Nguyen, Ngu; Sigg, Stephan

**Learning with Vertically-Partitioned Data, Binary Feedback, and Random Parameter Update**

*Published in:*
INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2019

*DOI:*
10.1109/INFCOMW.2019.8845203

Published: 01/04/2019

*Document Version*
Peer reviewed version

*Please cite the original version:*
Learning with Vertically-Partitioned Data, Binary Feedback, and Random Parameter Update

Ngu Nguyen  
Department of Communications and Networking  
Aalto University  
Espoo, Finland  
le.ngu.nguyen@aalto.fi

Stephan Sigg  
Department of Communications and Networking  
Aalto University  
Espoo, Finland  
stephan.sigg@aalto.fi

Abstract—Machine learning models can deal with data samples scattered among distributed agents, each of which holds a non-overlapping set of sample features. In this paper, we propose a training algorithm that does not require communication between these agents. A coordinator can access ground-truth labels and produces binary feedback to guide the optimization process towards optimal model parameters. We mimic the gradient descent technique with information observed locally at each agent. We experimented with the logistic regression model on multiple benchmark datasets and achieves promising results in terms of convergence rate and communication load.

Index Terms—distributed features, distributed optimization, logistic regression, gradient descent

I. INTRODUCTION

Machine learning has become one of the primary constituents to deploy intelligence into personal devices (smartphones), vehicles (smart-cars), infrastructure (smart-cities), and so on. Models of machine learning (e.g. a classifier) are constructed from training samples in order to make decisions without being explicitly instructed [1]. Usually, when training a classification model, the learner analyses its sample (i.e. an observation of the environment or an instance, represented as a feature vector) to predict the label of this observed sample. Then, the correct label is revealed to the learner as feedback so that it can iteratively modify its parameters to gradually produce better predictions. In the bandit setting (for example in the work of Kakade et al [2] and Ngo et al [3]), instead of the full information (true label), the learner receives a partial feedback, such as whether the prediction is correct or not. For instance, in a recommendation system [2], the binary feedback represents whether the user selects recommended items or not, while the user’s preference is unknown to the system. Another example is human-robot interaction context [3]. During the training process, the human shows an object or performs a gesture. Sequentially, the robot outputs a prediction (i.e. a label). Then, the human issues a feedback based on the accuracy of this predicted label (either Right or Wrong) so that the robot can improve its model.

Recently, machine learning systems are dealing with data that is captured in various locations [4]. It is not uncommon to store and process the data attributes separately in each location. This setting can be called vertically-partitioned data [5] or feature-wise data distribution [6]. One example is a sensor network in which each sensing device acquires one or some attributes of the observed environment. In some scenarios, storage, bandwidth, and energy constraints obstruct the transmission of all data to a central node. Stolpe et al [5] illustrated that with their approach, the more attributes are distributed at different nodes, the less communication load is required in comparison to transmitting all data to the central node (at least an order of magnitude). Furthermore, locally-collected data may contain private information that agents can not share with others or the coordinator. On the other hand, the coordinator may not be allowed to propagate ground-truth data. Researchers have altered machine learning algorithms to handle the aforementioned challenges. However, their work relies on either communication between agents [6] or complex feedback from the coordinator [5].

In this paper, we consider a setting in which entries of feature vectors are vertically partitioned over distributed agents. Suppose each sample is a vector that has $K$ attributes (features) $\mathbf{x}_i = [x_1, x_2, \ldots, x_K]^T$. There exists a parameter vector $\mathbf{w}_i = [w_1, w_2, \ldots, w_K]^T$ which would be combined with the feature vector $\mathbf{x}$ in a machine learning model (e.g. a classifier). Each agent holds a partition $P$ of sample attributes

We appreciate partial funding in the frame of EIT Digital HII Active project and the Academy of Finland.
Section IV-B).

optimal model (see Section IV-A) and more efficient than to show that our method is possible to converge towards an
improves the learned model. We apply our proposed approach
to control the optimization process. This protocol iteratively
The latter then analyses information from all agents to evaluate
illustrates the protocol in which an agent transmits the combi-
sequentially-selected actions iteratively optimize the model
feedback, the agents choose their corresponding actions. These
parametrizes one scalar value \( w_j \) instead of multiple values
x_{\mathcal{P}} \in \{ x_j \in \mathcal{P} \} \). The corresponding labels \( y_i \) are located at
at each step of the proposed optimization process,
a random value is generated and leveraged to calculate the
parameter updates with local information at each agent. More
specifically, at each step of the proposed optimization process,
a random value is generated and leveraged to calculate the
update step. Then, the new parameter is combined with new
parameters updates increase or decrease the performance of
the classifier under training. Depending on values of the binary
feedback, the agents choose their corresponding actions. These
sequentially-selected actions iteratively optimize the model
parameters without requiring communication between agents,
except limited data exchange with the coordinator. Figure 2
illustrates the protocol in which an agent transmits the combi-
nation of its parameter and local attribute to the coordinator.
The latter then analyses information from all agents to evaluate
the model quality and provides a correspondent feedback
to control the optimization process. This protocol iteratively
improves the learned model. We apply our proposed approach
to train logistic regression models [1] on benchmark datasets.
We compare our preliminary results with other approaches
to show that our method is possible to converge towards an
optimal model (see Section IV-A) and more efficient than other approaches with regard to the communication load (see Section IV-B).

