution (b) Values (c) Result

Figure 2: Gaussian Mixture classification example. The three Gaussians in Figure 2a were used to generate the data set in Figure 2b. The GM algorithm produced the estimation in Figure 2c.

5.3.2 Robustness

Outlier removal.

As an example application, we use the algorithm to calculate a statistically robust average. We consider a sensor network of 1,000 sensors reading values in $\mathbb{R}^2$. Most of these values are sampled from a given Gaussian distribution and we wish to calculate their average. Some values, however, are erroneous, and are unlikely to belong to this distribution. They may be the result of a malfunctioning sensor, or of a sensing error, e.g., an animal sitting on an ambient temperature sensor. These values are called *outliers* and should be removed from the statistics.

We use 950 values from the standard normal distribution, i.e., with a mean $(0, 0)$ and a unit covariance matrix I. Fifty additional values are distributed normally with covariance matrix $0.1 \cdot I$ and mean $(0, \Delta)$, with Δ ranging between 0 and 25. The distribution of all values is illustrated in Figure 3a.

For each value of Δ, the protocol is run until convergence. We use $k = 2$, so that each node has at most 2 collections at any given time — hopefully one for good values and one for outliers.

The results are shown in Figure 3b. The dotted line shows the average weight ratio belonging to outliers yet incorrectly assigned to the good collection. Outliers are defined to be values with probability density lower than $f_{min} = 5 \times 10^{-5}$ (for the standard normal distribution). The other two lines show the error in calculating the mean, where error is the average over all nodes of the distance between the estimated mean and the true mean $(0, 0)$. The solid line shows the

result of our algorithm, which removes outliers, while the dashed line shows the result of regular average aggregation, which does not.

We see that when the outliers are close to the good values, the number of misses is large — the proximity of the collections makes their separation difficult. However, due to the small distance, this mistake hardly influences the estimated average. As the outliers' mean moves further from the true mean, the identification of outliers becomes accurate and their influence is nullified.

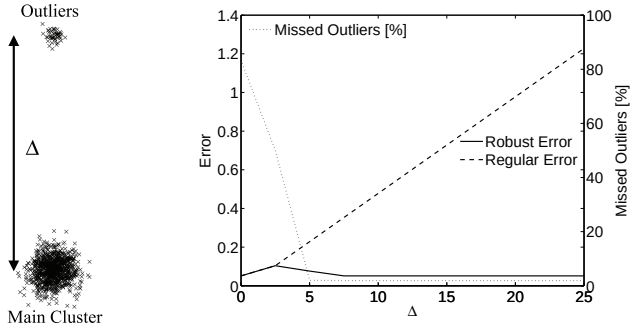


Figure 3: Effect of the separation of outliers on the calculation of the average: A 1,000 values are sampled from two Gaussian distributions (Figure 3a). As the outliers' distribution moves away from the good one, the regular aggregation error grows linearly. However, once the distance is large enough, our protocol can remove the outliers, which results in an accurate estimation of the mean.

Note that even for large Δ's, a certain number of outliers is still missed. These are values from the good distribution, relatively close to the main collection, yet with probability density lower than f_{min}. The protocol mistakingly considers these to be good values. Additionally, around $\Delta = 5$ the miss rate is dropped to its minimum, yet the robust error does not. This is due to the fact that bad values are located close enough to the good mean so that their probability density is higher than f_{min}. The protocol mistakes those to belong to f_G and allows them to influence the mean. That being said, for all Δ's, the error remains small, confirming the conventional wisdom that "classification is either easy or not interesting".

Crash robustness and convergence speed.

We next examine how crash failures impact the results obtained by our protocol. Figure 4 shows that the outlier removal mechanism is indifferent to crashes of nodes. The source data is similar to the one above, with $\Delta = 10$. After each round, each node crashes with probability 0.05. We show the average node estimation error of the mean in each round. As we have seen above, our protocol achieves a lower error then the regular one.

Figure 4 also demonstrates the convergence speed of our algorithm. With and without crashes, the convergence speed of our algorithm is equivalent to that of the regular average aggregation algorithm.

