TECHNICAL UNIVERSITY OF CRETE
ELECTRICAL AND COMPUTER ENGINEERING

DIPLOMA THESIS

Distributed Multivariate Regression via Functional Geometric Monitoring

Author:
Eftychia Seisaki

Thesis Committee:
Assoc. Prof. Vasilis Samoladas, (Supervisor)
Prof. Antonios Deligiannakis
Assoc. Prof. Michail Lagoudakis

A thesis presented in partial fulfillment of the requirements for the diploma in Electrical and Computer Engineering

December 22, 2021

Abstract

Distributed Multivariate Regression via Functional Geometric Monitoring

by Eftychia Seisaki

Multivariate linear regression is an important and massively used technique for modeling and predicting data behavior in many fields. In scenarios where the data evolves over time, it is essential to monitor the model in order to identify possible changes. This becomes more challenging, when the data is distributed at a number of different nodes and the regression model must be recomputed to avoid inaccuracy. In such dynamic settings, data centralization and periodic model recomputation can be wasteful. Therefore, the goal is to develop a technique which conserves a precise approximation of the model over the union of all nodes' data in a communication-efficient fashion.

We propose a monitoring algorithm for multivariate regression models of distributed data streams, based on the basic notions of Functional Geometric Monitoring (FGM), which guarantees a bounded model error and demands communication only when the estimated model has fairly departed from the current global. Our experimental results clearly demonstrate a reduction in communication cost while maintaining the desired model accuracy, compared to similar existing models.

Acknowledgements

First and foremost, I would like to sincerely thank my supervisor, professor Vasilis Samoladas, who provided me with feedback and support throughout the implementation of my diploma thesis. To continue with, I am specifically grateful to the rest of my thesis committee members, professors Antonios Deligiannakis and Michail Lagoudakis for their time to evaluate my work. Last but not least, I owe a thank to all my professors and the other members of this department for their contribution to my diploma acquisition and my forming as an engineer.

Contents

List of Figures

Chapter 1

Introduction

1.1 Description

Multivariate linear regression is a machine learning algorithm broadly used in order to recognize and predict data behavior in many scientific areas. Just to name a few, it is widely used for forecasting problems (the process of predicting new values from the old ones), for analysis of existing phenomena through discovered coefficients and as constructing parts in other algorithmic structures (e.g in sparse coding). However, in many of these cases, data usually changes or evolves over time and these alterations can make a previously computed model inaccurate. To avoid this situation, regression models should be updated to adapt to the new observations or to be recomputed at regular intervals of time. If the data is piecewise stationary, though, these solutions often have a significant overhead. This problem becomes even more tackling in a distributed environment. When data is distributed over an efficient number of nodes, we also have to manage the overhead of communication that is caused from the updates.

In order to compute regression models in distributed streaming settings, we need to find out not just how to do it efficiently, but also when. Recomputing the model after every new observation or even periodically is quite expensive as it involves unnecessary actions if the model changes sporadically. Consequently, we can deal with distributed linear regression in two different ways. First option is the efficient distributed computation of the model, an approach that has already been given a lot of attention [7, 3, 10]. The second approach, which is our area of focus, is to monitor the quality of a given model and recompute it only when necessary.

The monitoring approach checks on the input data and warns the system only if the pre estimated model is too different from the actual global, which would have been calculated from the current data. This monitoring procedure is explicitly demanding

in a distributed environment, since both global models are calculated from the union of data which are split through the nodes. As a result, a distributed monitoring algorithm must, also, handle the communication cost efficiently apart from monitoring the model. Thankfully, some algorithms in the area of data streams monitoring have recently been proposed, showing great experimental results in terms of communication efficiency. Sharfman, Schuster, and Keren [12], introduced the Geometric Monitoring (GM) method for non-linear functions over distributed streams by utilizing convex analysis theory. Vasilis Samoladas and Minos Garofalakis [9] proposed Functional Geometric Monitoring (FGM), a substantial theoretical and practical improvement of Geometric Monitoring, by monitoring geometric constraints on distributed succinct summaries of streams, such as histograms, sketches, or more generally, high dimensional vectors.

An interesting algorithm that successfully combines the monitoring method with distributed linear regression, has been introduced by Gabel, Keren and Schuster [4]. Their approach is based on Geometric Monitoring over multivariate least squares models of distributed, dynamic data streams. Given a previously computed global model it derives local constraints on the local data at each node. A node only communicates if its constraint is broken. These constraints guarantee that if no node communicates, the global hypothetical model is sufficiently close to the precomputed model.

1.2 Contribution

In this thesis, our main goal is to design an algorithm that solves distributed multivariate regression models, by utilizing the advancements in the field of distributed stream monitoring. In our case, we choose to use Functional Geometric Monitoring [9] as our monitoring method. The objective is to implement the algorithm on a simulated environment and to compare the extracted results to the results of the Gabel's algorithm [4] in terms of communication efficiency.

1.3 Outline

In **Chapter 2**, we give a detailed description of all related work used for the completion of this thesis. Firstly, we discuss about distributed Linear Regression in general, which is followed by the presentation of the basic ideas of Gabel's distributed least squares monitoring algorithm (DILSQ) and Functional Geometric Monitoring (FGM). This chapter aims to cover all necessary definitions and concepts before proceeding to the actual implementation.

In **Chapter 3**, after a brief notation, we give a thorough description of our algorithm, which is based on Functional Geometric Monitoring. We present a distributed algorithm for monitoring a regression model via the geometric approach. Our description includes all algorithm's components and theoretical concepts.

In **Chapter 4**, we conduct several experiments on different datasets in a simulated environment, to evaluate the performance of the algorithm. Firstly, we discuss the results from a synthetic fixed dataset under adverse conditions and compare them with the results of Gabel's approach. In the second set of experiments, we present experimental results on synthetic drift data.

Chapter 5 concludes the thesis by presenting the main contributions and suggests potential directions for future work.

Chapter 2

Background

This chapter is a small description of the basic theoretical background, where this work is based on. We start with a small reference at Linear Regression and its main concepts, we emphasize on distributed Regression and conclude by presenting the core ideas of the algorithms for Monitoring Least Squares Models of Distributed Streams (DILSQ) and Functional Geometric Monitoring (FGM).

2.1 Linear Regression

Linear Regression is an important and widely used technique of supervised machine learning for modeling the behavior of one or more continuous target variables y given the value of a D-dimensional vector of input variables x (features). The case of one target variable is called simple linear regression. For more than one target variable, the process is called multivariate linear regression.

When performing simple linear regression, the four main components are:

- Dependent Variable y — Target variable that will be estimated and predicted
- Independent Variable/Feature x — Predictor variable which is used to estimate and predict
- Slope w —Angle of the line
- Intercept w_0 — Where function crosses the y-axis

The last two, slope and intercept, are the coefficients/parameters of a linear regression model, so when we calculate the regression model, we're just calculating these two.

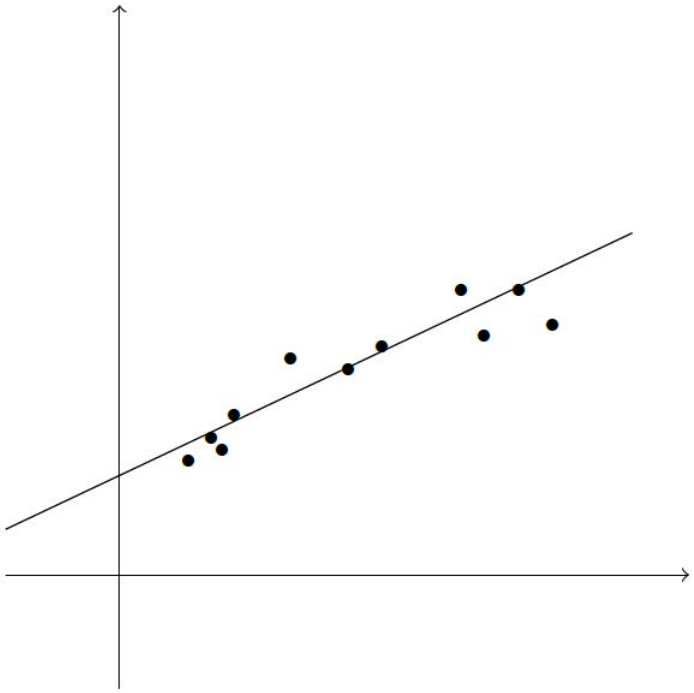


Figure 2.1: Linear regression for m = 1.

In the end, we're trying to find the best-fit line describing the data, out of an infinite number of lines.

In more detail, given a training data set with N observations x^n, where $n = 1, ..., N$, together with corresponding target values y^n, the goal is to predict the value of y for a new value of x. In the simplest approach, this is implemented by constructing an appropriate function $y_w(x)$, called the prediction rule, whose values for new inputs x constitute the predictions for the corresponding values of y. The simplest linear model for regression is one that involves a linear combination on the input variables

$$y_w(x) = w_0 + w_1 x_1 + ... + w_m x_m$$

where m is the number of features. The parameter w_0 allows for any fixed offset in the data and is sometimes called a bias parameter.

2.1.1 Cost Function

One critical point in the creation of a learning model is the choice of the cost function ($L_{w,x}$), which is the measure of success in order to decide if a chosen hypothesis rule $y(x, w)$ is an appropriate solution.The loss is the error in our predicted value of w_0 and w_n. Our goal is to minimize this error to obtain the most accurate value of these coefficients. In this thesis, we will refer to two popular options for regression models , the least squares model and the gradient descent and finally conclude to one that is better to use for our case.

