An efficient *local* Algorithm for Distributed Multivariate Regression in Peer-to-Peer Networks

Kanishka Bhaduri[*] Hillol Kargupta[†]

Abstract

This paper offers a **local** distributed algorithm for multivariate regression in large peer-to-peer environments. The algorithm is designed for distributed inferencing, data compaction, data modeling and classification tasks in many emerging peer-to-peer applications for bioinformatics, astronomy, social networking, sensor networks and web mining. Computing a global regression model from data available at the different peer-nodes using a traditional centralized algorithm for regression can be very costly and impractical because of the large number of data sources, the asynchronous nature of the peer-to-peer networks, and dynamic nature of the data/network. This paper proposes a two-step approach to deal with this problem. First, it offers an efficient local distributed algorithm that monitors the "quality" of the current regression model. If the model is outdated, it uses this algorithm as a feedback mechanism for rebuilding the model. The local nature of the monitoring algorithm guarantees low monitoring cost. Experimental results presented in this paper strongly support the theoretical claims.

1 Introduction

Multivariate Regression (MR) is a powerful statistical and machine learning tool that is widely used for prediction, classification, and data compression. Multivariate regression is relatively well understood given a sample of the data (input variables and corresponding target output values) at a single location. However, there are many emerging scenarios where data is distributed over a network of machines. Peer-to-Peer (P2P) networks offer one such example. P2P systems such as Gnutella, BitTorrents, e-Mule, Kazaa, and Freenet are increasingly becoming popular for many applications that go beyond downloading music without paying for it. Examples include P2P systems for network storage, web caching, searching and indexing of relevant documents and distributed network-threat analysis. The next generation of advanced P2P applications for bioinformatics [9] and client-side web mining [21][10] are likely to need support for advanced data analysis and mining. Performing basic operations like regression is very challenging in a P2P network because of the large number of data sources, the asynchronous nature of the P2P networks, and dynamic nature of the data.

This paper offers a local distributed algorithm for performing multivariate regression and monitoring the model in a P2P network. The approach is scalable, decentralized, asynchronous, and inherently based on in-network computation. The algorithmic framework is local, in the sense that the computation and communication load at each node is independent of the size or the number of nodes of the network. This is very important for the scalability of the algorithm in large P2P networks. The proposed methodology takes a two-step approach for building and maintaining MR models in P2P networks. The first step in our algorithm is the *monitoring phase* in which, given an estimate of the MR model to all the peers, they asynchronously track any change between the model and the global data using a provably correct local algorithm. The second step, known as the *computation phase*, uses the monitoring algorithm as a feedback loop for triggering a new round of MR model-building if necessary. The algorithm guarantees that as long as the MR model correctly represents the data, little computing and communication resources are spent for monitoring the environment. When the data undergoes a change in the underlying distribution and the MR model no longer represents it, the feedback loop indicates this and the model is rebuilt. Moreover, we also show that all the data need not be centralized to recompute the MR coefficients. We can do in-network aggregation for finding them; thereby using far less resources than brute force centralization. The specific contributions of this paper are as follows:

- To the best of the authors' knowledge this is one of the first attempts on developing a completely asynchronous and local algorithm for doing multi-variate regression in P2P networks which is robust to data and network changes.
- Besides this, we have also considerably enhanced our earlier work on a highly efficient and local algorithm for tracking the L2 norm of the global average vector.
- Most of the previous work in the literature focuses on linear regression in distributed environments. Our

[*]CSEE Dept, UMBC, kanishk1@cs.umbc.edu
[†]CSEE Dept, UMBC, and AGNIK LLC, hillol@cs.umbc.edu

technique can be applied to most types of common multivariate regression.

The rest of the paper is organized as follows. Related background material is presented in Section 2. Section 3 introduces the notations and problem definition. Section 4 presents the MR monitoring algorithm, while Section 5 discusses the MR computation problem. Section 3.3 defines local algorithms and analyzes the local behavior of our algorithm. Experimental results are presented in Section 6. Finally, Section 7 concludes this paper.

2 Background

This section provides the necessary background material.

2.1 Approach Statistical models can be built and updated from distributed data in various ways. The *periodic* approach is to simply rebuild the model from time to time. The *incremental* approach is to update the model whenever the data changes. Lastly, the *reactive* approach, what we propose here, is to monitor the change, and rebuild the model only when it no longer suits the data. The *periodic* approach can be highly inefficient since, there is the risk of wasting resources even if the data is stationary and also the of risk model inaccuracy if the updating is delayed. *Incremental* algorithms are very efficient; however their major drawback is that a separate algorithm needs to be handcrafted for every problem. Data driven *reactive* algorithms are efficient, simple and can accommodate a wide variety of function computation.

The work presented in this paper considers building and updating regression models from data distributed over a P2P network where each peer contains a subset of the data tuples. In the distributed data mining literature, this is usually called the horizontally partitioned or homogeneously distributed data scenario. Building a global regression model (defined on the union of all the data of all the peers) in large-scale networks and maintaining it is a vital task. Consider a network where there are a number of nodes (by node we mean peers, sensors, grid components etc.) and each node gets a stream of tuples (can be sensor readings, music files etc.) every few seconds thereby generating huge volume of data. We may wish to build a regression model on the global data to (1) compactly represent the data and (2) predict the value of a target variable. This is difficult since the data is distributed and more so because it is dynamic. Centralization obviously does not work because the data may change at a faster rate than the rate at which it can be centralized. Local algorithms are an excellent choice in such scenarios since in a local algorithm, each peer computes the result based on the information from only a handful of nearby neighbors. Hence local algorithms are highly scalable and guarantee that once the computation terminates, each node will have the correct regression model. Therefore, such an algorithm will enable the user to monitor regression models using low resources.

2.2 Related Work The work presented in this paper is related to two main bodies of literature - multivariate regression and computation in large distributed environments.

2.2.1 Distributed Multi-variate Regression: The problem of distributed multivariate regression has been addressed by many researchers till date. Hershberger et al. [16] considered the problem of performing global MR in a vertically partitioned data distribution scenario. The authors propose a wavelet transform of the data such that, after the transformation, effect of the cross terms can be dealt with easily. The local MR models are then transported to the central site and combined to form the global MR model. Such synchronized techniques will not scale in large, asynchronous systems such as modern P2P networks.

Many researchers have looked into the problem of doing distributed MR using distributed kernel regression techniques such as Guestrin et al. [15] and Predd et al. [25]. The algorithm presented by Guestrin et al. [15] performs linear regression in a network of sensors using in-network processing of messages. Instead of transmitting the raw data, the proposed technique transmits constraints only, thereby reducing the communication complexity drastically. Similar to the work proposed here, their work also uses local rules to prune messages. However the major drawback is that their algorithm is not suitable for dynamic data. It will be very costly if the data changes since, as the authors point out, that two passes are required over the entire network to make sure that the effect of the measurements of each node are propagated to every other node. Moreover, contrary to the broad class of problems that we can solve using our technique, their technique is only applicable for solving the linear regression problem.

Meta-learning is an interesting class of algorithms typically used for supervised learning. In a meta learning, such as bagging [7] or boosting [14] many models are induced from different partitions of the data and these “weak” models are combined using a second level algorithm which can be as simple as taking the average output of the models for any new sample. Such a technique is suitable for inducing models from distributed data as proposed by Stolfo et al. [27]. The basic idea is to learn a model at each site locally (no communication at all) and then, when a new sample comes, predict the output by simply taking an average of the local outputs. Xing et al. [30] present such a framework for doing regression in heterogenous datasets. However, these techniques perform poorly as the number of such data partitions increase to millions – as in typical P2P systems.

2.2.2 Computation in large distributed (P2P) systems:

Computation for P2P networks span three main areas: (1) best effort heuristics, (2) gossip based computations, (3) broadcast-based systems and (4) local algorithms. For a detailed survey interested readers can refer to [11].