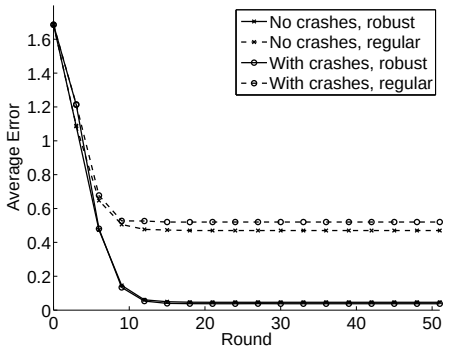


Figure 4: The effect of crashes on convergence speed and on the accuracy of the mean.

6. CONVERGENCE PROOF

We now prove that the generic algorithm presented in Section 4 solves the distributed classification problem. To prove convergence, we consider a pool of all the collections in the system, including all nodes and communication channels. This pool is in fact, at all times, a classification of the set of all input values. In Section 6.1 we prove that the pool of all collections converges, i.e., roughly speaking, it stops changing. Then, in Section 6.2, we prove that the classifications in all nodes converge to the same destination.

6.1 Collective Convergence

In this section, we ignore the distributive nature of the algorithm, and consider all the collections in the system (at both processes and communication channels) at time t as if they belonged to a single multiset $pool(t)$. A run of the algorithm can therefore be seen as a series of splits and merges of collections.

To argue about convergence, we first define the concept of collection descendants: for $t_1 \leq t_2$, a collection $c_2 \in pool(t_2)$ is a descendant of a collection $c_1 \in pool(t_1)$ if c_2 is equal to c_1, or the result of operations on c_1. Formally:

Definition 5 (Collection Genealogy). *We recursively define the descendants $desc(A, *)$ of a collection set $A \in pool(t)$. First, at t, the descendants are simply A. Next, consider $t' > t$.*

Assume the t' operation in the execution is splitting (and sending) (Lines 3–7) a set of collections $\{c_x\}_{x=1}^{l} \subset pool(t'-1)$. This results in two new sets of collections, $\{c_x^1\}_{x=1}^{l}$ and $\{c_x^2\}_{x=1}^{l}$, being put in $pool(t')$ instead of the original set. If a collection c_x is a descendant at $t'-1$, then the collections c_x^1 and c_x^2 are descendants at t'.

Assume the t' operation is a (receipt and) merge (Lines 8–11), then some m $(1 \leq m \leq k)$ sets of collections $\{M_x\}_{x=1}^{m}$ are merged and are put in $pool(t')$ instead of the merged ones. For every M_x, if any of its collections is a descendant at $t'-1$, then its merge result is a descendant at t'.

Formally:

$$desc(A,t') \triangleq$$
$$\triangleq \begin{cases} A & t' = t \\ desc(A,t'-1) \setminus \{c_x\}_{x=1}^{l} \cup \bigcup_{x=1}^{l}\{\{c_x^1, c_x^2\} | c_x \in desc(A,t'-1)\} & t' > t, \text{ operation } t' \text{ is split} \\ desc(A,t'-1) \setminus \bigcup_{x=1}^{m} M_x \cup \bigcup_{x=1}^{m}\{c_x | M_x \cap desc(A,t'-1) \neq \phi\} & t' > t, \text{ op. } t' \text{ is merge} \end{cases}$$

By slight abuse of notation, we write $v \in pool(t)$ when v is the mixture vector of a collection c, and $c \in pool(t)$; vector genealogy is similar to collection genealogy.

We now state some definitions and the lemmas used in the convergence proof. The proofs of some lemmas are deferred to the complete version of this paper [9]. We prove that, eventually, the descendants of each vector converge (normalized) to a certain destination. To do that, we investigate the angle between a vector v and the i'th axis unit vector, which we call the i'th *reference angle* and denote φ_i^v. We denote by $\varphi_{i,\max}(t)$ the maximal i'th reference angle over all vectors in the pool at time t:

$$\varphi_{i,\max}(t) \triangleq \max_{v \in \texttt{pool}(t)} \varphi_i^v \ .$$

LEMMA 2. *For $1 \leq i \leq n$, $\varphi_{i,max}(t)$ is monotonically decreasing.*

Since the maximal reference angles are bounded from below by zero, Lemma 2 shows that they converge, and we can define

$$\hat{\varphi}_{i,\max} \triangleq \lim_{t \to \infty} \varphi_{i,\max}(t) \ .$$

By slight abuse of terminology, we say that the i'th reference angle of a vector $v \in pool(t)$ *converges* to φ, if for every ε there exists a time t', after which the i'th reference angle of all of v's descendants is in the ε neighborhood of φ.