Least Squares

Ordinary Least squares is the algorithm that decides the correctness of the hypothesis rule with respect to the squared loss [2, 11]. The main goal for this algorithm is to find the best fit by minimizing the sum of squares error of the differences between the observed dependent variable (y_{real}) in the given data set and those predicted by the linear function (y_{pred}).

$$argminL_w = argmin\frac{1}{m}\sum_{i=1}^{m}(y_{pred} - y_{real})^2 = argmin\frac{1}{m}\sum_{i=1}^{m}(< w, x_i > -y_i)^2$$

To solve the problem we calculate the gradient of the objective function and compare it to zero. That is we need to solve

$$\frac{2}{m}\sum_{i=1}^{m}(< w, x_i > -y_i)x_i = 0$$

We can rerwite the problem $Aw = c$ where

$$A = (\sum_{i=1}^{m} x_i x_i^T) \quad and \quad c = \sum_{i=1}^{m} y_i x_i$$

If A is invertible the solution of the problem is

$$w = A^{-1}c$$

The OLS estimator is consistent when the regressors are exogenous and optimal when the errors are homoscedastic and serially uncorrelated. Under these conditions, the OLS is the maximum likelihood estimator. In other cases, more sophisticated least

squares variants are used [5], such as *Regularized Least Squares* where the minimized function includes a regularization term to mitigate the effects of outliers and avoid overfitting or *Generalized Least Squares*, which handles correlated measurements and errors by minimizing the Mahalanobis distance.

Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the "lack of fit" in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares cost function as in ridge regression (L2-norm penalty) and lasso (L1-norm penalty). Conversely, the least squares approach can also be used to fit models that are not linear models.

Gradient Descent

Gradient descent is an optimization algorithm that is used when training a machine learning model. It is based on a convex function and tweaks its parameters/ coefficients iteratively to minimize a given function to its local minimum. Firstly, a gradient is a derivative of a function that has more than one input variables. Known as the slope of a function in mathematical terms, the gradient simply measures the change in all weights w_n with regard to the change in error.

In more detail, Gradient Descent(GD) approach aims to minimize the cost function by tweaking its parameters (w and w_0):

$$L_w = \frac{1}{2m}\sum_{i=1}^{m}(w_0 + wx_i - y_i)^2$$

The partial derivatives of the cost function represent the change of the corresponding parameters. In the case of simple linear regression the partial derivatives are

$$\nabla L_{w_0} = \frac{\partial L}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m}(w_0 + wx_i - y_i)$$

$$\nabla L_w = \frac{\partial L}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}(w_0 + wx_i - y_i)x_i$$

After each iteration the estimates of w_0 and w are updated as below:

$$w_{0t} := w_{0t-1} - \alpha\nabla L_{w_0}$$

$$w_t := w_{t-1} - \alpha \nabla L_w$$

where α is the learning rate. The learning rate is a tuning parameter that determines the step size at each iteration while moving toward a minimum of the cost function.

To begin finding the right values, w and w_0 are initialized with some random numbers. Gradient descent then starts at that point and it takes one step after another in the steepest downside direction, until it reaches the point where the cost function L_w is as small as possible.

For gradient descent to reach the local minimum we must set the learning rate to an appropriate value, which is neither too low nor too high. This choice is very important, because if the steps it takes are too big, it may not reach the local minimum because it bounces back and forth between the convex function of gradient descent. If we set the learning rate to a very small value, gradient descent will eventually reach the local minimum but it may need a great number of iterations.

GD can be applied either on the entire training set (batch GD) or stochastically point by point (SGD) in cases where the size of the data set is too large. A common technique used in many machine learning models is a combination of those two techniques, the mini-batch GD.

Comparison between OLS and SGD

OLS compared to SGD is easy, accurate and fast when the data is somewhat contained. In cases where data is large, though, using mini-batch GD is the preferred approach.On the other hand, by using this method, the learning rate and batch size choices are critical and may add some computational and effort overhead.

In this thesis we choose to work on least squares regression as it is a more straightforward method for prediction and analysis of smaller datasets, without the need to use hyperparameters, such as learning rate. What is more, OLS is also used at DILSQ which is the comparator of our work. Thus, it will be more straightforward to draw conclusions based on the fact that our main point of interest is the possible improvements caused by our choice to use Functional Geometric Monitoring, as our monitoring method.

2.1.2 Utilities of Linear Regression

Linear regression has many practical uses. Most applications fall into one of the following two broad categories:

- If the goal is prediction, forecasting, or error reduction, linear regression can be used to fit a predictive model to an observed data set of values of the response and target variables. After developing such a model, if additional values of the target variables are collected without an accompanying response value, the fitted model can be used to make a prediction of the response.
- If the goal is to explain variation in the response variable that can be attributed to variation in the target variables, linear regression analysis can be applied to quantify the strength of the relationship between the response and the target variables, and in particular to determine whether some target variables may have no linear relationship with the response at all, or to identify which subsets of explanatory variables may contain redundant information about the response.

2.1.3 Distributed Regression

Linear Regression models for distributed streams

Distributed machine learning refers to multinode machine learning algorithms and systems that are designed to improve performance, increase accuracy, and scale to larger input data sizes. Increasing the input data size for many algorithms can significantly reduce the learning error and can often be more effective than using more complex methods.

Types of Parallelism in Distributed Learning

There are two main methods in order to distribute machine learning model training: model parallelism or data parallelism. In model parallelism, the model is segmented into different parts so to run in parallel. Each part can run on the same data in different nodes.This approach may decrease the need for communication between workers, as workers only need to synchronize the shared parameters.

In data parallelism the training data is divided into multiple subsets. Each subset can run on the same replicated model in a different node.There must be a synchronization method for model parameters at the end of the batch computation to ensure they are training a consistent model because the prediction errors are computed independently on each machine. These two methods however are not mutually exclusive, as they can be combined into a hybrid approach.

We however focus our study on data parallelism due to its fault tolerance nature, a property that is of high importance in distributed settings, and simple to implement.

Distributed Linear Regression

Distributed regression is a suite of methods that enable researchers to conduct multi-database regression analysis without the need to centrally combine all individual-level data from participating sites. Typically, the distributed sites compose a graph, each holding a portion of the data, and the goal is to solve the regression model- in our case the linear regression model - of the aggregated data.

We consider a distributed computing environment with a classic star network topology, comprised of k remote sites and a central coordinator. Each site has a copy

of the linear regression model, and has a subset of the whole data set . The training is done solely on the k remote sites, and the coordinator is responsible for synchronizing all the models into one global predictive model.

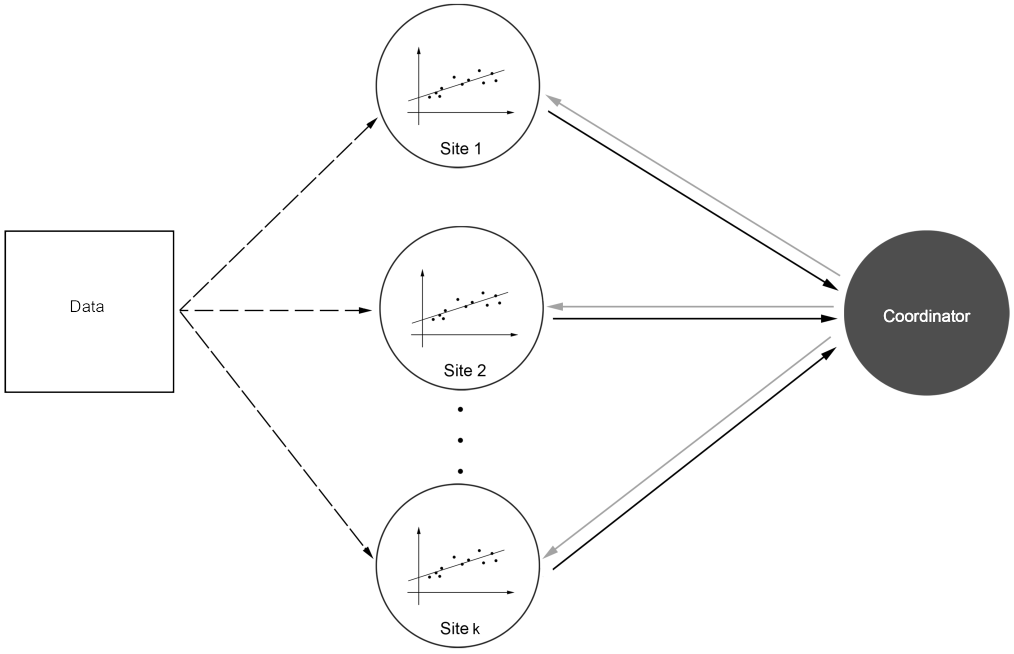


Figure 2.2: Distributed linear regression on a k-star network topology

Regression is a powerful modeling tool, so, extensive research has been made for both distributed and centralized modeling and monitoring; for a comprehensive survey, see [10].

While there are efficient solutions for computing the linear regression model over distributed nodes, only very few are dealing with monitoring it. The monitoring idea, though, is very intriguing, as we could find an accurate solution without actually relearning the model with every data update.

Prior works on distributed regression via monitoring

In DReMo [1] is proposed a distributed algorithm with low resouce overhead, for monitoring the quality of a regression model in terms of its coefficient of determination (R^2 statistic). When the nodes collectively determine that R^2 has dropped below a fixed therhold, the linear regression model is recomputed via a network-wide convergecast and the updated model is broadcast back to all nodes.