Algorithms using best effort heuristics have been developed for large distributed systems. The P2P k-Means algorithm by Banyopadhyay et al. [2] is one such example. Typically for such algorithms, a peer collects some samples from its own data and its neighbors and builds a model on this sample. The samples are generally collected using some variations of random walk-based techniques. These algorithms can be classified as probabilistic approximate algorithms since the results are bounded only on average. A different class is the set of deterministic approximate algorithms such as the inferencing problem in sensor networks using variational approximation technique proposed by Mukherjee et al. [23].

Gossip algorithms rely on the properties of random samples to provide probabilistic guarantees on the accuracy of the results. Researchers have developed different approaches for performing basic operations (*e.g.* average, sum, max, random sampling) on P2P networks using gossip techniques. Kempe *et al.* [18] and Boyd *et al.* [5] present such primitives. In gossip protocols, a peer exchanges data or statistics with a random peer. However, they can still be quite costly – requiring hundreds of messages per peer for the computation of just one statistic. Another closely related technique is to use deterministic gossip or flooding. In flooding, every peer floods/broadcasts the data and therefore, eventually the data/statistic is propagated through the entire network. Here again the major drawback is scalability and the answer to dynamic data.

Communication-efficient broadcast-based algorithms have been also developed for large clusters such as the one developed by Sharfman et al. [26]. Since these algorithms rely on broadcasts as their mode of communication, the cost quickly increases with increasing system size.

Local algorithms are a good choice for data mining in P2P networks since in a local algorithm, the result is generally computed by communicating with a handful of nearby neighbors. Local algorithms rely on data dependent conditions which we refer to as local rules, to stop propagating messages. This means that if the data distribution does not change, the communication overhead is very low. On the other hand, the local rules are violated when the distribution changes. While on one hand, local algorithms are highly efficient (and hence scalable), on the other hand they guarantee eventual convergence to the *exact* result (equal to that which would be computed given the entire data). This feature makes local algorithms exceptionally suitable for P2P networks as well as to wireless sensor networks.

Early work on local algorithms include the ones by Afek et al. [1] and Linial [20] in the context of graph theory. More recently, local algorithms have been developed for several data mining problems: association rule mining [29], facility location [19], L2 Thresholding [28], outliers detection [6], and meta-classification [22]. Finally, several efforts were made at the description of local algorithm complexity [3].

3 Notations and Problem Definition

3.1 Notations Let $V = \{P_1, \dots, P_n\}$ be a set of peers connected to one another via an underlying communication infrastructure such that the set of P_i's neighbors, N_i, is known to P_i. Additionally, for at a given time instance t, P_i is given a stream of data vectors in $\mathbb{R}^d$. The local data of peer P_i at time t is $S_{i,t} = \left[\left(\overrightarrow{x_1^{i,t}}, f(\overrightarrow{x_1^{i,t}})\right), \left(\overrightarrow{x_2^{i,t}}, f(\overrightarrow{x_2^{i,t}})\right), \dots, \left(\overrightarrow{x_s^{i,t}}, f(\overrightarrow{x_s^{i,t}})\right)\right]$, where each $\overrightarrow{x_j^{i,t}}$ is a (d-1)-dimensional data point $\left[x_{j1}^{i,t} x_{j2}^{i,t} \dots x_{j(d-1)}^{i,t}\right]$ and f is a function from $\mathbb{R}^{d-1} \to \mathbb{R}$. Every data point can be viewed as an input and output pair.

Peers communicate with one another by sending sets of input vectors (or statistics thereof, as defined below). Let $X_{i,j}$ denote the last set of vectors sent by peer P_i to P_j. Assuming reliable messaging, once a message is delivered both P_i and P_j know $X_{i,j}$ and $X_{j,i}$. Now we define four sets of vectors which are crucial to our algorithm.

DEFINITION 3.1. (*Knowledge*) *The* ***knowledge*** *of P_i is the union of $S_{i,t}$ with $X_{j,i}$ for all $P_j \in N_i$ and is denoted by* $\mathcal{K}_i = S_{i,t} \cup \bigcup_{P_j \in N_i} X_{j,i}$.

DEFINITION 3.2. (*Agreement*) *The* ***agreement*** *of P_i and any of its neighbors P_j is $\mathcal{A}_{i,j} = X_{i,j} \cup X_{j,i}$.*

DEFINITION 3.3. (*Withheld knowledge*) *The subtraction of the agreement from the knowledge is the* ***withheld knowledge*** *of P_i with respect to a neighbor P_j* i.e. $\mathcal{W}_{i,j} = \mathcal{K}_i \setminus \mathcal{A}_{i,j}$.

DEFINITION 3.4. (*Global knowledge*) *The* ***global knowledge*** *is the set of all inputs at time t and is denoted by* $\mathcal{G}_t = \bigcup_{i=1,\dots,n} S_{i,t}$.

Since these vector sets can be arbitrarily large, we define two sufficient statistics on these sets which the peers will use to communicate: (1) the *average vector* which is the average of all the vectors in the respective sets (e.g. for $S_{i,t}$ it is $\overrightarrow{S_{i,t}} = \frac{1}{|S_{i,t}|} \sum_{\vec{x} \in S_{i,t}} \vec{x}$ similarly $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$, $\overrightarrow{\mathcal{W}_{i,j}}$, $\overrightarrow{X_{i,j}}$, $\overrightarrow{X_{j,i}}$ and $\overrightarrow{\mathcal{G}_t}$), and (2) the *sizes* of the sets denoted by $|S_{i,t}|$, $|X_{i,j}|$, $|X_{j,i}|$, $|\mathcal{K}_i|$, $|\mathcal{A}_{i,j}|$, $|\mathcal{W}_{i,j}|$, and $|\mathcal{G}_t|$. If we assume that communication always takes place in an overlay tree topology, we can write the following expressions for the sizes of the sets:

1. $|\mathcal{A}_{i,j}| = |X_{i,j}| + |X_{j,i}|$
2. $|\mathcal{K}_i| = |S_{i,t}| + \sum_{P_j \in N_i} |X_{j,i}|$, and
3. $|\mathcal{W}_{i,j}| = |\mathcal{K}_i| - |\mathcal{A}_{i,j}|$.

Similarly for the average vectors we can write,

1. $\overrightarrow{\mathcal{A}_{i,j}} = \frac{|X_{i,j}|}{|\mathcal{A}_{i,j}|}\overrightarrow{X_{i,j}} + \frac{|X_{j,i}|}{|\mathcal{A}_{i,j}|}\overrightarrow{X_{j,i}}$
2. $\overrightarrow{\mathcal{K}_i} = \frac{|S_{i,t}|}{|\mathcal{K}_i|}\overrightarrow{S_{i,t}} + \sum_{P_j \in N_i} \frac{|X_{j,i}|}{|\mathcal{K}_i|}\overrightarrow{X_{j,i}}$
3. $\overrightarrow{\mathcal{W}_{i,j}} = \frac{|\mathcal{K}_i|}{|\mathcal{W}_{i,j}|}\overrightarrow{\mathcal{K}_i} - \frac{|\mathcal{A}_{i,j}|}{|\mathcal{W}_{i,j}|}\overrightarrow{\mathcal{A}_{i,j}}$

Note that, for any peer, any of these vectors can be computed based solely on its local data and what it gets from its immediate neighbors. This, makes the communication of the algorithm localized. We are interested in computing a regression model defined on $\mathcal{G}_t$.

Next we formally state the problem definition.

3.2 Problem Definition In MR, the task is to learn a function $\widehat{f}(\overrightarrow{x})$ which "best" approximates $f(\overrightarrow{x})$ according to some measure such as least square. Now depending on the representation chosen for $\widehat{f}(\overrightarrow{x})$, various types of regression models (linear or nonlinear) can be developed. We leave this type specification as part of the problem statement for our algorithm, rather than an assumption.