II. RELATED WORK

To train a machine learning model over data partitioned
agents (e.g. [5]) or facilitating information exchange between them [6]. Stolpe et al [5] proposed to use 1-class support vector machine in anomaly detection with low communication cost. Their technique, namely Vertically Distributed Core Vector Machine (VDCVM), requires each data agent to transmit a single numerical value instead of all attributes to the coordinator. The coordinator is able to send sample indices and parameter updates to the agents. The feedback mechanism in this method is more complex than ours, which requires only binary feedback. Ying et al [6] investigates the problem of learning a model from both large datasets and large-dimensional feature space scenarios. Each networked agent holds its own set of features. Instead of relying on a coordinator, they perform a consensus iteration with their neighbours to agree on model updates. Hence, this approach utilized more communication load between neighbouring nodes. In our proposed method, the nodes updates their parameters using random values and only local information.

With regard to random updates, Sigg [7] proposed a similar local search technique with binary feedback that was implemented in distributed adaptive transmit beam-forming. The technique randomly alters phase-frequency combination to achieve synchronization of carriers. However, the search space of this problem is different with ours. In our setting, each parameter of a machine learning model can be updated with various step-size whose domain is \( \mathbb{R} \); hence, the search space is much larger.

Our work is also related to the recent meta-learning paradigm that considers optimization as a learning problem. Rather than using generic and hand-engineered optimizers such as gradient descent, the update rule can be learned through exploiting problem structures. Andrychowicz et al [8] formulates the update rule with a recurrent neural network (RNN). Their approach outperformed generic optimizers when applied in the problems that it had been trained on. Beyond the update rule, Chen et al [9] aims to generate algorithms for global black-box optimization. They utilized a trained RNN to explore and exploit the domain of objective functions. Even though these techniques [8], [9] are superior to standard optimizers, they themselves require a training process. Furthermore, their applicability on vertically-partitioned datasets has not been studied so far.

III. METHODOLOGY

Optimization techniques are the central topics in machine learning [10]. During the model training process, they aim to find the optimal solution \( \theta \in \mathbb{R}^d \) that optimizes an objective function \( f(\theta) \). One of the widely-used approaches for differentiable functions (e.g. gradient descent algorithm [1]) is to iteratively update the current solution \( \theta \) with a step vector \( \Delta \theta \):

\[
\theta_i = \theta_{i-1} + \Delta \theta \quad (1)
\]

The step vector \( \Delta \theta \) is computed with a function \( \pi \) using information from the objective function and previous updates.
For example, in case of the gradient descent method, \( \pi = -\gamma \nabla f(\theta) \).

We focus on logistic regression because it has been widely used in applied sciences [11]. We focus on the binary classification case in which there are two classes. To handle multi-class problems, we can apply multinomial logistic regression cases [12] or one-vs-all scheme that combines binary classifiers [13]. In the binary classification case, where \( x_i \in \mathbb{R}^d \) is a feature vector (or sample), \( y_i \in \{0, 1\} \) is the corresponding label, and \( w \in \mathbb{R}^d \) is the parameter vector (i.e. \( \theta \)) that is obtained during the training process, logistic regression models a probability distribution \( p(y_i|x_i;w) \):

\[
h(x_i) = \frac{1}{1 + e^{-w^T x_i}},
\]

\[
p(y_i = 1|x_i; w) = h(x_i)
\]

\[
p(y_i = 0|x_i; w) = 1 - h(x_i)
\]

Applying gradient descent technique [1], we aim to gradually optimize the log likelihood function \( l(w) = \log L(w) \), where \( L(w) \) is the loss function, over \( n \) training samples:

\[
l(w) = \log L(w) = \sum_{i=1}^{m} y_i \log h(x) + (1 - y_i) \log (1 - h(x))
\]