Another interesting approach is introduced in [2] ,where the problem reduces to monitoring a ratio. Here, the client (global) ratio theshold is converted into conditions on individual distributed data sources (local sites). Whenever the condition associated with a source is violated, the source pushes its data values to an aggregator, which in turn pulls data values from other sources to determine whether the client threshold condition is indeed violated.

We aim to monitor the model error as it is a more general approach: prediction error and fit can be inferred from model error but not vice versa.

2.2 Monitoring OLS of Distributed Streams

Gabel, Keren and Schuster [4] presented a distributed algorithm that monitors multivariate regression model with the help of Geometric Monitoring. Local constraints, which are derived on the local data at each node, are used to decide when to recompute the model. As long as the constraint at a node is not broken, it is guaranteed that, without further communication, the global hypothetical model is sufficiently close to the previously computed global model.

2.2.1 Problem Definition

Let $(x^1, y^1), (x^2, y^2), ..., (x^n, y^n)$ be a set of n observation pairs of $m < n$ features and one target variable y, where x^i are column vectors in $\Re^m$, and y^i are the corresponding response scalars. We denote two vextor X, y that:

$$X = \begin{bmatrix} {x^1}^T \\ {x^2}^T \\ \vdots \\ {x^n}^T \end{bmatrix}, \text{ where each is } x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \text{ and } y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{bmatrix}$$

The optimal solution to this problem is OLS, which is given by

$$w = (X^T X)^{-1} X^T y$$

The main goal is to maintain an accurate estimation w_0 of the current global OLS model w , as long as new observations (x^i, y^i) are distributed across k nodes. In order to achieve that, the model estimation error is monitored.

Let w_0 be the existing model, previously computed at some point in the past, and let w be the hypothetical OLS model from current observations. Given an error threshold ϵ an alert is raised only if

$$||w - w_0|| > \epsilon$$

minimizing this way, communication.

Notation

We can define $A = X^T X$ and $c = X^T y$ so that OLS method becomes:

$$w = (X^T X)^{-1} X^T y = A^{-1} c$$

Data in each node j can be written as: $A^j = {X^j}^T X^j$ and $c^j = {X^j}^T y$. From now, we will express observations as $\{A^j, c^j\}_k$ instead of $\{x^i, y^i\}_n$. We will write global A as the sum of local matrices A^j, like $A = \sum_{j=1}^{k} A^j$. Likewise, global matrix c can be writen as $c = \sum_{j=1}^{k} c^j$.
The global coefficient w can be calculated form the averages of local matrices A^j, c^j:

$$w = (\frac{1}{k}\sum_{j=1}^{k} A^j)^{-1} (\frac{1}{k}\sum_{j=1}^{k} c^j) = k(\sum_{j=1}^{k} A^j)^{-1} \frac{1}{k}(\sum_{j=1}^{k} c^j) = A^{-1} c$$

We also describe local drifts as the difference of local data from its initial values during sync:

$$D^j = A^j - A_0^j \text{ and } d^j = c^j - c_0^j$$

2.2.2 Geometric Monitoring and Safe Zone

Monitoring the above condition is difficult because the global model cannot be inferred from the local model at each node. To overcome this obstacle, we turn to geometric monitoring.

Geometric Monitoring monitors whether a function of distributed data streams crosses a threshold [12]. Constraints are imposed on local data at the nodes, rather than on the function of the global aggregate.

Given a function of the average of all local data and the threshold, we compute a convex safe zone for each node. Convexity is crucial at this approach. As long as, local data stay inside the safe zone,the global average is guaranteed to not cross the threshold. Communication is used, only when local data drifts outside the safe zone. In such case, coordinator is gathering data from all nodes and recomputes w_0 and the safe zone.

We use the euclidean norm to define a good convex safe zone, which monitors local drifts D^j, d^j to be sufficiently small:

$$(D^j, d^j) \in C \Rightarrow ||(\hat{A}_0 + D^j)^{-1}(\hat{c_0^{-1}} + d^j) - \hat{A_0}^{-1}\hat{c_0}|| \leq \epsilon$$

After some calulations, we derive the following convex constraint:

$$\epsilon||\hat{A_0}^{-1}D^j|| + ||\hat{A_0}^{-1}d^j|| + ||\hat{A_0}^{-1}D^j w_0|| \leq \epsilon \Rightarrow ||w - w_0|| \leq \epsilon$$

where D^j is the drift vector of $A = X^T X$, d^j is the drift vector of $c = X^T y$ and A_0, w_0 are the global estimates. The above function is convex as a sum of euclidean norms. The derivation of the contstraint C is quite technical and the details are available at paper [4].

Performance

The DILSQ algorithm is able to avoid costly communication and model recomputations, while guaranteeing bounded model error. Evalutation on real-world datasets shows a communication reduction of up to two orders of magnitude. Furthermore, based on simulations on synthetic datasets, it is safe to assume that the algorithm scales well with the number of nodes.

2.2.3 DILSQ algorithm

The DILSQ Algorithm is shown bellow. Alg.1 shows the resulting monitoring algorithm each node runs. Each node applies the local constraint to its own data. When a violation occurs at any node, it is reported to the coordinator node. In Alg2, the coordinator poll all nodes for their local data, computes the updated global estimates and distributes them back to the nodes.

This is the simplest violation resolution protocol. Similarly, the coordinator can use any algorithm to compute A_0^{-1}, β_0.

Algorithm 1: Node j update with new observation x,y.

update local state and local drifts D^j, d^j;
if $\epsilon||\hat{A_0}^{-1}D^j|| + ||\hat{A_0}^{-1}d^j|| + ||\hat{A_0}^{-1}D^j w_0|| > \epsilon$ **then**
    report violation to coordinator;
    receive new $w_0, \hat{A_0}^{-1}$ from coordinator;
    $(A_0^j, c_0^j) \leftarrow (A^j, c^j)$;
end

Algorithm 2: Coordinator violation resolution algorithm.

poll all nodes for A^j, c^j;
compute updated global estimate $\hat{A_0}^{-1}, w_0$ from A^j, c^j and distribute;

2.3 Functional Geometric Monitoring

A substantial improvement on the GM protocol is the Functional Geometric Monitoring (FGM) protocol. FGM describes a technique which can be applied to any monitoring problem in order to perform distributed monitoring with communication costs that are lower , often by orders of magnitude, compared to centralizing all data to a coordinator. In order to accomplish monitoring, FGM should be parameterized by a problem-specific family of functions.

2.3.1 Approximate Query Monitoring

We assume that there are k distributed sites, and at each site, a local stream is collected, denoted as a vector $S_i(t)$. This local state vector changes as stream updates arrive. Without loss of generality, assume that the global stream state is the average of the local stream states i.e. $S(t) = \frac{1}{k}\sum_{i=1}^{k} S_i(t)$. Every site communicates with the coordinator, where users pose quaries on the global stream. The main job of the coordinator is to maintain at all times a close estimate $Q(E(t))$ of the true value of the query $Q(S(t))$, so that

$$Q(S(t)) \in (1 \pm \varepsilon)Q(E(t))$$

When a violation occurs, each site transmits its drift vector $X_i(t) = S_i(t) - E_i$ and the cordinator updates global estimate $E(t)$.

2.3.2 Safe functions

The system is in a safe state as long as $E + \frac{\sum_{i=1}^{k} X_i}{k} = S \in A$, where A is the admissible region. To guarantee this , FGM creates a safe function φ depending on A, E and k. Each site tracks each safe function as X_i is updated. System safety is guaranteed when

$$\psi = \sum_{i=1}^{k} \varphi(X_i) \leq 0$$

Hence, the problem of watching a boolean conjunction in order to detect a violation has been converted into a sum-monitoring problem. The introduction of safe functions offers significant opportunities for improved distributed stream monitoring.

2.3.3 Safe function and convexity

The selection of a good safe function is very important as it can have a huge impact on the communication efficiency of the system. Of course, not all safe functions for a particular admissible region A are equally good. It should be the case that φ should take as small values as possible, so that ψ remains negative longer and prolong the rounds.

Good safe functions are described by the following theorem:

Theorem 1 *A (A,E)-safe function φ has maximal quiescent region, among all (A,E)-safe-functions, for every k, if,*

- *φ is convex*
- *L(φ) is a maximal convex subset of A and*
- *φ is level-minimal*

All technical details as well as the proof of this theorem can be found in [9]. As we can see, convexity is a central point in the monitoring problem.

2.3.4 The basic FGM protocol

The FGM protocol work in rounds and monitors the threshold condition

$$\sum_{i=1}^{k} \phi(X_i) \leq 0$$

At the beginning of a round the coordinator knows the current state of the system E=S. It selects a safe function φ. At each point in time let $\psi = \sum_{i=1}^{k} \varphi(X_i)$ The round's steps are:

1. At the beginning of a round, the coordinator ships φ to every site. Local sites initialize their drift vectors to 0. With these settings, initially it is $\psi = k\varphi(0)$.
2. Then, the coordinator initiates a number of subrounds, to be described below. At the end of all subrounds, $\psi \geq \epsilon_\psi k\varphi(0)$ for some small ϵ_ψ
3. Finally, the coordinator ends the round by collecting all drift vectors and updating E.