In MR, for each data point $\overrightarrow{x}$, the error between $\widehat{f}(\overrightarrow{x})$ and $f(\overrightarrow{x})$ can be computed as $\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2$. In our scenario, since this error value is distributed across the peers, a good estimate of the global error is $Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2$. There exist several methods for measuring how suitable a model is for the data under consideration. We have used the L2-norm distance between the current network data and the model as a quality metric for the model built. Given a dynamic environment, our goal is to maintain a $\widehat{f}(\overrightarrow{x})$ at each peer at any time which best approximates $f(\overrightarrow{x})$.

Problem 1. [**MR Problem**] Given a time varying dataset $S_{i,t}$, a user-defined threshold ϵ and $\widehat{f}(\overrightarrow{x})$ to all the peers, the MR problem is to maintain a $\widehat{f}(\overrightarrow{x})$ at each peer such that, at any time t, $\left\|Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2\right\| < \epsilon$.

For ease of explanation, we decompose this task into two subtasks. First, given a representation of $\widehat{f}(\overrightarrow{x})$ to all the peers, we want to raise an alarm whenever $\left\|Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2\right\| > \epsilon$, where ϵ is a user-defined threshold. This is the *model monitoring problem*. Secondly, if $\widehat{f}(\overrightarrow{x})$ no longer represents $f(\overrightarrow{x})$, we sample from the network (or even better do an in-network aggregation) to find an updated $\widehat{f}(\overrightarrow{x})$. This is the *model computation problem*. Mathematically, the subproblems can be formalized as follows.

Problem 2.[**Monitoring Problem**] Given $S_{i,t}$, and $\widehat{f}(\overrightarrow{x})$ to all the peers, the monitoring problem is to output 0 if $\left\|Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2\right\| < \epsilon$, and 1 otherwise, at any time t.

Problem 3.[**Computation Problem**] The model computation problem is to find a new $\widehat{f}(\overrightarrow{x})$ based on a sample of the data collected from the network.

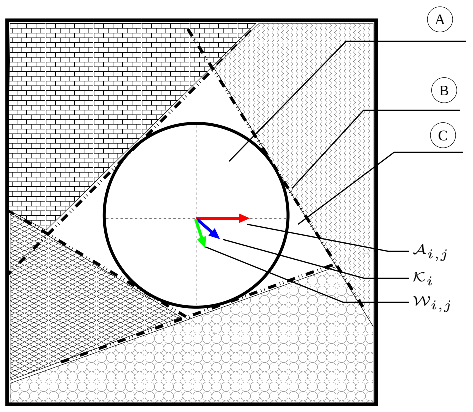


Figure 1: (A) the area inside an ϵ circle (B) A tangent defining a half-space (C) The areas between the circle and the union of half-spaces are the tie areas. Also shown are knowledge, agreement and withheld knowledge in red, blue and green respectively.

The global average error between the true model and the computed one is a point in $\mathbb{R}$. In order to use L2 norm thresholding as the metric for tracking this average error, we transform this 1-D problem to a 2-D problem by defining a vector in $\mathbb{R}^2$ with the first component set to the average error for the peer and the second component set to 0. Therefore, determining if the average error is less than ϵ is equivalent to finding if the average error vector lies inside a circle of radius ϵ. Our theorem (presented in Section 4) requires us to split the entire space in $\mathbb{R}^2$ (henceforth called the *domain*) into non-overlapping convex regions such that $\left\|\overrightarrow{\mathcal{G}_t}\right\|$ has the same value inside each of these convex regions. We denote the set of all such convex regions by C_δ. For the regression monitoring algorithm, it consists of the following regions - the inside of the ϵ-circle (R_c) and the half-spaces (R_{h_1}, R_{h_2}, ...) defined by each of these tangent lines (and hence unit vectors). The angle between each of these unit vectors is $\frac{2\pi}{d}$, where d is the number of tangent lines chosen. Note that the region in which the output is 0 is R_c, which itself is convex. The output is 1 inside any of these half-planes. The areas uncovered by C_δ denote the *tie* regions.

For simplicity, we assume that the network topology forms a tree. Note that, as shown in [4], such a tree can be efficiently constructed and maintained using variations of Bellman-Ford algorithms [13][17].

3.3 Local Algorithm Local algorithms, as defined by Das et al. [10], are parameterized by two quantities – (1) α – which is the number of neighbors a peer contacts in order to find answer to a query and (2) γ – which is the total size of the response which a peer receives as the answer to all the queries executed throughout the lifetime of the algorithm. α can be a constant or a function parameterized by the size of the network while γ can be parameterized by both the size of the data of a peer and the size of the network.

The idea is to design algorithms that offers bounded total communication cost per node and also spatially localized communication among the neighbors. We call such an (α, γ)-local algorithm ***efficient*** if both α and γ are either small constants or some slow growing functions (sublinear) with respect to its parameters. We prove that the regression monitoring algorithm is (α, γ)-local in Section 4.

4 Step 1: Monitoring Regression Model

In MR monitoring problem, each peer is given a dataset $S_{i,t}$ and an estimate $\widehat{f}(\overrightarrow{x})$. Our goal is to monitor $\left|\left|Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2\right|\right|$.

The problem now is to monitor the L2 norm of the average error. As stated before, although the error is a single number instead of a vector, we can easily define a vector in $\mathbb{R}^2$ with the second component 0. We present here a local algorithm which monitors the regression coefficients using the L2 norm of the average error vector. A preliminary L2 norm monitoring algorithm was also dealt in our previous paper [28] in the context of k-means clustering. The algorithm outputs 0 if $\left|\left|\overrightarrow{\mathcal{G}_t}\right|\right| < \epsilon$ and 1 otherwise. The algorithm presented in [28] is prone to noise in the data since it communicates all the data for every data change. In this paper, we take care of that problem by applying a different condition for sending messages and ensuring that all data is not sent whenever a peer communicates. Rather, we keep some data (in the form of witheld knowledge) so that if the data changes later, the change is less noisy. Moreover we tune the algorithm to make it suitable for regression. Thus, we feel that this is a significant contribution in its own right.

The regression monitoring algorithm guarantees eventual correctness, which means that once computation terminates, each peer computes the correct result as compared to a centralized setting. In a termination state, no messages traverse the network, and hence a peer can decide solely based on $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$, and $\overrightarrow{\mathcal{W}_{i,j}}$, if $\overrightarrow{\mathcal{G}_t}$ resides either inside the ϵ circle or one of the half-spaces defined by the tangent lines. In either case, as stated by the Theorem below, the peer achieves a termination state.

THEOREM 4.1. **[Convex Stopping Rule]** *Let $P_1, \ldots, P_n$ be a set of peers connected to each other over a spanning tree $G(V, E)$. Let $\mathcal{G}_t$, $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, and $\mathcal{W}_{i,j}$ be as defined in the previous section. Let C_δ denote the set of convex regions i.e. the inside of the ϵ circle and the non-overlapping half-spaces. Further, let $R \in C_\delta$ be any such convex region in C_δ. If at time t no messages traverse the network, and for each P_i, $\overrightarrow{\mathcal{K}_i} \in R$ and for every $P_j \in N_i$, $\overrightarrow{\mathcal{A}_{i,j}} \in R$ and either $\overrightarrow{\mathcal{W}_{i,j}} \in R$ or $\mathcal{W}_{i,j} = \emptyset$, then $\overrightarrow{\mathcal{G}_t} \in R$.*

Proof (SKETCH): We omit the formal proof here due to shortage of space. Simply speaking, the theorem can be proved by taking any two arbitrary peers and exchanging all of their withheld knowledge. We call this as the unification step. After unifying all the peers it can be shown that $\overrightarrow{\mathcal{G}}_t \in R$. ∎

The significance of Theorem 4.1 is that under the condition described P_i can stop sending messages to its neighbors and output $\left|\left|\overrightarrow{\mathcal{K}_i}\right|\right|$. The idea is to ensure that $\overrightarrow{\mathcal{K}_i}$ and $\overrightarrow{\mathcal{G}_t}$ finally reside in the same convex region. If the result of the theorem holds for every peer, then Theorem 4.1 guarantees this is the correct solution; else, there must either be a message in transit, or some peer P_k for whom the condition does not hold. Then either P_k will send a message which will change its output or the message will be received, leading to a change in $\overrightarrow{\mathcal{K}_k}$ eventually. Thus eventual correctness is guaranteed.

