Efficient P2P Ensemble Learning with Linear Models on Fully Distributed Data

Róbert Ormándi, István Hegedűs
University of Szeged
Szeged, Hungary
{ormandi,ihegedus}@inf.u-szeged.hu

Márk Jelasity
University of Szeged and Hungarian Academy of Sciences
Szeged, Hungary
jelasity@inf.u-szeged.hu

Abstract—Machine learning over fully distributed data poses an important problem in peer-to-peer (P2P) applications. In this model we have one data record at each network node, but without the possibility to move raw data due to privacy considerations. For example, user profiles, ratings, history, or sensor readings can represent this case. This problem is difficult, because there is no possibility to learn local models, yet the communication cost needs to be kept low. Here we propose gossip learning, a generic approach that is based on multiple models taking random walks over the network in parallel, while applying an online learning algorithm to improve themselves, and getting combined via ensemble learning methods. We present an instantiation of this approach for the case of classification with linear models. Our main contribution is an ensemble learning method which—through the continuous combination of the models in the network—implements a virtual weighted voting mechanism over an exponential number of models at practically no extra cost as compared to independent random walks. Our experimental analysis demonstrates the performance and robustness of the proposed approach.

Index Terms—P2P; bagging; online learning; stochastic gradient descent; random walk

I. INTRODUCTION

The main attraction of peer-to-peer (P2P) technology for distributed applications and systems is acceptable scalability at a low cost (no central servers are needed) and a potential for privacy preserving solutions, where data never leaves the computer of a user in a raw form. The label P2P covers a wide range of distributed algorithms that follow a specific system model, in which there are only minimal assumptions about the reliability of communication and the network components. A typical P2P system consists of a very large number of nodes (peers) that communicate via message passing. Messages can be delayed or lost, and peers can join and leave the system at any time.

In recent years, there has been an increasing effort to develop collaborative machine learning algorithms that can be applied in P2P networks. This was motivated by the various potential applications such as spam filtering, user profile analysis, recommender systems and ranking. An important special case of P2P data processing is fully distributed data, where each node holds only one data record containing personal data, preferences, ratings, history, local sensor readings, and so on. Often, these personal data records are the most sensitive ones, so it is essential that we process them locally. At the same time, the learning algorithm has to be fully distributed, since the usual approach of building local models and combining them is not applicable.

Our goal here is to present algorithms for the case of fully distributed data. The design requirements specific to the P2P aspect are the following. First, the algorithm has to be extremely robust. Even in extreme failure scenarios it should maintain a reasonable performance. Second, prediction should be possible at any time in a local manner; that is, all nodes should be able to perform high quality prediction immediately without any extra communication. Third, the algorithm has to have a low communication complexity; both in terms of the number of messages sent, and the size of these messages as well. Privacy preservation is also one of our main goals, although in this study we do not analyze this aspect explicitly.

The gossip learning approach we propose involves models that perform a random walk in the P2P network, and that are updated each time they visit a node, using the local data record. There are as many models in the network as the number of nodes. Any online algorithm can be applied as a learning algorithm that is capable of updating models using a continuous stream of examples. Since models perform random walks, all nodes will experience a continuous stream of models passing through them. Apart from using these models for prediction directly, nodes can also combine them in various ways using ensemble learning.

The generic skeleton of gossip learning involves three main components: an implementation of random walk, an online learning algorithm, and ensemble learning. In this paper we focus on an instantiation of gossip learning, where the online learning method is a stochastic gradient descent for linear models. In addition, nodes do not simply update and then pass on models during the random walk, but they also combine these models in the process. This implements a distributed “virtual” ensemble learning method similar to bagging, in which we in effect calculate a weighted voting over an exponentially increasing number of linear models.

M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences. This work was partially supported by the Future and Emerging Technologies programme FP7-COSI-ICT of the European Commission through project QLectives (grant no.: 231200).
Our specific contributions include the following: (1) we propose gossip learning, a novel and generic approach for P2P learning on fully distributed data, which can be instantiated in various different ways; (2) we introduce a novel, efficient distributed ensemble learning method for linear models that virtually combines an exponentially increasing number of linear models; and (3) we provide a theoretical and empirical analysis of the convergence properties of the method in various scenarios.

The outline of the paper is as follows. Section II elaborates on the fully distributed data model. Section III summarizes related work and the background concepts. In Section IV we describe our generic approach and a naive algorithm as an example. Section V presents the core algorithmic contributions of the paper along with a theoretical discussion, while Section VI contains an experimental analysis. Section VII concludes the paper.