Execution of subrounds: The goal of each subround is to monitor the condition $\psi \leq 0$, with a precision of roughly ϑ, performing as little communication as possible. Subrounds are executed as follows:

1. At the beginning of a subround, the coordinator knows the value of ψ. It computes $\vartheta = \frac{\psi}{2k}$ and ships ϑ to each local site. Also, the coordinator initializes a counter $c = 0$. Each local site records its initial value $z_i = \varphi(X_i)$, where $2k\vartheta = -\sum_{i=0}^{k} z_i$ and initializes a counter $c_i = 0$.
2. Each local site i maintains its local drift vector X_i, as it processes stream updates. When X_i is updated, site i updates its counter

   $$c_i := max\{c_i, \left\lfloor \frac{\phi(X_i) - z_i}{\vartheta} \right\rfloor\}$$

   If this update increases the counter, the local site sends a message to the coordinator, with the increase to c_i
3. When the coordinator receives a message with a counter increment from some site, it adds the increment to its global counter c. If $c > k$ the coordinator finishes the subround by collecting all $\varphi(X_i)$ from all local sites, recomputing ψ. If $\psi \geq \epsilon_\psi k\varphi(0)$, the subrounds end, else another subround begins.

Chapter 3

Monitoring Regression Model via FGM

We now present our approach to distributed multivariate regression that utilizes the FGM protocol. It is based on previous work [6] which implements a similar protocol for other machine learning algorthms, such as neural networks. Our method focuses solely on least square regression models for distributed streams and the main goal is a communication-efficient monitoring algorithm that allows communication between the workers and the coordinator only when nessesary.

The distributed protocol is similar to the FGM protocol [9] with some adjustments to the safe function and the averaging procedure in order to adapt to the regression problem.

3.1 Problem Definition

Let $\{(x^1, y^1), (x^2, y^2), ...(x^n, y^n)\}$ be a set of n observation pairs of $m < n$ features and one target variable. Our main goal is to minimize the sum of squared errors between y^i to the mapping of x^i, as follows:

$$argmin \frac{1}{m} \sum_{i=1}^{m} (< w, x^i > - y^i)^2$$

In other words, we seek a model w that minimizes the above cost function.

3.1.1 Regression Method

As it is mentioned above, in our implementation we will calculate w by using ordinary least squares method. In any case, our algorithm is independent of how the model is computed, so we can easily change the model to gradient descent as well as more complex least squares variants (GLS, RLS etc).

The optimal solution for this convex problem is given by

$$w = (X^T X)^{-1} X^T y$$

where X is the $n \times m$ matrix of row vectors $X = ({x^1}^T, ..., {x^n}^T)^T$ and y is the column vector composed of response scalars $y = (y^1, ..., y^n)^T$.

3.1.2 Distributed Streams and averaging procedure

Let's assume that the observations $\{(x^i, y^i)\}$ are distributed across k nodes, and that these observations are dynamic - they change over time, as nodes receive new observations that replace the older ones. Each site maintains and computes an OLS model and produces its own local A^j, c^j.

Every site communicates with a coordinator, where users pose queries on the global stream. Also, the coordinator maintains an estimated global model w_0.

As data evolves, we wish to maintain an accurate estimation of the current global model w .We choose the global model to be calculated as the average of the local data A^j, c^j, as shown at 2.2.1.

3.2 Admissible region & Safe Function

Firstly, we have to present a safe state in our monitoring algorithm. In our case we choose monitoring the norm of the model w, so that $Q(w) = \|w\|$ (with $\|.\|$ denoting the Euclidean norm).

Assume at a time t, the coordinator holds the estimate w_0, and reports $||w_0||$ to the user. This estimate should stay the same, as long as $w(t) \in A$ with

$$A = \{x \in \mathbb{R}^D | \, \|x\| \in (1 \pm \varepsilon) \, \|w_0\|\}$$

As mentioned above we are in a safe state as long as $\frac{\sum_{j=1}^{k} X_j}{k} \in A$. Each site j tracks its local drifts $X_j = (D^j, d^j)$ by monitoring safe function $\varphi(D^j, d^j)$, as observations are updated. System safety is guaranteed by tracking the sign of the sum

$$\psi = \sum_{j=1}^{k} \varphi(D^j, d^j) \leq 0$$

Turning to our case of monitoring the norm $\|w\|$, our safe function can be derived from DILSQ convex constraint and is defined as

$$\epsilon||\hat{A_0}^{-1} D^j|| + ||\hat{A_0}^{-1} d^j|| + ||\hat{A_0}^{-1} D^j w_0|| \leq \epsilon \Rightarrow$$

$$\epsilon||\hat{A_0}^{-1} D^j|| + ||\hat{A_0}^{-1} d^j|| + ||\hat{A_0}^{-1} D^j w_0|| - \epsilon \leq 0 \Rightarrow$$

$$\phi(D^j, d^j) = \epsilon||\hat{A_0}^{-1} D^j|| + ||\hat{A_0}^{-1} d^j|| + ||\hat{A_0}^{-1} D^j w_0|| - \epsilon$$

3.3 Sliding Window

As we are processing data streams, which are data that continuously change values over time, so they have unbounded length, it is important to find a way to process them efficiently. One way is by using a sliding window model where we consider recent data more useful than older. As a result we calculate w by using only the last n elements to arrive in the stream, where n is the window size. Each node computes each local w from last seen n samples, while w_0 is built from the last samples before sync. Computing drift, however, requires subtracting observations that left the sliding window. If w and w_0 do not overlap, then clearly $D^j = A^j - A_0^j$ and $d^j = c^j - c_0^j$.

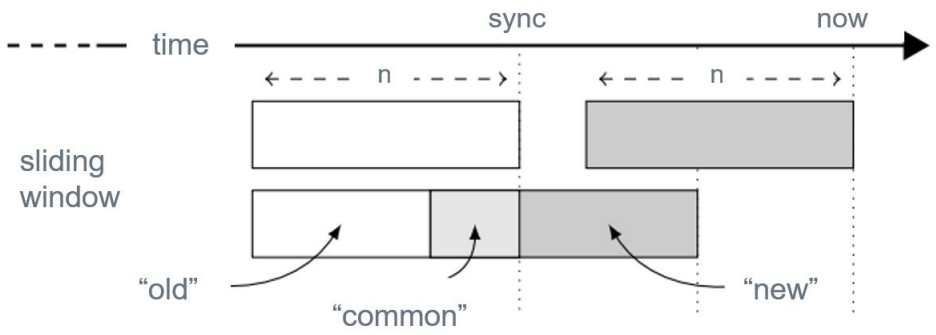


Figure 3.1: Sliding Window

It is also possible, however, that the current window overlaps the window used to build w_0. Above Figure illustrates this case: D^j, d^j becomes the sum of new samples from minus the sum of old samples. As a result, we are not delve into further strategies for choosing the optimal warm-up dataset size.

3.4 Algorithm Description

We describe R-FGM: a communication-efficient monitoring algorithm for multivariate linear regression of distributed data streams by using the basic FGM protocol. In the beginning, the coordinator proceeds through a warming state, where it collects data from all sites in order to calculate an initial estimate w_0.

After the warm-up, coordinator ships the parameters w_0, φ, k to all sites. The size of the warm-up dataset is a choice to be made by the user, and its size can affect the communication cost of the early rounds. In our study ,we evaluate the algorithm without taking into consideration this warm-up state. The monitoring method is the basic FGM protocol as mentioned at 2.3.4.

Alg. 3 shows the resulting monitoring algorithm each node runs. Each node monitors the safe function mentioned at 3.2 to its own data.

Also Alg 3. describes the behavior of the coordinator. As mentioned before, any algorithm can be used to compute w.

Algorithm 3: R-FGM

require: $\phi, \varepsilon, \epsilon_\psi, m, k, WindowSize$

A : *Initialization at coordinator*

warm-up and produce w_{init};

`/* safe function is explained above */`

$w_o \leftarrow w_{init}, counter \leftarrow 0, \psi \leftarrow k\phi((0,0)), \vartheta \leftarrow -\frac{\psi}{2k}$;

send $w_0, \hat{A_0}^{-1}, \vartheta$ to all sites and start the first round ;

B : *Site j starts a new round*

receive new $w_0, \hat{A_0}^{-1}, \vartheta$ from coordinator;

$A^j \leftarrow A_0^j, c^j \leftarrow c_0^j$;

$\vartheta_j \leftarrow \vartheta, counter_j \leftarrow 0, z_j \leftarrow \phi((0,0))$;

C : *Site j starts a new subround*

$\vartheta_j \leftarrow \vartheta, counter_j \leftarrow 0, z_j \leftarrow \phi((D^j, d^j))$

D : *Site j update with new observation x,y*

update (A^j, c^j) with $(x, y)_{new}$;

insert $(x, y)_{new}$ to head of sliding window ;

retrieve $(x, y)_{old}$ exiting end of sliding window ;

subtract $(x, y)_{old}$ from (A^j, c^j);

$(D^j, d^j) \leftarrow (A^j - A_0^j, c^j - c_0^j)$;

if $\left\lfloor \frac{\phi((D^j, d^j)) - z_j}{\vartheta} \right\rfloor > counter_j$ **then**

  increment $\leftarrow \left\lfloor \frac{\phi((D^j, d^j)) - z_j}{\vartheta} \right\rfloor - counter_j$;

  $counter_j \leftarrow \left\lfloor \frac{\phi((D^j, d^j)) - z_j}{\vartheta} \right\rfloor$;

  send increment to coordinator ;

end

E : *Coordinator receives an increment:*

$counter \leftarrow counter +$ increment;

if $counter > k$ **then**

  collect all $\phi((D^j, d^j))$ from all sites ;

  $\psi \leftarrow \sum_{j=1}^{k} \phi((D^j, d^j))$;

  if $\psi \geq \epsilon_\psi k\phi(0)$ **then**

    collect all (D^j, d^j) from all sites ;

    update $\hat{A_0}^{-1}, w_0$ with (D^j, d^j);

    $counter \leftarrow 0, \psi \leftarrow k\phi(0,0), \vartheta \leftarrow -\frac{\psi}{2k}$;

    send $\hat{A_0}^{-1}, w_0, \vartheta$ to all sites to start a new round (B);

  end

  else

    $counter \leftarrow 0, \vartheta \leftarrow -\frac{\psi}{2k}$;

    send ϑ to all sites to start a new subround (C);

  end

end

Chapter 4

Experimental Evaluation

4.1 Experimental Setup

The objective of this section is to investigate the practical performance of the Regression FGM protocol in a simulated distributed system. Furthermore, we aim to compare it with other similar protocols. The DILSQ algorithm has been experimentally proven to be much more communication efficient than any static averaging protocol. Therefore, we primarily intent to empirically confirm the communication gains against DILSQ.