In order for the monitoring algorithm to track the global average error, we need to specify the input to the algorithm. For the input, every data point $[\overrightarrow{x}, f(\overrightarrow{x})] \in S_{i,t}$ is mapped to $\left(\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2, 0\right)$. Reusing the notation, henceforth we will refer to this as $S_{i,t}$.

```
ALGORITHM 4.1. Monitoring Regression Model
Input: $\epsilon$, $C_\delta$, $S_{i,t}$, $N_i$ and $L$.
Output: 0 if $\left|\left|\overrightarrow{\mathcal{K}_i}\right|\right| < \epsilon$, 1 otherwise
Initialization: Initialize vectors;
if MessageRecvdFrom $\left(P_j, \overrightarrow{X}, |X|\right)$ then
    $\overrightarrow{X_{j,i}} \leftarrow \overrightarrow{X}$ and $|X_{j,i}| \leftarrow |X|$
    Update vectors
end if
if $S_{i,t}$, $N_i$ or $\mathcal{K}_i$ changes then
    for all Neighbors $P_j$ do
        if $LastMsgSent > L$ time units ago then
            if $R = \emptyset$ then
                Set $\overrightarrow{X_{i,j}} \leftarrow \frac{|\mathcal{K}_i|\overrightarrow{\mathcal{K}_i} - |X_{j,i}|\overrightarrow{X_{j,i}}}{|\mathcal{K}_i| - |X_{j,i}|}$ {/*Tie Region*/}
            Set $|X_{i,j}| \leftarrow |\mathcal{K}_i| - |X_{j,i}|$
          end if
          if $\overrightarrow{\mathcal{A}_{i,j}} \notin R$ or $\overrightarrow{\mathcal{W}_{i,j}} \notin R$ then
            Set $\overrightarrow{X_{i,j}}$ and $|X_{i,j}|$ such that $\overrightarrow{\mathcal{A}_{i,j}}$ and
            $\overrightarrow{\mathcal{W}_{i,j}} \in R$ {/*Theorem Condition*/}
          end if
          MessageSentTo$\left(P_j, \overrightarrow{X_{i,j}}, |X_{i,j}|\right)$
          LastMsgSent ← CurrentTime
          Update all vectors
        else
          Wait $L$ time units and then check again
        end if
      end for
    end if
```

Algorithm 4.1 presents the pseudo-code. The inputs to the algorithm are $S_{i,t}$, N_i, ϵ and C_δ and L. Each peer initializes its local vectors $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{W}_{i,j}}$. A peer may need to send a message if its local data changes, if it receives a message or if the set of neighbors change. In any of these cases, the peer checks if the condition of the theorem holds. First peer P_i finds the region $R \in C_\delta$ such that $\overrightarrow{\mathcal{K}_i} \in R$ ($R = \emptyset$ otherwise). If $R = \emptyset$, then $\overrightarrow{\mathcal{K}_i}$ is in a tie region and hence P_i has to send all its data. If, for all $P_j \in N_i$, both $\overrightarrow{\mathcal{A}_{i,j}} \in R$ and $\overrightarrow{\mathcal{W}_{i,j}} \in R$, P_i does nothing; else it needs to set $\overrightarrow{X_{i,j}}$ and $|\overrightarrow{X_{i,j}}|$ and send those, such that after the message is sent, the condition of the theorem holds for this peer. As we already pointed out that if a peer communicates all of its data, then if the data changes again later, the change is far more noisy than the original data. So we always set $\overrightarrow{X_{i,j}}$ and $|X_{i,j}|$ such that some data is retained while still maintaining the conditions of the theorem. We do this by checking with an exponentially decreasing set of values of $|\mathcal{W}_{i,j}|$ until either all $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{W}_{i,j}} \in R$, or $|\mathcal{W}_{i,j}|$=0, in which case we have to send everything. Note that other than these two cases, a peer need not send a message since the theorem guarantees eventual correctness. Similarly, whenever it receives a message ($\overrightarrow{X}$ and $|\overrightarrow{X}|$), it sets $\overrightarrow{X_{j,i}} \leftarrow \overrightarrow{X}$ and $|\overrightarrow{X_{j,i}}| \leftarrow |\overrightarrow{X}|$. This may trigger another round of communication since its $\overrightarrow{\mathcal{K}_i}$ can now change.

To prevent message explosion, in our event-based system we employ a "leaky bucket" mechanism which ensures that no two messages are sent in a period shorter than a constant L. Note that this mechanism does not enforce synchronization or affect correctness; at most it might delay convergence. This technique has been used elsewhere also [28].

Locality: Determining the communication complexity of local algorithms in dynamic environments is still an open research issue. Researchers have looked into the problem using the idea of veracity radius [3] for simple aggregation problems. Considering the (α, γ) framework, there always exist problem instances for which any eventually correct algorithm e.g. [22][29][28] (including the one described here) will have worst case γ = O(size of network). For this algorithm, this can happen when the average error vector lies in the tie region. While O(size of network) is the upper bound on the communication complexity, more accurate bounds on γ can be developed by identifying the specific problems and input instances. We feel that there is an intrinsic relation between γ and ϵ, though it needs to be investigated more and we plan to report those in a future article. On the other hand, γ = O(1) whenever the system moves from one termination state to the other and the data change is not significant to make all the peers communicate globally. We present this formally in Lemma 4.1.

LEMMA 4.1. *[**Locality**] The regression monitoring algorithm is $(O(1), \gamma)$-local where γ can vary between O(1) and O(size of network).*

Proof (SKETCH): The algorithm requires each peer to communicate with its immediate neighbors only. Hence, $\alpha = O(1)$, independent of the size of the network.

Now consider the situation where the system is in a termination state where $\overrightarrow{S_{i,t}} \in R_c$ for each peer (hence $\overrightarrow{\mathcal{G}_t} \in R_c$) and then the data changes. We seek to count the number of messages which take the system to another termination state. If the data change is such that for each peer still $\overrightarrow{S_{i,t}} \in R_c$, any given peer will send at most one message to each of its neighbors. One the other hand, if the new $\overrightarrow{S_{i,t}} \notin R_c$, but rather in a tie region, any given peer may need to send as many messages which will convince it that indeed $\overrightarrow{\mathcal{G}_t} \notin R_c$. In the worst case, each peer may need to get data from all the nodes in the network in order to get convinced of this change. In this case, γ = O(size of the network). Therefore γ can vary between O(1) and O(size of network) ∎

The high scalability of the local algorithms is due to the fact that α and γ are constants and is independent of the network size for many interesting problem instances as corroborated by our extensive experimental results.

5 Step 2: Computing Regression Model

The regression monitoring algorithm presented in the earlier section can be viewed as a flag which is raised by a peer whenever $\left\|Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2\right\| > \epsilon$. In this section we discuss how the peers collaborate to find a new $\widehat{f}(\overrightarrow{x})$ using a convergecast-broadcast technique.