II. FULLY DISTRIBUTED DATA

Our focus is on fully distributed data, where each node in the network has a single feature vector, that cannot be moved to a server or to other nodes. Since this model is not usual in the data mining community, we elaborate on the motivation and the implications of the model.

In the distributed computing literature the fully distributed data model is typical. In the past decade, several algorithms have been proposed to calculate distributed aggregation queries over fully distributed data, such as the average, the maximum, and the network size (e.g., [1]–[3]). Here, the motivation for not moving any raw data is mainly to achieve robustness and adaptivity through not relying on any central servers. In some systems, like in sensor networks or mobile ad hoc networks, the physical constraints on communication also prevent the collection of the data.

An additional motivation for not moving data is privacy preservation, where local data is not revealed in its raw form, even if the computing infrastructure made it possible. This is especially important in P2P social networking [4], where the key motivation is giving the user full control over personal data. Clearly, in P2P social networks, there is a need for more complex aggregation queries, and ultimately, for data models, to support features such as recommendations and spam filtering, and to make the system more robust with the help of, for example, distributed intruder detection. In other fully distributed systems data models are also important for monitoring and control.

Motivated by the emerging need for building complex data models over fully distributed data in different systems, we work with the abstraction of fully distributed data, and we aim at proposing generic algorithms that are applicable in all compatible systems.

In the fully distributed model, the requirements of an algorithm also differ from those of parallel data mining algorithms, and even from previous work on P2P data mining. Here, the decisive factor is the cost of message passing. Besides, the number of messages each node is allowed to send in a given time window is limited, so computation that is performed locally has a cost that is typically negligible when compared to communication delays. For this reason prediction performance has to be investigated as a function of the number of messages sent, as opposed to wall clock time. Since communication is crucially important, evaluating robustness to communication failures, such as message delay and message loss, also gets a large emphasis.

The approach we present here is applicable successfully also when each node stores many records (and not only one); but its advantages to known approaches to P2P data mining become less significant, since communication plays a smaller role when local data is already usable to build reasonably good models. In the following we focus on the fully distributed model.

III. BACKGROUND AND RELATED WORK

We organize the discussion of the background of our work along the generic model components outlined in the Introduction and explained in Section V online learning, ensemble learning, and peer sampling. We also discuss related work in P2P data mining.

a) Online Learning.: Gossip learning relies on algorithms that iterate over available training data, or process a continuous stream of data records, and evolve a model by updating it for each individual data record according to some update rule. Ma et al. provide a nice summary of such algorithms for large scale data [5]. Among these methods, we focus on stochastic gradient descent; a very simple algorithm that has been shown to be very well suited for the large scale data mining scenarios we are interested in [6, 7]. In this paper, we use the Pegasos algorithm, which is a gradient method for linear SVM learning [8].

b) Ensemble Learning.: Most distributed large scale algorithms apply some form of ensemble learning to combine models learned over different samples of the training data. Rokach presents a survey of ensemble learning methods [9]. We apply a method for combining the models in the network that is related to both bagging [10] and “pasting small votes” [11]: when the models start their random walk, initially they are based on non-overlapping small subsets of the training data due to the large scale of the system (the key idea behind pasting small votes) and as time goes by, the sample sets grow, approaching the case of bagging (although the samples that belong to different models will not be completely independent in our case).

c) Peer Sampling in Distributed Systems.: The sampling probability for each data record is defined by peer sampling algorithms that are used to implement the random walk. Here we apply uniform sampling. A set of approaches to implement uniform sampling in a P2P network apply random walks themselves over a fixed overlay network, in such a way that the corresponding Markov-chain has a uniform limiting distribution [12–14]. In our algorithm, we apply gossip-based
peer sampling [13] where peers periodically exchange small random subsets of addresses, thereby providing a local random sample of the addresses at each point in time at each node. The advantage of gossip-based sampling in our setting is that samples are available locally and without delay. Furthermore, the messages related to the peer sampling algorithm can piggyback the random walks of the models, thereby avoiding any overheads in terms of message complexity.

d) P2P Learning. In the area of P2P computing, a large number of fully distributed algorithms are known for calculating global functions over fully distributed data, generally referred to as aggregation algorithms. We mention only two examples: Astrolabe [11] and gossip-based averaging [2]. These algorithms are simple and robust, but are capable of calculating only simple functions such as the average. Nevertheless, these simple functions can serve as key components for more sophisticated methods, such as the EM algorithm [16]. However, here we seek to provide a rather generic approach that covers a wide range of machine learning models, while maintaining a similar robustness and simplicity.