The next lemma shows that there exists a time after which the pool is partitioned into classes, and the vectors from each class merge only with one another. Moreover, the descendants of all vectors in a class converge to the same reference angle.

LEMMA 3 (CLASS FORMATION). *For every $1 \leq i \leq n$, there exists a time $t_{i,max}$ and a set of vectors*

$$V_{i,max} \subset pool(t_{i,max})$$

s.t. the i'th reference angles of the vectors converge to $\hat{\varphi}_{i,max}$, and their descendants are merged only with one another.

We next prove that the pool of auxiliary vectors converges:

LEMMA 4 (AUXILIARY COLLECTIVE CONVERGENCE). *The set of auxiliary variables in the system converges, i.e., there exists a time t, s.t. the normalized descendants of each vector in pool(t) converge to a specific destination vector, and merge only with descendants of vectors that converge to the same destination.*

PROOF. By Lemmas 2 and 3, for every $1 \leq i \leq n$, there exist a maximal i'th reference angle, $\hat{\varphi}_{i,\max}$, a time, $t_{i,\max}$, and a set of vectors, $V_{i,\max} \subset pool(t_{i,\max})$, s.t. the i'th reference angles of the vectors $V_{i,\max}$ converge to $\hat{\varphi}_{i,\max}$, and the descendants of $V_{i,\max}$ are merged only with one another.

The proof continues by induction. At $t_{i,\max}$ we consider the vectors that are not in $desc(V_{i,\max}, t_{i,\max})$. The descendants of these vectors are never merged with the descendants of the $V_{i,\max}$ vectors. Therefore, the proof applies to them with a new maximal i'th reference angle. This can be applied repeatedly, and since the weight of the vectors is bounded from below by q, we conclude that there exists a time t after which, for every vector v in the pool at time $t' > t$, the i'th reference of v converges. Denote that time $t_{conv,i}$.

Next, let $t_{conv} = \max\{t_{conv,i} | 1 \leq i \leq n\}$. After t_{conv}, for any vector in the pool, all of its reference angles converge. Moreover, two vectors are merged only if all of their reference angles converge to the same destination. Therefore, at t_{conv}, the vectors in $pool(t_{conv})$ can be partitioned into disjoint sets s.t. the descendants of each set are merged only with one another and their reference angles converge to the same values. For a class x of vectors whose reference angles converge to $(\varphi_i^x)_{i=1}^n$, its destination in the mixture space is the normalized vector $(\cos \varphi_i^x)_{i=1}^n$. □

We are now ready to derive the main result of this section.

COROLLARY 1. *The classification series pool(t) converges.*

PROOF. Lemma 4 shows that the pool of vectors is eventually partitioned into classes. This applies to the weighted summaries pool as well, due to the correspondence between summaries and auxiliaries (Lemma 1).

For a class of collections, define its destination collection as follows: Its weight is the sum of weights of collections in the class at t_{conv}, and its summary is that of the mixture space destination of the class's mixture vectors. Using requirement R1, it is easy to see that after t_{conv}, the classification series $pool(*)$ converges to the set of destination collections formed this way. □

6.2 Distributed Convergence

We show that the classifications in each node converge to the same classification of the input values.

LEMMA 5. *There exists a time t_{dist} after which each node holds at least one collection from each class.*

PROOF. First note that after t_{conv}, once a node has obtained a collection that converges to a destination x, it will always have a collection that converges to this destination, since it will always have a descendant of that collection — no operation can remove it.