The basic idea is to use the *convergecast* phase to sample data from the network to a central post and compute, based on this sample, a new $\widehat{f}(\overrightarrow{x})$. The *broadcast* phase distributes this $\widehat{f}(\overrightarrow{x})$ to the network. The monitoring algorithm now monitors the quality of the result. The efficiency and

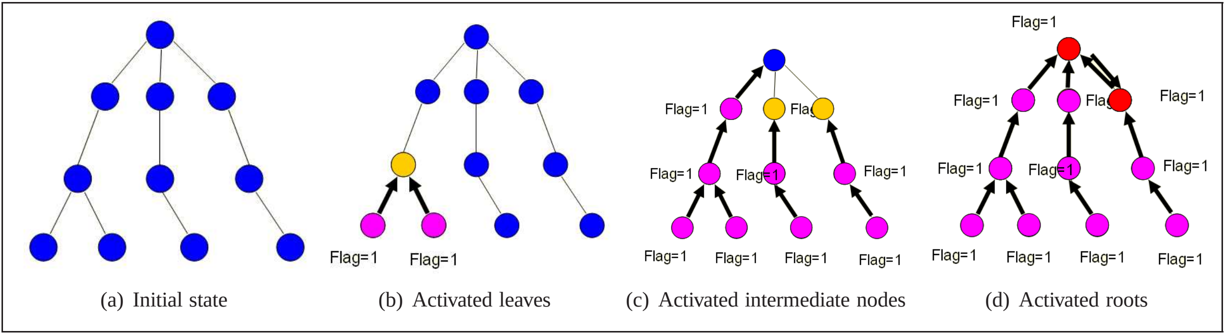


Figure 2: Convergecast and broadcast through the different steps. The blue circles represent states when the peers do not raise a flag. The purple circles represent the state when a peer has raised and alert (flag=1) and sent its data to its parent. As soon as the peer gets data from all but one neighbor it turns yellow. Finally, the roots are denoted by red.

correctness of the monitoring algorithm allows a very simple sampling technique to be used – if an ill-fit model is built at all, it will soon be detected by the local algorithm resulting in another round of convergecast in the worst case. Another point to note is that, in our convergecast-broadcast process, we do not specify the root of the convergecast tree. Rather we let the network structure (edge delays and data skewness) decide it. This is significant since it ensures (1) decentralized control, (2) load balancing, and (3) robustness against a single point of failure.

In the convergecast phase there are two main parameters. Each peer maintains a user selected alert mitigation constant, τ and the sample size. An alert should be stable for a given period of time τ before the peer can send its data, thereby preventing a possibly false alarm from propagating. In order to do this, the peer relies on the underlying monitoring algorithm. If the monitoring algorithm raises a flag, the peer notes the time, and sets a timer to τ time units. If the timer expires, or a data message is received from one of its neighbors, P_i first checks if there is an existing alert and if it has been recorded τ or more time units ago. If so, it counts the number of neighbors from which it has received data messages. Once it receives data messages from all of its neighbors, the peer computes a new regression model $\widehat{f}(\overrightarrow{x})$ based on the sample it received and sends it to itself. It then moves to the broadcast phase and sends $\widehat{f}(\overrightarrow{x})$ to all its neighbors. On the other hand, if it has received data messages from all but one of the neighbors then it takes a uniform sample (of user-defined size) from its own data and the data it has received from its neighbors. It then forwards the sample to the peer from which it has not received data and marks its state as broadcast. The peer does nothing if it has not received data from two or more neighbors. Note that, at each peer, the sampling technique is such that, each data point gets an equal chance of being included in the sampled data set. We do this by properly weighing every data point by size of the subtree from which the sample was received.

The broadcast phase is fairly straightforward. Every peer which receives the new $\widehat{f}(\overrightarrow{x})$, restarts a new regression monitoring algorithm with this new $\widehat{f}(\overrightarrow{x})$. It then, sends the new $\widehat{f}(\overrightarrow{x})$ to its other neighbors and changes the status to convergecast. There could be one situation in which a peer receives a new $\widehat{f}(\overrightarrow{x})$ when it is already in the broadcast phase. This is when two neighbor peers concurrently become roots for the convergecast tree. To break this tie, we select the $\widehat{f}(\overrightarrow{x})$ to propagate the root of which has a higher id. Figure 2 shows a snap-shot of the convergecast broadcast steps as it progresses up the communication tree.

The pseudo-code is presented in Algorithm 5.1.

```
ALGORITHM 5.1. P2P Regression Algorithm
Input: $\epsilon$, $C_\delta$, $S_{i,t}$, $N_i$, $L$, $\widehat{f}$ and $\tau$
Output: $\widehat{f}$ such that $\left\|Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2\right\| < \epsilon$
Initialization: Initialize vectors;
MsgType = MessageRecvdFrom($P_j$)
if $MsgType = Monitoring\_Msg$ then
    Pass Message to Monitoring Algorithm
end if
if $MsgType = New\_Model\_Msg$ {/*Broadcast*/}
then
    Update $\widehat{f}$
    Forward new $\widehat{f}$ to all neighbors
    Datasent=false
    Restart Monitoring Algorithm with new $\widehat{f}$
end if
if $MsgType = Dataset\_Msg$ {/*Convergecast*/} then
    if Received from all but one neighbor then
        $flag$=Output Monitoring Algorithm()
        if $Datasent = false$ and $flag = 1$ then
            if DataAlert stable for $\tau$ time then
                $D$=Sample($S_{i,t}$, $Recvd\_Dataset$)
```

```
          $Datasent$=true;
          Send $D$ to remaining neighbor
        else
          *DataAlert*=CurrentTime
        end if
      end if
    end if
    if *Received from all neighbors* then
      $D$=Sample($S_{i,t}$, $Recvd\_Dataset$)
      $\widehat{f}$=Regression($D$)
      Forward new $\widehat{f}$ to all neighbors
      *Datasent*=false
      Restart Monitoring Algorithm with new $\widehat{f}$;
    end if
  end if
  if $S_{i,t}$, $N_i$ or $\mathcal{K}_i$ changes then
    Run Monitoring Algorithm
    *flag*=Output_Monitoring_Algorithm()
    if *flag*=1 and $P_j$=IsLeaf() then
      Execute the same conditions as
      $MsgType = Dataset\_Msg$
    end if
  end if
```

Special case : Linear Regression In many cases, sampling from the network is communication intensive. We can find the coefficients using an in-network aggregation if we choose to monitor a widely used regression model *viz.* linear regression (linear with respect to the parameters or the unknown weights).

Since the dataset of each peer consists of s d-dimensional tuples $[\overrightarrow{x}, f(\overrightarrow{x})]$ and $\overrightarrow{x} = [x_1 x_2 \ldots x_{d-1}]$, we want to fit a $d-1$ degree polynomial since the last attribute corresponds to the output: $\widehat{f}(\overrightarrow{x}) = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_{d-1}x_{d-1}$, where a_i's are the coefficients that need to be estimated from the global dataset $\mathcal{G}_t$. We drop the cross terms involving x_i and x_j for simplicity. Using least square technique and after simplification we get,

$$\begin{pmatrix} |\mathcal{G}_t| & \sum_{j=1}^{|\mathcal{G}_t|} x_1^j & \cdots & \sum_{j=1}^{|\mathcal{G}_t|} x_{d-1}^j \\ \sum_{j=1}^{|\mathcal{G}_t|} x_1^j & \sum_{j=1}^{|\mathcal{G}_t|} (x_1^j)^2 & \cdots & \sum_{j=1}^{|\mathcal{G}_t|} x_1^j * x_{d-1}^j \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{j=1}^{|\mathcal{G}_t|} x_{d-1}^j & \sum_{j=1}^{|\mathcal{G}_t|} x_{d-1}^j * x_1^j & \cdots & \sum_{j=1}^{|\mathcal{G}_t|} (x_{d-1}^j)^2 \end{pmatrix}$$

$$\times \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{|\mathcal{G}_t|} f(\overrightarrow{x}^j) \\ \sum_{j=1}^{|\mathcal{G}_t|} f(\overrightarrow{x}^j) x_1^j \\ \vdots \\ \sum_{j=1}^{|\mathcal{G}_t|} f(\overrightarrow{x}^j) x_{d-1}^j \end{pmatrix} \Rightarrow \mathbf{Xa} = \mathbf{Y}$$