Our experiments will be conducted on two distinct online learning scenarios, with the use of two types of synthetic datasets.The first one, is the fixed dataset, a distributed stream with no concept drift and the other is the drift dataset, which consists of a distributed stream with concept drift.

For each dataset, we run through the data, simulate the nodes and the coordinator, count messages in bytes and keep track of the resulting true models w and the current monitored models w_0. Our simulations use discrete time (epochs). Our main performance metrics are accuracy, rounds and normalized messages- the average messages sent per epoch by each node (in bytes).

In order to gain a better understanding of the performance of our monitoring algorithm, we executed a few sets of experiments with different parameters.

The parameters that were changed between experiments were:

- **k**: the number of nodes
- **features**: the number of features for regression
- **error**: the monitoring error
- **window size**: the size of the sliding window

while the constants of the configuration were:

- **points**: the number of data points
- **epoch**: the number of epochs used to create drift dataset
- **var**: the variance used for bias on dataset creation
- **window step**: the size of slider for sliding window

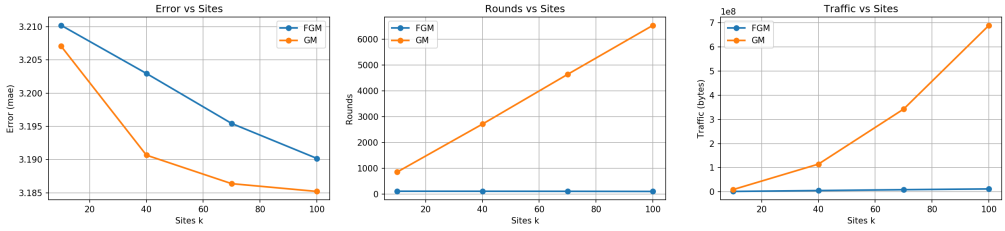
4.2 Synthetic Datasets

As mentioned before, we use two types of synthetic dataset. In the fixed dataset the real model $w_{real} \epsilon \mathbb{R}^m$ is fixed, with elements drawn i.i.d from $N[0,1]^2$. We generate R rounds with k nodes, each receiving at each round a new data vector x of size m and scalar y. Vector x is drawn i.i.d from $N[0,1]$, and $y = x^T w_{real} + n$ where $n \sim N(0, \sigma^2)$ is Gaussian white noise of strength σ.

In the drift dataset the coefficients of w_{real} take recurring values that change suddenly at the beginning of each epoch and are fixed during the rest of the epoch.We generate observations for E epochs using the same procedure.

4.3 Fixed Dataset

Default parameter values are k = 10 nodes, m = 10 features (dimensions), noise magnitude $\sigma = 5$, window size W = 2000 error threshold $\epsilon = 0.1$. Furthermore, we generate 150.000 (x,y) pairs of data. All parameters not shown on the graphs, remain fixed at these default values.



(A) (B) (C)

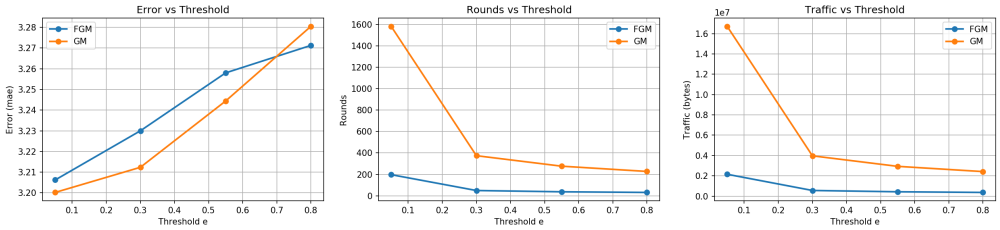
Figure 4.1: Diagrams of accuracy, rounds and total traffic over sites (fixed dataset)

Figure 4.1 shows the behavior while sites number increases. We observe that the predictive accuracy is approximately the same for the two monitoring protocols, both staying at low levels.

However, this accuracy is achieved by R-FGM with substantially less communication. It is evident from subfigures B,C depicting rounds and traffic over sites , that R-FGM communication scales much better than DILSQ, as the number of remote sites increases. Moreover, these improvements in communication efficiency come at almost no cost, as the model accuracy trained by R-FGM is insignificantly worse than DILSQ.

In Figure 4.2, we can observe the model accuracy, number of rounds and the total traffic as "threshold"'s value increases. The model accuracy is diminishing as long as threshold increases for both monitoring algorithms, which is expected because of the lack of communication. That is a natural result of the relaxation of the violation condition.

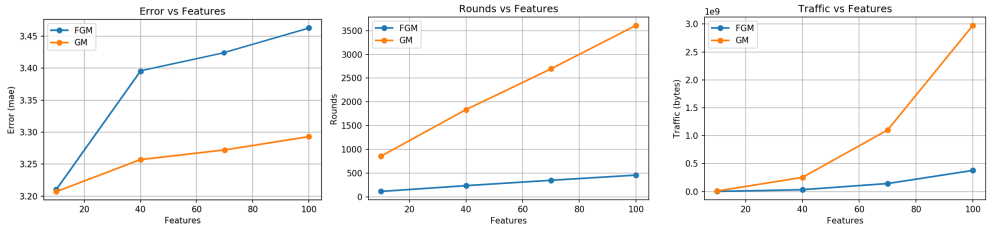
It is also notable that both curves about the number of rounds and traffic in figures B,C follow the same decreasing direction. As we can see, though, R-FGM does less rounds than DILSQ, for every threshold value, even for smaller ones. In other words, R-FGM achieves less communication for different values of threshold.



(A) (B) (C)

Figure 4.2: Diagrams of accuracy, rounds and total traffic over threshold (fixed dataset)

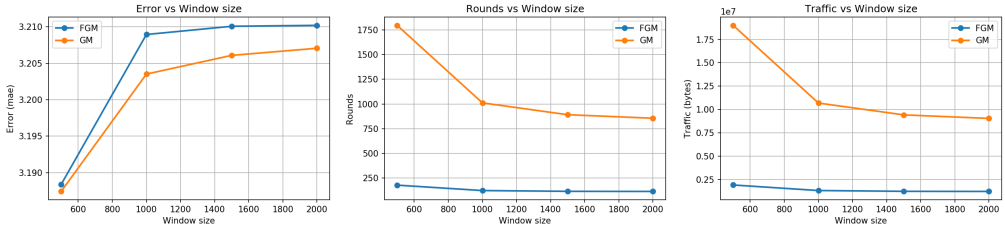
Figure 4.3 addresses the accuracy, rounds and traffic over different values of features. We observe that DISLQ achieves slightly better model accuracy than R-FGM for more dimensions. This minor gain, though, has a significant communication cost. A typical example which highlights this assumption, is that for 100 features R-FGM has low error, almost equal to that of DILSQ in approximately 500 rounds. On the other hand DILSQ needs around 3500 rounds for the exact same scenario.



(A) (B) (C)

Figure 4.3: Diagrams of accuracy, rounds and total traffic over features (fixed dataset)

Figure 4.4 shows what happens for different window sizes. For small window size constraint violations are more frequent and both monitoring algorithms need more rounds in order to maintain high accuracy. It is obvious from rounds and total traffic curves , though, that R-FGM is not as much affected as DILSQ in terms of communication cost from window size.



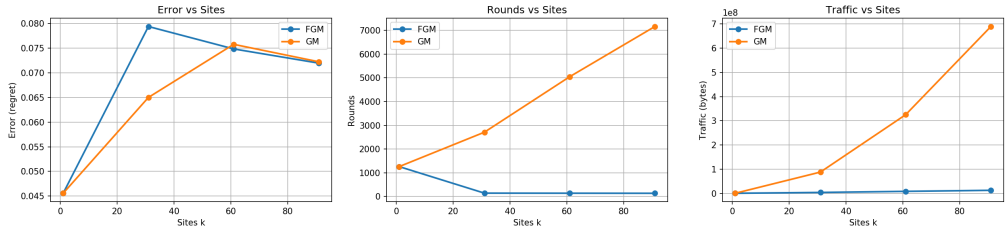
(A) (B) (C)

Figure 4.4: Diagrams of accuracy, rounds and total traffic over window (fixed dataset)

4.4 Drift dataset

Default parameter values are k = 10 nodes, m = 10 features (dimensions), noise magnitude $\sigma = 5$, window size W = 2000 error threshold $\epsilon = 0.1$. Furthermore, we generate 3 epochs of 40 (x,y) pairs of data. All parameters not shown on the graphs, remain fixed at these default values.

First we are going to analyse the results of experiments with the use of the synthetic drift dataset without any warm up.