The stochastic gradient descent algorithm gradually updates each parameter \( w_i \) toward the optimal value:

\[
w^j = w^j + \lambda \frac{\partial}{\partial w_i} l(w) = w^j + \lambda (y_i - h(x_i))x_i^T
\]

where \( \lambda \) is the learning rate, which controls the update speed of the model parameters \( w \). If each parameter can be updated separately, based on all \( n \) training samples, the method becomes coordinate descent (or ascent) [14]:

\[
w^j = w^j + \lambda \sum_{i=1}^{n} (y_i - h(x_i))x_i^T
\]

In order to update the parameters \( w \) in Equation 5, the agents require \( y_i - h(x_i) \). However, in our setting, they can not access the ground-truth label \( y_i \) nor data from other agents. Hence, to mimic the update, we generate a uniformly-random value \( r \in [-1, 1] \) since \( h(x_i) \in [0, 1] \) and \( y_i \in \{0, 1\} \). \( r \) can affect both amount and direction of updates. Equation 5 becomes:

\[
w^j = w^j + \lambda \sum_{i=1}^{n} rx_i^T
\]

Note that the learning rate \( \lambda \) and the local attribute of each sample \( x_i \) are accessible locally at each agent. Since \( r \) is generated randomly, it can reduce the performance of our classification model, i.e. it may change the parameters to wrong directions far from the optimal values. We, thus, need a mechanism to correct the malicious update. A binary feedback can be applied to handle this requirement. After receiving all inputs \( w_i^j, x_i \) from the agents, the coordinator can compute the loss function in Equation 3. Based on the loss value, it can roughly evaluate the current learned model. To deal with the variance of loss values, we assess the model after collecting a number of processed feature vectors \( w^T x_i \). The idea is based on the mini-batch stochastic gradient descent [10]. \( n \) in Equation 6 is the size of each mini-batch of samples.

The agent state transition is visualized in Figure 3. It reflects how agents select suitable actions depending on their states. When receiving a feedback notifying that the change does not reduce the loss function, a \textit{reversible} agent roll-backs the update, i.e. its parameter is reversed to the previous value. It becomes a \textit{waiting} agent. This mechanism prevents many agents from updating at the same time because the updated parameters are more likely to move further from the optimal ones when there are many parallel changes. Algorithm 1 summarizes the state transition of each agent according to the feedback it receives. Line 4 - 6 add an update that combines a random quantity and a local feature to the current parameter; then changes the agent state to \textit{reversible}. These steps can be customized by putting a probability of updating \( P \) at each agent, which can control the randomness in convergence. Line 8 - 10 reverses the update if it does not improve the model quality. The remaining case transforms the agent from \textit{waiting} to \textit{ready-to-update} state. Algorithm 2 describes how the coordinator generates feedback values when receiving processed data from the agents. Note that to simplify the notation, these algorithms are applied on a single sample. It calculates the loss value at Line 1 before detecting the loss trend. Then, the feedback is generated from Line 3 to Line 7. In the case of mini-batch updates, the update part and the function \textit{IsLossDecreasing} at Line 2 should be customized appropriately.

IV. EXPERIMENTS

A. On Convergence

To ensure that our algorithm can converge, we performed experiments on two public datasets: KDD99 [15] and PhishingWebsites [16]. We used 10% of the former, which contain 489843 samples of 41 features. This dataset was generated from raw TCP dump data during the 1998 DARPA Intrusion
Algorithm 1: Update rule for each agent

Data: A binary feedback value $b \in \{0, 1\}$
Result: Parameter $w_j$ of the agent $j$ and its state $s_j$

1 if $b = 0$ then
   // Previous update does not improve the model
   begin
      switch $s_j$ do
         case 0 do
            // Possible to update with a probability of $P$
            $w_j = w_j + rx_i^j$
            $s_j = 1$
         end
         case 1 do
            // Replacing the current parameter with its previous value
            $w_j = \text{Reverse}(j)$
            $s_j = 2$ // Waiting state
         end
         otherwise do
            $s_j = 0$
         end
   end
else
   if $s_j = 0$ then
      // Possible to update with a probability of $P$
      $w_j = w_j + rx_i^j$ $s_j = 1$
   else
      $s_j = 0$
   end
end

Algorithm 2: Feedback generation at the coordinator

Data: A set of values $w_j, x_i^j$ from each agent $j$ over the sample $x_i$
Result: A binary feedback value $b \in \{0, 1\}$

1 $l(w) = y_i \log(h(x_i)) + (1 - y_i) \log(1 - h(x_i))$
2 $t = \text{IsLossDecreasing}(l(w))$
3 if $t = \text{True}$ then
   $b = 1$
else
   $b = 0$
end

Fig. 4. Comparison of convergence rate between our approach and the centralized algorithm on KDD99 dataset [15]

Detection Evaluation Program, prepared and managed by MIT Lincoln Labs [15]. The latter has 11055 samples of 30 features and is analysed for predicting phishing websites [16]. In these scenarios, we assume that the coordinator can obtain the labels (ground-truth data) via expert knowledge or user’s reports.