where x_i^j is the value of the i^{th} attribute in $\mathcal{G}_t$ for the j^{th} row and $f(\overrightarrow{x}^j)$ is the corresponding output. Therefore for computing the matrix (or more appropriately vector) **a**, we need to evaluate the matrices **X** and **Y**. This can be done in a communication efficient manner by noticing that the entries of these matrices are simply sums. Hence, instead of sending the raw data in the convergecast round, peer P_i can forward a locally computed matrix $\mathbf{X}_i$ and $\mathbf{Y}_i$. Peer P_j, on receiving this, can forward a new matrix $\mathbf{X}_j$ and $\mathbf{Y}_j$ by aggregating, in a component-wise fashion, its local matrix and the received ones. Note that the avoidance of the sampling technique ensures that the result is exactly the same compared to a centralized setting. Moreover, the dimensionality of the matrices $\mathbf{X}_i$ and $\mathbf{Y}_i$ is $d.d + d.1 = O(d^2)$. This shows that the communication complexity is only dependent on the degree of the polynomial or the number of attributes. Since, in most cases, the number of attributes is much small compared to the sample size required in the convergecast round, there can be significant savings in terms of communication.

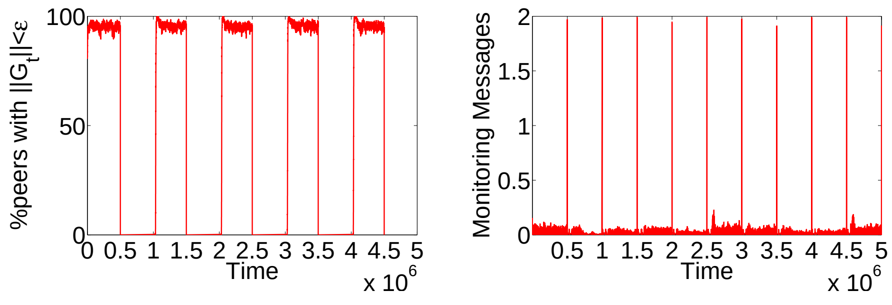
6 Experimental Results

In this section we discuss the experimental setup and analyze the performance of the P2P regression algorithm.

6.1 Experimental Setup We have implemented our algorithms in the Distributed Data Mining Toolkit (DDMT) [12] developed by the DIADIC research lab at UMBC. We use topological information generated by the *Barabasi Albert (BA)* model in BRITE [8] since it is often considered a reasonable model for the internet. BA also defines delay for network edges, which is the basis for our time measurement[1]. On top of the network generated by BRITE, we overlay a communication tree.

6.2 Data Generation The input data of a peer is a vector $(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$, where the first $d-1$ dimensions correspond to the input variables and the last dimension corresponds to the output. We have conducted experiments on both linear and non-linear regression models. For the linear model, the output is generated according to $x_d = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_{d-1}x_{d-1}$. We have used two functions for the non-linear model: (1) $x_3 = a_0 + a_1 a_2 x_1 + a_0 a_1 x_2$ (multiplicative) and (2) $x_3 = a_0 * sin(a_1 + a_2 x_1) + a_1 * sin(a_2 + a_0 x_2)$ (sinusoidal). Every time a simulated peer needs an additional data point, it chooses the values of $x_1, x_2, \ldots x_{d-1}$, each independently in the range -100 to +100. Then it generates the value of the target variable x_d using any of the above functions and adds a uniform noise in the range 5 to 20% of the value of the target output. The regression weights $a_0, a_1, \ldots, a_{d-1}$'s are changed randomly at controlled intervals to create an epoch change.

[1]Wall time is meaningless when simulating thousands of computers on a single PC.



(a) Percentage of peers with $\|\mathcal{K}_i\| < \epsilon$ (b) Monitoring Cost

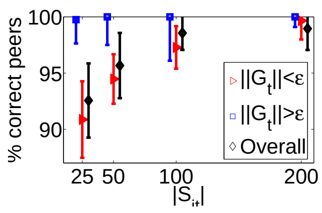
Figure 3: A typical experiment is run for 10 equal length epochs. Quality and overall cost are measured across the entire experiment – including transitional phases. The monitoring cost is measured on the last 80% of every epoch, in order to ignore transitional effects.

6.3 Measurement Metric In our experiments, the two most important parameters for measurement are the *quality* of the result and the *cost* of the algorithm. For the monitoring algorithm, quality is measured in terms of the percentage of peers which correctly compute an alert, *i.e.*, the number of peers which report that $\left\|\overrightarrow{\mathcal{K}_i}\right\| < \epsilon$ when $\left\|\overrightarrow{\mathcal{G}_t}\right\| < \epsilon$ and similarly $\left\|\overrightarrow{\mathcal{K}_i}\right\| > \epsilon$ when $\left\|\overrightarrow{\mathcal{G}_t}\right\| > \epsilon$. For the regression computation algorithm, quality is defined as the L2 norm distance between the solution of our algorithm and the actual regression weights. We refer to the cost of the algorithm as the number of *normalized messages* sent, which is the number of messages sent by each peer per unit of leaky bucket L. Hence, 0.1 normalized messages means that nine out of ten times the algorithm manages to avoid sending a message. We report both overall cost and the monitoring cost (stationary cost), which refers to the "wasted effort" of the algorithm. We also report, where appropriate, messages required for convergecast and broadcast of the model.

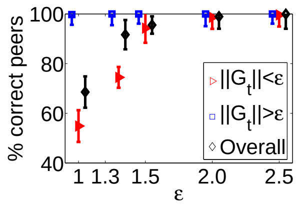
6.4 Typical Experiments A typical experiment is shown in Figure 3. In all the experiments, about 4% of the data of each peer is changed every 1000 simulator ticks. Moreover, after every 5×10^5 simulator ticks, the data distribution is changed. To start with, every peer is supplied the same regression coefficients as the coefficients of the data generator. Figure 3(a) shows that for the first epoch, the quality is very high (nearly 96%). After 5×10^5 simulator ticks, we change the weights of the generator without changing the coefficients given to each peer. Therefore the percentage of peers reporting $\left\|\overrightarrow{\mathcal{K}_i}\right\| < \epsilon$ drops to 0. For the cost, Figure 3(b) shows that the monitoring cost is low throughout the experiment if we ignore the transitional effects.

6.5 Results: Regression Monitoring There are four external parameters which can influence the behavior of the regression monitoring algorithm: size of local buffer $|S_{i,t}|$, the radius of the circle ϵ, size of the leaky bucket L and noise in the data. Apart from these there are also the system size (number of peers) and dimensionality of the multivariate regression problem which can affect performance. In this section we present the quality (inside *i.e.* $\left\|\overrightarrow{\mathcal{G}_t}\right\| < \epsilon$, outside *i.e.* $\left\|\overrightarrow{\mathcal{G}_t}\right\| > \epsilon$ and overall) and cost of the algorithm w.r.t. different parameters. Note that, unless otherwise stated, we have used the following default values for the different parameters: number of peers = 1000, $|S_{i,t}| = 50$, $\epsilon = 1.5$, $d = 10$ and $L = 500$ (where the average edge delay is about 1100 time units). As we have already stated, independent of the regression function chosen, the underlying monitoring problem is always in $\mathbb{R}^2$. The results reported in this section are with respect to linear model since it is the most widely used regression model. Results of monitoring more complex models are reported in the next section.