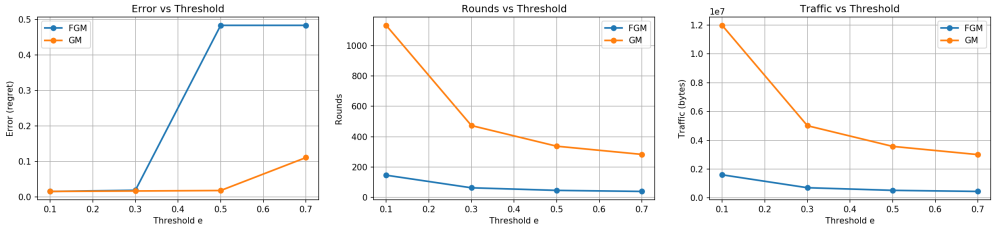


(A) (B) (C)

Figure 4.5: Diagrams of accuracy, rounds and total traffic over sites (drift dataset without warm up)

Figure 4.5 depicts the behaviour of the two monitoring algorithms regarding accuracy, rounds and total traffic while increasing the number of sites. Regarding model accuracy, we can see that both algorithms have an increasing error which has a peak point and then it is diminishing and staying at low levels.

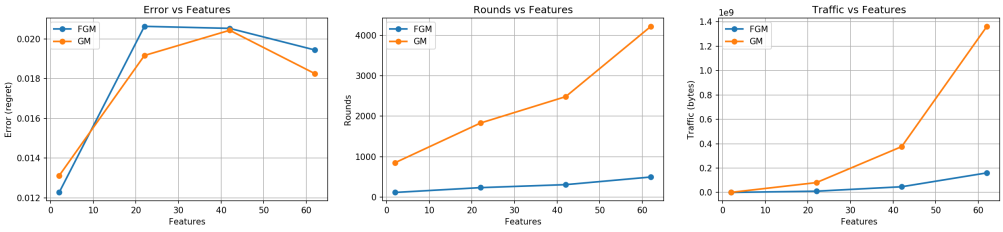
On the other hand, we observe a completely different behavior regarding communication cost. We notice that both algorithms for a small number of nodes need approximately the same number of rounds in order to keep the model accurate.As we proliferate sites, though, R-FGM achieves to discrease the number of rounds along with total traffic and stabilizes it at satisfying levels. On the contrary, DILSQ has a growing communication cost as long as sites are increasing.



(A) (B) (C)

Figure 4.6: Diagrams of accuracy, rounds and total traffic over threshold (drift dataset without warm up)

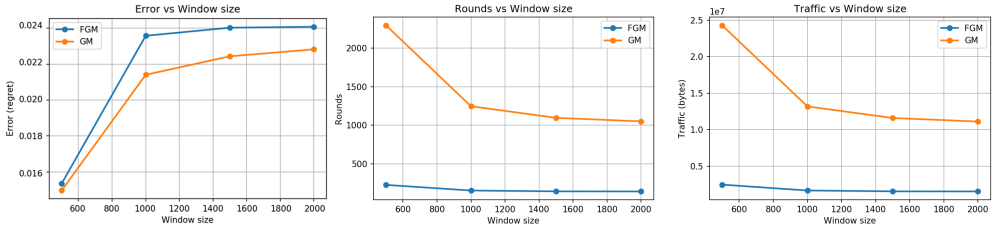
Figure 4.6 addresses the accuracy, rounds and traffic over different values of threshold. In that case, we observe that R-FGM is not as accurate as DILSQ for larger numbers of threshold, which are, though, below the maximum acceptable error. Furthermore, both algorithms improve their communication cost as we relax the error boundary, which is quite logical ,but R-FGM has a significant communication gain regardless the value of the threshold.



(A) (B) (C)

Figure 4.7: Diagrams of accuracy, rounds and total traffic over features (drift dataset without warm up)

Figure 4.3 shows what happens for different number of features. Again, both algorithms have similar curves,depicting low values of accuracy, but DILSQ needs a lot more communication in order to keep it than R-FGM.

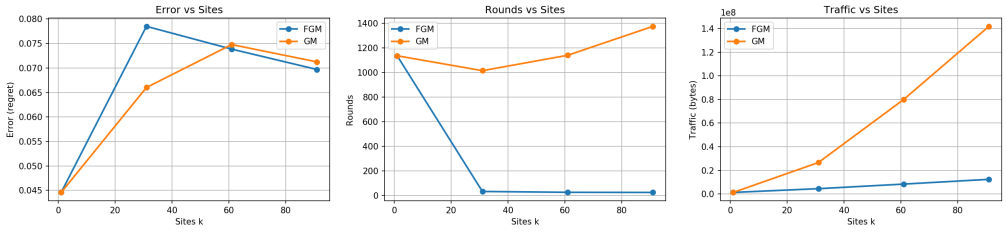


(A) (B) (C)

Figure 4.8: Diagrams of accuracy, rounds and total traffic over window size (drift dataset without warm up)

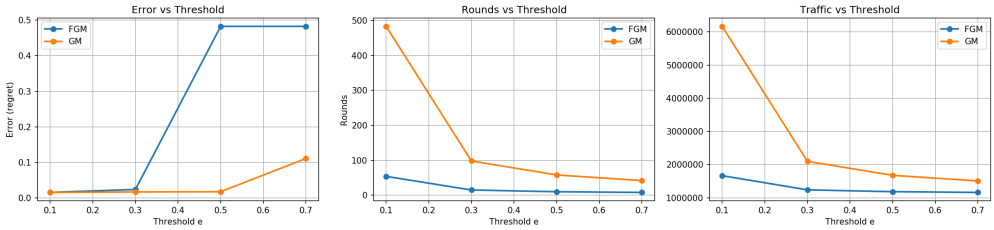
Figure 4.8 depicts the behaviour of the two monitoring algorithms regarding accuracy, rounds and total traffic while increasing the number of window size. Like with fixed dataset, small window size affects significantly communication cost for DILSQ in contrast with R-FGM.

Next we are going to have a brief look, at the results of the experiments using the same synthetic dataset with warm-up, so that we can have a clear opinion about the possibilities of the two algorithms.



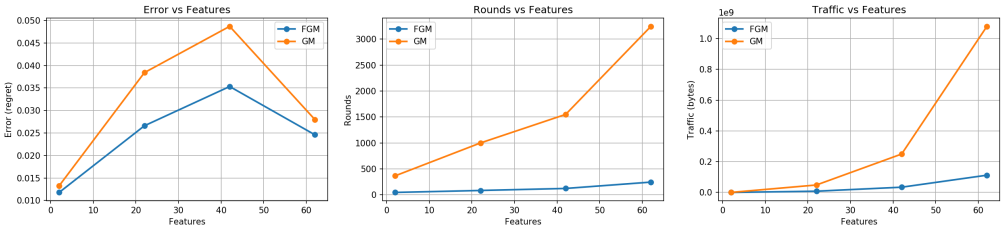
(A) (B) (C)

Figure 4.9: Diagrams of accuracy, rounds and total traffic over sites (drift dataset with warm up)



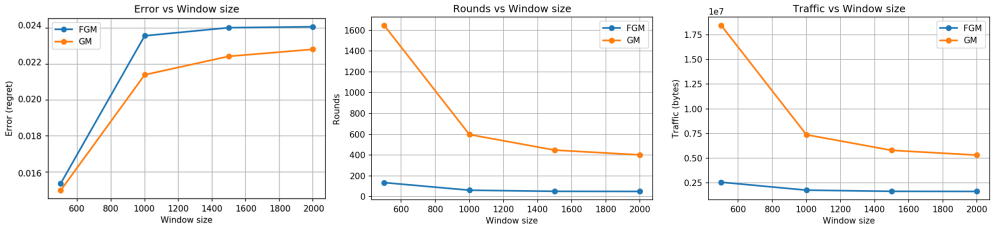
(A) (B) (C)

Figure 4.10: Diagrams of accuracy, rounds and total traffic over threshold (drift dataset with warm up)



(A) (B) (C)

Figure 4.11: Diagrams of accuracy, rounds and total traffic over features (drift dataset with warm up)



(A) (B) (C)

Figure 4.12: Diagrams of accuracy, rounds and total traffic over window size (drift dataset with warm up)

Figures 4.9, 4.10, 4.11, 4.12 addresses the behaviour of the two monitoring algorithms regarding accuracy, rounds and traffic. All curves verify our prior findings when using the dataset without warm-up.

Something we can point out, though, is that DILSQ has a huge overhead in terms of communication when a warm-up does not precede. That can be clearly seen, from figures 4.10 where for a low value of error DILSQ needs around 500 rounds and $6*10^6$ bytes of traffic to achieve high accuracy while R-FGM needs arround 80 rounds and $1.8 *10^6$ bytes of traffic for approximately same accuracy.

4.5 Rounds and Subrounds Relation

Figure 4.13 depicts the relation between rounds and subrounds for R-FGM algorithm, while running the experiments with the drift dataset with warm-up. There is one diagram for each variable (threshold, features, nodes, window).

As we can see, in cases where we would expect heavier communication, such as a very short value of threshold, a small number of nodes, a short window size or many features, the communication cost is smaller than expected, because many subrounds take place. This is a result of the fact that the cost of a subround is far less than that of a round.