With both datasets, we randomly split 75% for training and 25% for testing. We configured the mini-batch size in our algorithm to 10000 and 500 for the first and second dataset, respectively. Training of the KDD99 dataset [15] employed 100 epochs (iterations) while the number of epochs used for the PhishingWebsites dataset [16] is 500. We configured the probability of updating $P = 0.5$ for all agents. On the other hand, for comparison, we used scikit-learn 0.20.1 library to implement the centralized classifier trained with stochastic gradient descent. In this setting, all data must be transmitted to the central node before training. The mini-batch size and number of epochs were the same for the centralized algorithm. We carried out the experiments on a desktop computer (Intel Core i5 1.8GHz, 8GB RAM). For both our classifier and the centralized algorithm, we set the learning rate $\lambda = 0.001$.

Since our technique relies on a random number generator to compute the parameter update, we repeat the experiment ten times for each dataset. We plot the average (mean) loss values over all epochs in ten trials, along with their confidence interval. Figure 4 and Figure 5 show the trend of loss functions in KDD99 and PhishingWebsites, respectively. The figures show that our proposed algorithm can converge to an optimal solution. Figure 6 illustrates how the parameters are stable at the last epochs: they are updated but then reverse to previous values when receiving a negative feedback. We plan to integrate $L_1$- and $L_2$-regularization terms into our formulation as future work.

B. On Amount of Transmitted Data

In this section, we compare the exchanged data amount of our method and others. Ying et al [6] proposed

\[ \text{https://scikit-learn.org/stable/} \]
Pipelined variance-reduced dynamic diffusion learning algorithm (PVRD$^2$) to train a classifier over a vertically-partitioned dataset in which networked agents hold their own feature sets. The algorithm employed a consensus strategy to agree on the update. During one iteration, each agent communicates a vector of $J \times C \times B$ values, where $J$ is the number of iteration in each consensus step, $C$ is the number of classes, and $B$ is the mini-batch size [6]. In our algorithm, during one iteration, each agent transmits $B$ scalar values and the coordinator broadcasts 1 binary feedback. In our comparison on the KDD99 dataset [15], $J = 1$, $C = 2$, and we varied $B$. It has been used to as a standard benchmark dataset for classification algorithms, for example in [5] and [11]. Note that we aim to classify normal and abnormal cases (attacking), hence $C = 2$. There are 41 agents, corresponding to the number of features in the dataset. Figure 7 displays the length of exchanged data of our algorithm and PVRD$^2$ [6], which shows that ours is more communication-efficient with less values to be transmitted in one iteration.

Although the 1-class anomaly detection scenario of VD-CVM [5] is different to our focus in this paper, we provide an estimate number of transferred bytes. In one iteration, the number of bytes that VDCVM communicates is:

$$B_{VDCVM} = s \times 4 + s \times k \times 8 + 4 + m \times 8 + 8$$

(7)

where $s = 59$ is the size of a randomly-selected subset to find the furthest data point for the algorithm, $k$ is the number of attributes, and $m$ is the number of nodes (or agents). Two constants 4 and 8 are the size in bytes of an integer and real number. The detailed explanation of Equation 7 can be found in [5]. Within this context, $k = m$ and $s$ can be roughly translated to the mini-batch size in our algorithm. Hence:

$$B_{Ours} = s \times k \times 8 + 1$$

(8)

The last term of Equation 8 is the coordinator binary feedback which one byte is redundant to contain. From the equations, our method sends significantly less bytes than the other.

V. Conclusion

In this paper, we investigate the problem of training a machine learning model over distributed data. In some scenarios, data is partitioned among multiple sites each of which holds a set of features. One example is a monitoring system comprised of networked agents and each observes some attributes of the environment. Due to resource constraints and privacy, the communication between these agents is limited or event prohibited. There exists a coordinator that can access the ground-truth data (i.e. labels) and guide the model training. We propose an approach that leverage randomly-generated values and local information to collaboratively optimize the parameters of a classification model. Our method does not require the communication between the agents. Furthermore, the coordinator needs to provide only simple feedback to control the optimization process. We performed experiments on a widely-used model, logistic regression, over popular benchmark datasets Our approach utilized less communication load while showing a promising convergence rate. In future, we plan to integrate regularizations into our formulation and prove its convergence with mathematical analysis.
REFERENCES