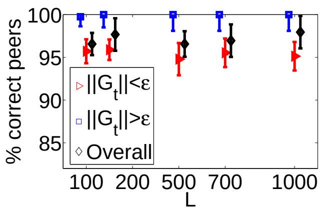
Figures 4(a) and 4(e) show the quality and cost of the algorithm as the size of local buffer is changed. As expected, the inside quality increases and the cost decreases as the size of buffer increases. The outside quality is very high throughout. This stems from the fact that, with the noise in the data, it is easy for a peer to get flipped over when it is checking for inside a circle. On the other hand, noise cannot change the belief of the peer when the average is outside. In the second set of experiments, we varied ϵ from 1.0 to 2.5 (Figure 4(b) and 4(f)). Here also, the quality increases as ϵ is increased. This is because with increasing ϵ, there is a bigger region in which to bound the global average. This is also reflected with decreasing number of messages. Note that, even for $\epsilon = 1.0$, the normalized messages are around 1.6, which is far less than the theoretical maximum of 2 (assuming two neighbors per peer). The third set of experiments analyzes the effect of leaky bucket L. As shown in Figure 4(c) quality does not depend on L, while Figure



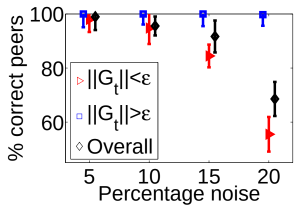
(a) Quality vs. $|S_{it}|$



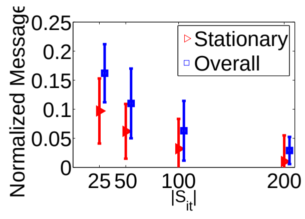
(b) Quality vs. ϵ



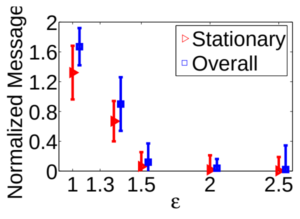
(c) Quality vs. L



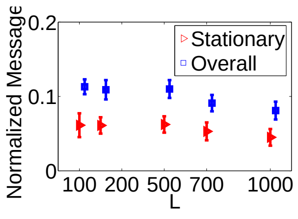
(d) Quality vs. Noise



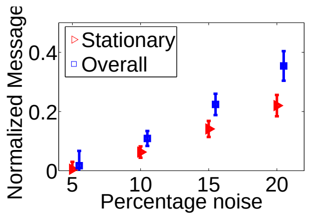
(e) Cost vs. $|S_{it}|$



(f) Cost vs. ϵ



(g) Cost vs. L



(h) Cost vs. Noise

Figure 4: Behavior of the monitoring algorithm w.r.t the different algorithm parameters.

4(g) shows that the cost decreases slowly with increasing L. Finally, Figures 4(d) and 4(h) depict the dependence of the noise on the monitoring algorithm. Quality degrades and cost increases with increasing noise. This is expected, since with increasing noise a peer is more prone to random effects. This effect can, however, be nullified by using a large buffer or bigger ϵ.

Our next experiment analyzes the scalability of the monitoring algorithm w.r.t the number of peers and dimension of the multivariate problem. As Figures 5(a) and 5(c) show, both the quality and cost of the algorithm converge to a constant as the number of peers increase. This is a typical behavior of local algorithms. For any peer, since the computation is dependent on the result from only a handful of its neighbors, the overall size of the network does not degrade the quality or cost. Similarly, Figures 5(b) and 5(d) show that the quality or the cost does not depend on the dimension of the multivariate problem either. This independence of the quality and cost can be explained by noting that the underlying monitoring problem is in $\mathbb{R}^2$. Therefore for a given problem, the system size or dimensionality of the problem has no effect on the quality or the cost.

Overall, the results show that the monitoring algorithm offers extremely good quality, incurs low monitoring cost and has high scalability.

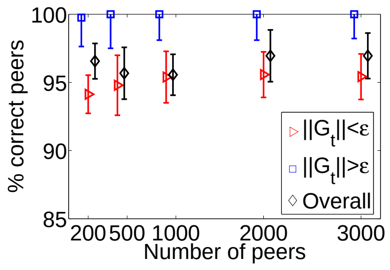
6.6 Results: Regression Models Our next set of experiments measure the quality of the regression model computed by our algorithm against a centralized algorithm having access to the entire data. There are two important parameters to be considered here – (1) the alert mitigation constant (τ) and (2) the sample size (for non-linear regression). For computing the non-linear regression coefficients, we have implemented the Nelder-Mead simplex method [24].

We have conducted experiments on three datasets. Each quality graph in Figure 6 presents two sets of error bars. The square markers show the L2 norm distance between the distributed coefficients and the actual ones. Also shown in each figure is the L2 norm distance between the coefficients found by a centralized algorithm and the actual ones (diamond markers). The first pair of figures, Figures 6(a) and 6(d) show the results of computing a linear regression model. Our aim is to measure the effect of variation of alert mitigation period τ on quality and cost. As shown in Figure 6(a), the quality of our algorithm deteriorates as τ increases. This is because, on increasing τ, a peer builds a model later and therefore is inaccurate for a longer intermediate period. Figure 6(d) shows that the number of data collection rounds (dot markers) decrease from four times to twice per epoch. This results from a decrease in the number of false alerts. Also shown are monitoring messages (green squares).

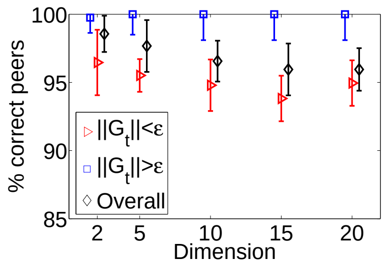
Figures 6(b) and 6(e) analyzes the quality of our algorithm while computing a non-linear multiplicative regression model *viz.* $x_3 = a_0 + a_1 a_2 x_1 + a_0 a_1 x_2$. Figure 6(b) presents the quality as other parameter *viz.* sampling size is varied. As expected, the results from the distributed and centralized computations converge with increasing sample size. Also the number of data collection rounds as depicted in Figure 6(e) decrease as sample size is increased.

The third pair of figures, Figures 6(c) and 6(f) show the same results for a sinusoidal model : $x_3 = a_0 * sin(a_1 + a_2 x_1) + a_1 * sin(a_2 + a_0 x_2)$. Here also the quality becomes better and the cost decreases as the sample size is increased.

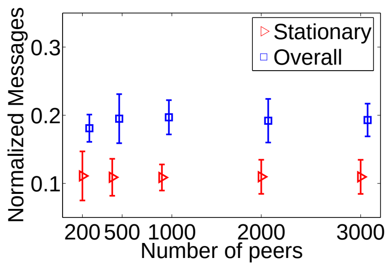
To sum everything up, the regression computation algorithm offers excellent accuracy and low monitoring cost. Also, the number of convergecast-broadcast rounds is also



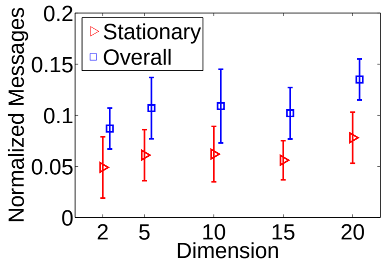
(a) Quality vs. number of peers



(b) Quality vs. dimension

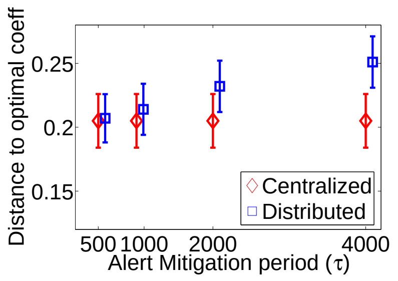


(c) Cost vs. number of peers

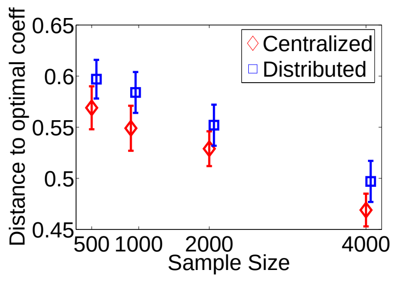


(d) Cost vs. dimension

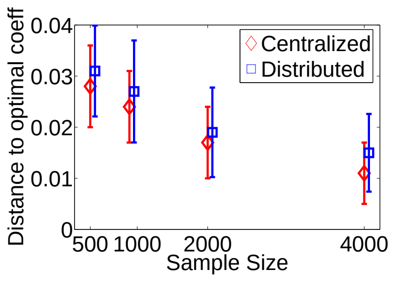
Figure 5: Scalability with respect to both number of peers and dimension of the multivariate problem.