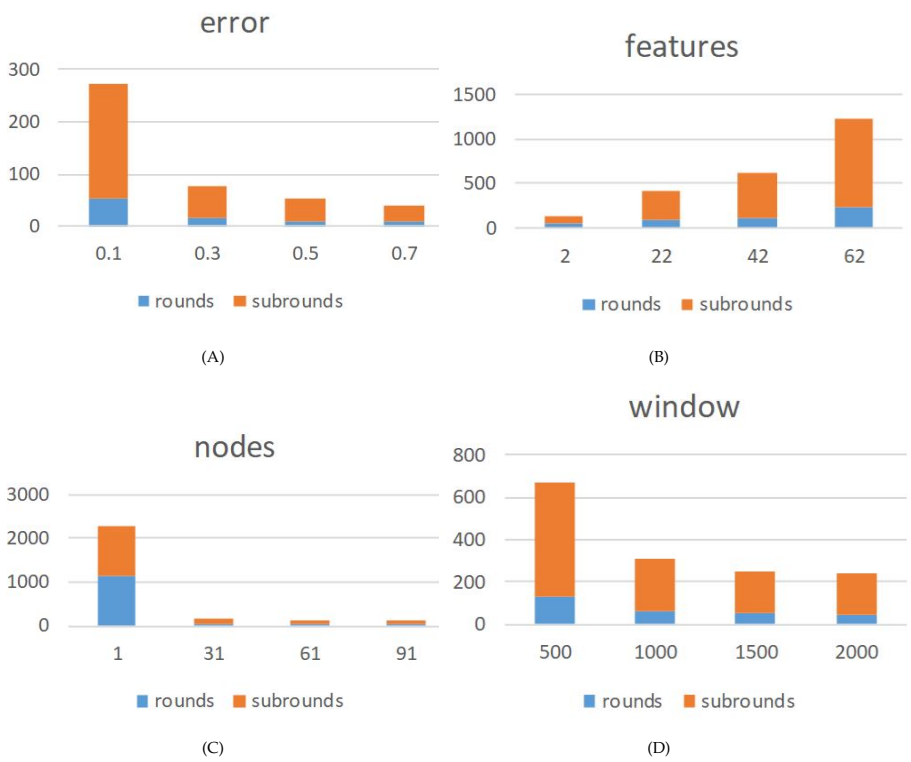


Figure 4.13: Diagrams depicting the ammount of subrounds over the amount of rounds for R-FGM

Chapter 5

Conclusions & Future Work

We have proposed R-FGM, a monitoring algorithm for multivariate least squares models of distributed data streams, which is based on Functional Geometric Monitoring and aims to offer improvements over previous techniques in terms of performance, scalability and robustness.

During the implementation of this thesis, we have compared our approach with DILSQ, who is the first communication-efficient monitoring algorithm for least-squares regression models that limits the error in model coefficients, aiming to avoid costly communication and model recomputations, while guaranteeing bounded model error.

For this purpose, we ran multiple experiments on a simulated environment with different kinds of synthetic datasets and a variance of variables and concluded that R-FGM is more efficient than DILSQ in terms of communication for a big population of nodes or a great number of features. At the same time, it is applicable to traditional learning scenarios, but also to non-static ones with concept drift.

We emphasize that correctness of our method is independent of the alghorithm used to compute the regression model. Hence, it is straightforward to adapt our method to other settings. We could, for example, test the algorithm with more sophisticated least squares variants, such as regularized least squares or generalized least squares.

Last but not least, another interesting future direction would be to implement a real system that uses R-FGM for distributed leat squares regression.

Bibliography

[1] K. Bhaduri, K. Das, and C. R. Giannella. *Distributed Monitoring of the R 2 Statistic for Linear Regression*, pages 438–449. Proceedings of the 2011 SIAM International Conference on Data Mining, 2011.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*, pages 137–144. Springer-Verlag, 2006.

[3] J. B. G. Mateos and G. Giannakis. *Distributed sparse linear regression*. IEEE Trans. Sig. Proc., 2010.

[4] M. Gabel, D. Keren, and A. Schuster. *Monitoring Least Squares Models of Distributed Streams*. Association for Computing Machinery, 2015.

[5] F. Hayashi. *Econometrics*. Princeton Univercity Press, 2000.

[6] V. Konidaris. *Machine Learning Algorithms via Geometric Monitoring*. Diploma thesis, Technical University Crete, 2019.

[7] C. G. Lopes and A. H. Sayed. *Distributed Adaptive Incremental Strategies: Formulation and Performance Analysis*. IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, 2006.

[8] V. Samoladas. *ddssim. https://github.com/vsamtuc/ddssim.* 2018.

[9] V. Samoladas and M. Garofalakis. *Functional Geometric Monitoring for Distributed Streams*. 22nd International Conference on Extending Database Technology (EDBT), 2019.

[10] A. H. Sayed. *Adaptive Networks*. Proc. IEEE, 2014.

[11] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*, pages 120–125. Cambridge University Press, 2014.

[12] I. Sharfman, A. Schuster, and D. Keren. *A geometric Approach to Monitoring Threshold Functions Over Distributed Data Streams*. Association for Computing Machinery, 2007.

Appendix A

A Simulator for monitoring data streams

In order to simulate and evaluate our algorithm, we built an algorithm that simulates protocols for distributed data streams and it will comply with our experiments easily. Our simulator is of general purpose, easily customizable and extensible.

The simulator consists of independent modules that implement environmental and structural details in several levels. User can choose which of them to use, according to his needs. At Figure A.1 , we can see the class diagram of the simulator, depicting each independent module-class.

This algorithm is implemented in Python and is based on the implementation of ddssim [8], a simulator which was developed in C++ by my supervisor, Vasilis Samoladas. Below we briefly describe each module of the simulator:

- **Host**: A host can represent a single network destination (site), or a set of network destinations.
- **HostGroup**: A host group represents a broadcast address. This is simply an abstract base class. The implementation of this class can be anything. All that this class interface provides is the methods that are required by the communication traffic computation.
- **Channel**: Channels are used to collect network statistics. Each channel counts the number of messages and the total message size (in bytes). A channel is defined by the source host, the destination host and the endpoint in which is included.
- **MulticastChannel**: A channel can be a multicast channel. This is always associated with some one-way rpc method, sending data from a single source host A to

a destination host group B. Again, there are two channels associated with A and B. One channel counts the traffic sent by A, and the second channel counts the traffic received by all the hosts in host group B. For example, if there are 3 hosts in group B, and a message of 100 bytes is sent, one channel will register one additional message and 100 additional bytes, and the other will register 3 additional messages and 300 additional bytes.

- **Interface**: Represents an interface in an rpc protocol. An interface is like a 'remote type'. It represents a collection of remote functions that are implemented on a remote host.
- **Protocol**: A protocol is the collection of RPC interfaces used in a network.
- **Endpoint**: Represents an rpc endpoint. Each rpc endpoint (method) is associated with a request channel, and—if it is not one way—with a response channel.
- **Proxy**: An rpc proxy represents a proxy object for some host.When host A wants to call a remote method on host B, it makes the call through an rpc proxy method, so that the network traffic can be accounted for. Host A is the owner of the proxy and host B is the proxied host. Each proxy is associated with an rpc interface, which represents the collection of remote calls (rpc functions) being proxied. In middleware terms, the proxy instantiates the interface.
- **Sender**: Base class for implementing proxies. To define a proxy on a class, user should define this class as a subclass of Sender class.
- **Network**: A collection of hosts and channels. This class manages the network elements: hosts, groups, channels, rpc endpoints.
- **StarNetwork**: Represents a basic star network. It Creates a network with k sites and one coordinator and also does the connections. Each site can send message to coordinator and coordinator only broadcasts messages to its sites.

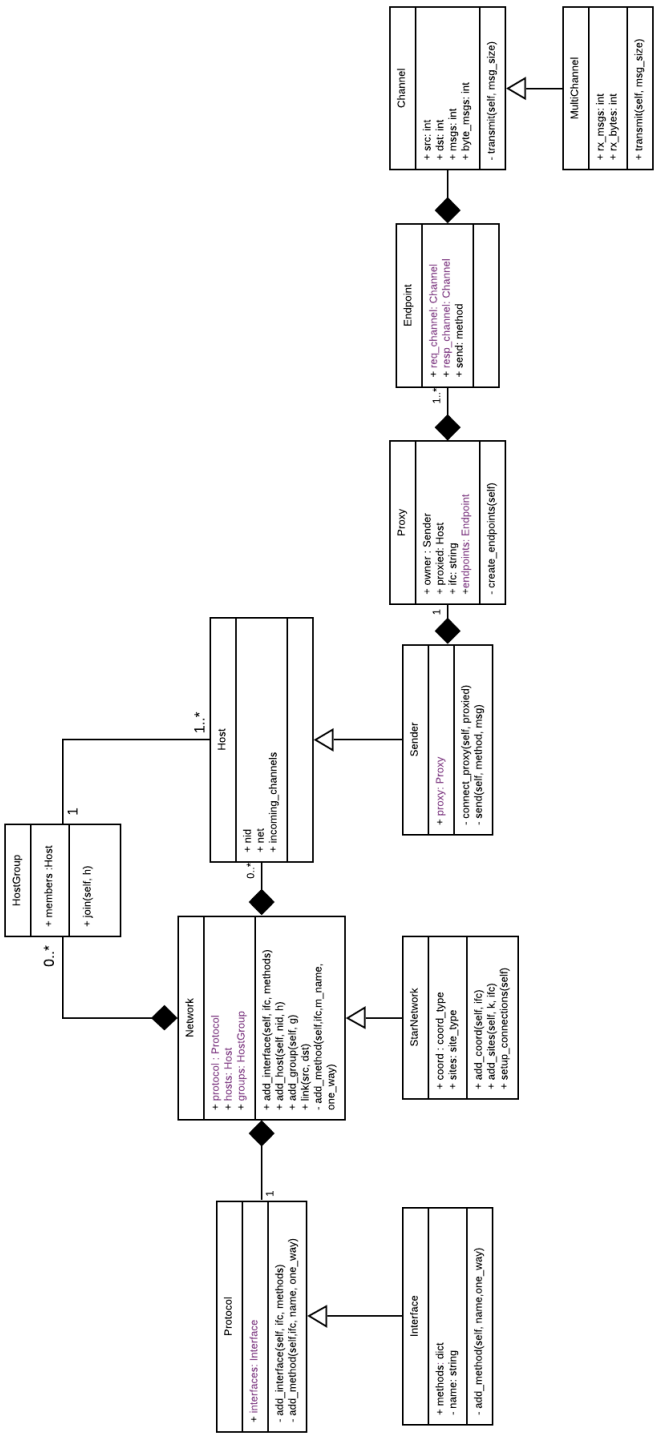


Figure A.1: The class diagram of the simulator.