Consider a node i that obtains a collection that converges to a destination x. It eventually sends a descendant thereof to each of its neighbors due to the fair choice of neighbors. This can be applied repeatedly and show that, due to the connectivity of the graph, eventually all nodes hold collections converging to x. □

Boyd et al. [3] analyzed the convergence of weight based average aggregation. The following lemma can be directly derived from their results:

LEMMA 6. *In an infinite run of Algorithm 1, after t_{dist}, at every node, the relative weight of collections converging to a destination x converges to the relative weight of x (in the destination classification).*

We are now ready to prove the main result of this section.

THEOREM 1. *Algorithm 1, with any implementation of the functions* `valToSummary` *,* `partition` *and* `mergeSet` *that conforms to Requirements R1–R4, solves the Distributed Classification Problem (Definition 4).*

PROOF. Corollary 1 shows that pool of all collections in the system converges to some classification *dest*, i.e., there exists mappings ψ_t from collections in the pool to the elements in *dest*, as in Definition 3. Lemma 5 shows that there exists a time t_{dist}, after which each node obtains at least one collection that converges to each destination.

After this time, for each node, the mappings ψ_t from the collections of the node at t to the *dest* collections show convergence of the node's classification to the classification *dest* (of all input values). Corollary 1 shows that the summaries converge to the destinations, and Lemma 6 shows that the relative weight of all collections that are mapped to a certain collection x in *dest* converges to the relative weight of x. □

7. CONCLUSION

We presented the problem of distributed data classification, where nodes obtain values and must calculate a classification thereof. We presented a generic distributed data classification algorithm that solves the problem efficiently by employing adaptive in-network compression. The algorithm is completely generic and captures a wide range of algorithms for various instances of the problem. We presented a specific instance thereof — the Gaussian Mixture algorithm, where collections are maintained as weighted Gaussians, and merging decisions are done using the Expectation Maximization heuristic. Finally, we provided a proof that any implementation of the algorithm converges.

Acknowledgements

We thank Yoram Moses and Nathaniel Azuelos for their valuable advice.

This work was partially supported by the Technion Funds for Security Research, by the Israeli Ministry of Industry, Trade and Labor Magnet Consortium, and by European Union Grant No. 216366.

8. REFERENCES

[1] G. Asada, M. Dong, T. Lin, F. Newberg, G. Pottie, W. Kaiser, and H. Marcy. Wireless integrated network sensors: Low power systems on a chip. In *ESSCIRC*, pages 9–16, sep 1998.

[2] Y. Birk, L. Liss, A. Schuster, and R. Wolff. A local algorithm for ad hoc majority voting via charge fusion. In *DISC*, pages 275–289, 2004.

[3] S. P. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: design, analysis and applications. In *INFOCOM*, pages 1653–1664, 2005.

[4] S. Datta, C. Giannella, and H. Kargupta. K-means clustering over a large, dynamic network. In *SDM*, 2006.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J. Royal Stat. Soc.*, 39(1):1–38, 1977.

[6] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, 2000.

[7] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. In *DSN*, pages 443–452, 2001.

[8] I. Eyal, I. Keidar, and R. Rom. Distributed clustering for robust aggregation in large networks. In *HotDep*, 2009.

[9] I. Eyal, I. Keidar, and R. Rom. Distributed data classification in sensor networks. Technical Report CCIT 762, 2010.

[10] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[11] M. Haridasan and R. van Renesse. Gossip-based distribution estimation in peer-to-peer networks. In *International Workshop on Peer-to-Peer Systems (IPTPS 08)*, February 2008.

[12] J. Heller. *Catch-22*. Simon & Schuster, 1961.

[13] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS*, pages 482–491, 2003.

[14] W. Kowalczyk and N. A. Vlassis. Newscast em. In *NIPS*, 2004.

[15] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[16] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys*, pages 250–262, 2004.

[17] J. Sacha, J. Napper, C. Stratan, and G. Pierre. Reliable distribution estimation in decentralised environments. *Submitted for Publication*, 2009.

[18] D. J. Salmond. Mixture reduction algorithms for uncertain tracking. Technical report, RAE Farnborough (UK), January 1988.

[19] B. Warneke, M. Last, B. Liebowitz, and K. Pister. Smart dust: communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, Jan 2001.