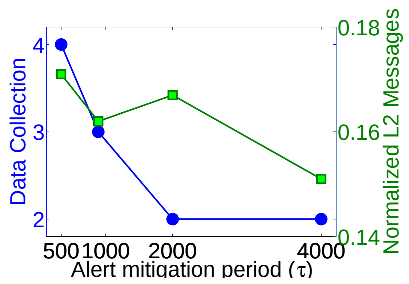
(a) Quality for linear model



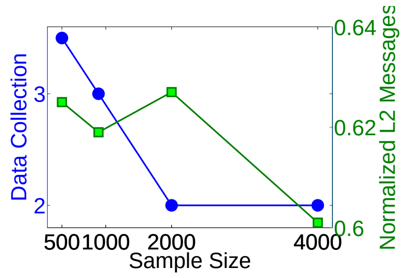
(b) Quality for multiplicative model



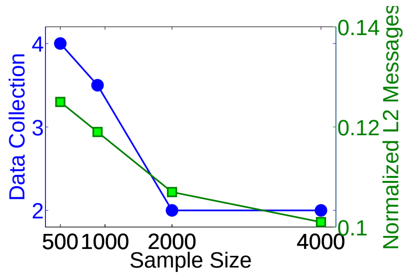
(c) Quality for sinusoidal model



(d) Cost for linear model



(e) Cost for multiplicative model



(f) Cost for sinusoidal model

Figure 6: Quality and cost of computing regression coefficients for both linear and non-linear models.

two times per epoch on an average. We have tested our algorithm on several regression functions and the results are highly satisfactory.

7 Conclusions and Future Work

To the best of the authors' knowledge this is one of the first attempts on developing completely local and asynchronous regression algorithm for P2P systems which maintains the same regression models given all the data to all the peers. The algorithm is suitable for scenarios in which the data is distributed across a large P2P network as it seamlessly handles data changes and node failures. We have performed dynamic experiments with random epoch changes which showed that the algorithm is accurate, efficient and highly scalable. Such algorithms are needed for next generation P2P applications such as P2P bioinformatics, P2P web mining and P2P astronomy using National Virtual Observatories. As a next step, we plan to explore other methods of learning such as support vector machines and decision trees.

Acknowledgement

This work was supported by the United States National Science Foundation Grant IIS-0093353 and NASA Grant NNX07AV70G.

References

[1] Y. Afek, S. Kutten, and M. Yung. Local Detection For Global Self Stabilization. *In Theoretical Computer Science*, 186(1–2):199–230, October 1997.

[2] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta. Clustering Distributed Data Streams in Peer-to-Peer Environments. *Information Science*, 176(14):1952–1985, 2006.

[3] Y. Birk, I. Keidar, L. Liss, A. Schuster, and R. Wolff. Veracity Radius - Capturing the Locality of Distributed Computations. In *Proceedings of PODC '06*, pages 102–111, Colorado, Denver, 2006.

[4] Y. Birk, L. Liss, A. Schuster, and R. Wolff. A Local Algorithm for Ad Hoc Majority Voting via Charge Fusion. In *Proceedings of DISC'04*, pages 275–289, Amsterdam, Netherlands, 2004.

[5] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip Algorithms: Design, Analysis and Applications. In *Proceddings of INFOCOMM'05*, pages 1653–1664, Miami, March 2005.

[6] J. Branch, B. Szymanski, C. Gionnella, R. Wolff, and H. Kargupta. In-network outlier detection in wireless sensor networks. In *Proceedings of* ICDCS'06, Lisbon, Portugal, July 2006.

[7] Leo Breiman. Bagging predictors. *Machine Learning*, 2:123–140, 1996.

[8] BRITE. `http://www.cs.bu.edu/brite/`.

[9] Chinook. `http://smweb.bcgsc.bc.ca/chinook/index.html`.

[10] Kamalika Das, Kanishka Bhaduri, Kun Liu, and Hillol Kargupta. Distributed Identification of Top-l Inner Product Elements and its Application in a Peer-to-Peer Network. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Accepted (In press), 2007.

[11] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed Data Mining in Peer-to-Peer Networks. *IEEE Internet Computing*, 10(4):18–26, 2006.

[12] DDMT. `http://www.umbc.edu/ddm/Sftware/DDMT/`.

[13] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princton University Press, 1962.

[14] J. Friedman, T. Hastie, and R. Tibshirani. Additive Logistic Regression: a Statistical View of Boosting. Technical report, Dept. of Statistics, Stanford University, 1998.

[15] Carlos Guestrin, Peter Bodi, Romain Thibau, Mark Paski, and Samuel Madden. Distributed Regression: an Efficient Framework for Modeling Sensor Network Data. In *Proceedings of IPSN'04*, pages 1–10, Berkeley, California, 2004.

[16] D. E. Hershberger and H. Kargupta. Distributed Multivariate Regression Using Wavelet-based Collective Data Mining. *Journal of Parallel and Distributed Computing*, 61(3):372–400, 2001.

[17] J.M. Jaffe and F.H. Moss. A Responsive Distributed Routing Algorithm for Computer Networks. *IEEE Transactions on Communications*, 30(7):1758–1762, July 1982.

[18] D. Kempe, A. Dobra, and J. Gehrke. Computing Aggregate Information using Gossip. In *Proceedings of FOCS'03*, Cambridge, 2003.

[19] D. Krivitski, A. Schuster, and R. Wolff. Local Hill Climbing in Sensor Networks. In *Proceedings of DMSN workshop, in conjunction with SDM'05*, Newport Beach, California, April 2005.

[20] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal of Computing*, 21:193–201, 1992.

[21] K. Liu, K. Bhaduri, K. Das, and H. Kargupta. Client-side Web Mining for Community Formation in Peer-to-Peer Environments. *SIGKDD Explorations*, 8(2):11–20, December 2006.

[22] P. Luo, H. Xiong, K. Lü, and Z. Shi. Distributed Classification in Peer-to-Peer Networks. In *Proceedings of KDD'07*, pages 968–976, 2007.

[23] S. Mukherjee and H. Kargupta. Distributed Probabilistic Inferencing in Sensor Networks using Variational Approximation. *Journal of Parallel and Distributed Computing*, 68(1):78–92, 2008.

[24] J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

[25] J.B. Predd, S.R. Kulkarni, and H.V. Poor. Distributed Kernel Regression: An Algorithm for Training Collaboratively. *arXiv.cs.LG archive*, 2006.

[26] I. Sharfman, A. Schuster, and D. Keren. A Geometric Approach to Monitoring Threshold Functions over Distributed Data Streams. In *Proceedings of SIGMOD'06*, pages 301–312, Chicago, Illinois, June 2006.

[27] S. J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan. JAM: Java Agents for Meta-Learning over Distributed Databases. In *Proceedings of KDD'97*, pages 74–81, Newport Beach, California, 1997.

[28] R. Wolff, K. Bhaduri, and H. Kargupta. Local L2 Thresholding Based Data Mining in Peer-to-Peer Systems. In *Proceedings of SDM'06*, pages 430–441, Bethesda, MD, 2006.

[29] R. Wolff and A. Schuster. Association Rule Mining in Peer-to-Peer Systems. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 34(6):2426 – 2438, December 2004.

[30] Y. Xing, M. G. Madden, J. Duggan, and G. J. Lyons. Distributed Regression for Heterogeneous Data Sets. *Lecture Notes in Computer Science*, 2810:544–553, 2003.