Appendix B

Detailed Experimental Results

algorithm	warmup	dataset	variable	sites	features	threshold	window	accuracy	rounds	traffic
fgm	n	fixed	thres	10	10	0.05	2000	3.206238	197	2154512
fgm	n	fixed	thres	10	10	0.3	2000	3.230013	49	560016
fgm	n	fixed	thres	10	10	0.55	2000	3.258008	37	430000
fgm	n	fixed	thres	10	10	0.8	2000	3.271285	31	365392
gm	n	fixed	thres	10	10	0.05	2000	3.200209	1581	16715424
gm	n	fixed	thres	10	10	0.3	2000	3.212371	374	3969504
gm	n	fixed	thres	10	10	0.55	2000	3.244413	276	2934624
gm	n	fixed	thres	10	10	0.8	2000	3.280497	227	2417184
fgm	n	fixed	feat	10	10	0.1	2000	3.210213	112	1237364
fgm	n	fixed	feat	10	40	0.1	2000	3.395346	234	32667296
fgm	n	fixed	feat	10	70	0.1	2000	3.354168	347	1.43E+08
fgm	n	fixed	feat	10	100	0.1	2000	3.463253	456	3.78E+08
gm	n	fixed	feat	10	10	0.1	2000	3.207066	855	9048864
gm	n	fixed	feat	10	40	0.1	2000	3.257175	1836	2.53E+08
gm	n	fixed	feat	10	70	0.1	2000	3.272267	2695	1.1E+09
gm	n	fixed	feat	10	100	0.1	2000	3.29302	3611	2.98E+09
fgm	n	fixed	site	10	10	0.1	2000	3.210213	112	1237364
fgm	n	fixed	site	40	10	0.1	2000	3.202971	111	4947936
fgm	n	fixed	site	70	10	0.1	2000	3.195437	109	8530312
fgm	n	fixed	site	100	10	0.1	2000	3.190174	104	11663996
gm	n	fixed	site	10	10	0.1	2000	3.207066	855	9048864
gm	n	fixed	site	40	10	0.1	2000	3.190666	2709	1.15E+08
gm	n	fixed	site	70	10	0.1	2000	3.186374	4635	3.43E+08
gm	n	fixed	site	100	10	0.1	2000	3.185202	6530	6.9E+08
fgm	n	fixed	win	10	10	0.1	500	3.18837	176	1928177
fgm	n	fixed	win	10	10	0.1	1000	3.208957	121	1335160
fgm	n	fixed	win	10	10	0.1	1500	3.210083	113	1248168
fgm	n	fixed	win	10	10	0.1	2000	3.210213	112	1237364
gm	n	fixed	win	10	10	0.1	500	3.18741	1798	19006944
gm	n	fixed	win	10	10	0.1	1000	3.203523	1011	10696224
gm	n	fixed	win	10	10	0.1	1500	3.206092	891	9429024
gm	n	fixed	win	10	10	0.1	2000	3.207066	855	9048864

Figure B.1: Table 1 - Training via R-FGM using fixed dataset without warmup

algorithm	warmup	dataset	variable	sites	features	threshold	window	accuracy	rounds	traffic
fgm	n	drift	thres	10	10	0.1	1300	0.015811	146	1604448
fgm	n	drift	thres	10	10	0.3	1300	0.019117	63	708160
fgm	n	drift	thres	10	10	0.5	1300	0.483218	46	526028
fgm	n	drift	thres	10	10	0.7	1300	0.483218	39	449936
gm	n	drift	thres	10	10	0.1	1300	0.015752	1133	11984544
gm	n	drift	thres	10	10	0.3	1300	0.017013	473	5014944
gm	n	drift	thres	10	10	0.5	1300	0.018356	337	3578784
gm	n	drift	thres	10	10	0.7	1300	0.111043	283	3008544
fgm	n	drift	feat	10	2	0.1	1300	0.012271	113	140056
fgm	n	drift	feat	10	22	0.1	1300	0.02064	232	10414740
fgm	n	drift	feat	10	42	0.1	1300	0.020538	306	46802624
fgm	n	drift	feat	10	62	0.1	1300	0.019449	495	1.61E+08
gm	n	drift	feat	10	2	0.1	1300	0.013116	850	817824
gm	n	drift	feat	10	22	0.1	1300	0.019171	1830	80896704
gm	n	drift	feat	10	42	0.1	1300	0.020442	2483	3.76E+08
gm	n	drift	feat	10	62	0.1	1300	0.018257	4223	1.36E+09
fgm	n	drift	site	1	10	0.1	1300	0.045569	1259	1344976
fgm	n	drift	site	31	10	0.1	1300	0.07941	140	4800296
fgm	n	drift	site	61	10	0.1	1300	0.074859	137	9269592
fgm	n	drift	site	91	10	0.1	1300	0.07197	134	13562840
gm	n	drift	site	1	10	0.1	1300	0.045569	1259	1330560
gm	n	drift	site	31	10	0.1	1300	0.065016	2709	88746240
gm	n	drift	site	61	10	0.1	1300	0.07579	5047	3.25E+08
gm	n	drift	site	91	10	0.1	1300	0.072249	7165	6.89E+08
fgm	n	drift	win	10	10	0.1	500	0.023398	226	2479904
fgm	n	drift	win	10	10	0.1	1000	0.016829	153	1680808
fgm	n	drift	win	10	10	0.1	1500	0.016498	142	1560856
fgm	n	drift	win	10	10	0.1	2000	0.015207	141	1549960
gm	n	drift	win	10	10	0.1	500	0.023635	2298	24286944
gm	n	drift	win	10	10	0.1	1000	0.016493	1247	13188384
gm	n	drift	win	10	10	0.1	1500	0.015433	1097	11604384
gm	n	drift	win	10	10	0.1	2000	0.014703	1051	11118624

Figure B.2: Table 2 - Training via R-FGM using drift dataset without warmup

algorithm	warmup	dataset	variable	sites	features	threshold	window	accuracy	rounds	traffic
fgm	n	drift	thres	10	10	0.1	1300	0.015811	146	1604448
fgm	n	drift	thres	10	10	0.3	1300	0.019117	63	708160
fgm	n	drift	thres	10	10	0.5	1300	0.483218	46	526028
fgm	n	drift	thres	10	10	0.7	1300	0.483218	39	449936
gm	n	drift	thres	10	10	0.1	1300	0.015752	1133	11984544
gm	n	drift	thres	10	10	0.3	1300	0.017013	473	5014944
gm	n	drift	thres	10	10	0.5	1300	0.018356	337	3578784
gm	n	drift	thres	10	10	0.7	1300	0.111043	283	3008544
fgm	n	drift	feat	10	2	0.1	1300	0.012271	113	140056
fgm	n	drift	feat	10	22	0.1	1300	0.02064	232	10414740
fgm	n	drift	feat	10	42	0.1	1300	0.020538	306	46802624
fgm	n	drift	feat	10	62	0.1	1300	0.019449	495	1.61E+08
gm	n	drift	feat	10	2	0.1	1300	0.013116	850	817824
gm	n	drift	feat	10	22	0.1	1300	0.019171	1830	80896704
gm	n	drift	feat	10	42	0.1	1300	0.020442	2483	3.76E+08
gm	n	drift	feat	10	62	0.1	1300	0.018257	4223	1.36E+09
fgm	n	drift	site	1	10	0.1	1300	0.045569	1259	1344976
fgm	n	drift	site	31	10	0.1	1300	0.07941	140	4800296
fgm	n	drift	site	61	10	0.1	1300	0.074859	137	9269592
fgm	n	drift	site	91	10	0.1	1300	0.07197	134	13562840
gm	n	drift	site	1	10	0.1	1300	0.045569	1259	1330560
gm	n	drift	site	31	10	0.1	1300	0.065016	2709	88746240
gm	n	drift	site	61	10	0.1	1300	0.07579	5047	3.25E+08
gm	n	drift	site	91	10	0.1	1300	0.072249	7165	6.89E+08
fgm	n	drift	win	10	10	0.1	500	0.023398	226	2479904
fgm	n	drift	win	10	10	0.1	1000	0.016829	153	1680808
fgm	n	drift	win	10	10	0.1	1500	0.016498	142	1560856
fgm	n	drift	win	10	10	0.1	2000	0.015207	141	1549960
gm	n	drift	win	10	10	0.1	500	0.023635	2298	24286944
gm	n	drift	win	10	10	0.1	1000	0.016493	1247	13188384
gm	n	drift	win	10	10	0.1	1500	0.015433	1097	11604384
gm	n	drift	win	10	10	0.1	2000	0.014703	1051	11118624

Figure B.3: Table 3 - Training via R-FGM using drift dataset with warmup