In the past few years there has been an increasing number of proposals for P2P machine learning algorithms as well, like those in [17]–[22]. The usual assumption in these studies is that a peer has a subset of the training data on which a model can be learned locally. After learning the local models, algorithms either aggregate the models to allow each peer to perform local prediction, or they assume that prediction is performed in a distributed way. Clearly, distributed prediction is a lot more expensive than local prediction; however, model aggregation is not needed, and there is more flexibility in the case of changing data. In our approach we adopt the fully distributed model, where each node holds only one data record. In this case we cannot talk about local learning: every aspect of the learning algorithm is inherently distributed. Since we assume that data cannot be moved, the models need to visit data instead. In a setting like this, the main problem we need to solve is to efficiently aggregate the various models that evolve slowly in the system so as to speed up the convergence of prediction performance.

IV. GOSSIP LEARNING: THE BASIC IDEA

Algorithm [1] provides the skeleton of the gossip learning framework. The same algorithm is run at each node in the network. The algorithm consists of an active loop of periodic activity, and a method to handle incoming models. Incoming models are simply put in a FIFO queue. In the present discussion, we need at most two models locally, so the maximal size of the queue is 2. Note, that if more models are stored, then additional techniques such as local voting can be applied, which we do not discuss here for simplicity. New elements are always added, even if the size is already 2, in which case the oldest element is removed.

We make no assumptions about either the synchrony of the loops at the different nodes or the reliability of the messages. We do assume that the period of the loop $\Delta$ is the same at all nodes. For simplicity, here we assume that the active loop is initiated at the same time at all the nodes, and we do not consider any stopping criteria, so the loop runs indefinitely. The assumption about the synchronized start allows us to focus on the convergence properties of the algorithm, but it is not a crucial requirement in practical applications. In fact, randomly restarted loops actually help in following drifting concepts and changing data, which is the subject of our ongoing work.

The algorithm contains abstract methods that can be implemented in different ways to obtain a concrete learning algorithm. The main placeholders are SELECTPEER() and CREATEMODEL(). Method SELECTPEER() is the interface for the peer sampling service, as described in Section III. Here we use the NEWSCAST algorithm [13], which is a gossip-based implementation of peer sampling. We do not discuss NEWSCAST here in detail, all we assume is that SELECTPEER() provides a uniform random sample of the peers without creating any extra messages in the network, given that NEWSCAST gossip messages (that contain only a few dozen network addresses) can piggyback gossip learning messages.

The core of the approach is CREATEMODEL(). Its task is to create a new updated model based on locally available information—the models contained in the queue, and the local single training data record—to be sent on to a random peer. Algorithm 2 lists three implementations that are still abstract. They represent those three possible ways of breaking down the task that we will study in this paper.

We assume that the queue is always full, that is, it contains 2 elements. We initialize the queue to contain two null vectors. The abstract method UPDATE represents the online learning algorithm—the second main component of our framework besides peer sampling—that updates the model based on one example (the local example of the node). Procedure CREATEMODEL.RW implements the case where models independently perform random walks over the network. We will use this algorithm as a baseline.

The remaining two variants apply a method called MERGE, either before the update (MU) or after it (UM). Method MERGE helps implement the third component: ensemble learn-
A completely impractical example for an implementation of **MERGE** is the case where the model space consists of all the sets of basic models of a certain type. Then **MERGE** can simply merge the two input sets, **UPDATE** can update all the models in the set, and prediction can be implemented via, for example, majority voting (for classification) or averaging the predictions (for regression). With this implementation, all nodes would collect an exponentially increasing set of models, allowing for a much better prediction after a much shorter learning time in general than based on a single model \([10]\), although the learning history for the members of the set would not be completely independent.

This implementation is of course impractical because the size of the messages in each cycle of the main loop would increase exponentially. Our main contribution is to discuss and analyze a special case: linear models. For linear models we will propose an algorithm where the message size can be kept constant, while producing the same (or similar) behavior as the impractical implementation above. The subtle difference between the MU and UM versions will also be discussed.

Let us close this section with a brief analysis of the cost of the algorithm in terms of computation and communication. As of communication: each node in the network sends exactly one message in each \(\Delta\) time units. The size of the message depends on the selected hypothesis space; normally it contains the parameters of a single model. In addition, the message also contains a small constant number of network addresses as defined by the NewsCAST protocol (typically around 20). The computational cost is one or two update steps in each \(\Delta\) time units for the UM or the MU variants, respectively. The exact cost of this step depends on the selected online learner.

V. **MERGING LINEAR MODELS THROUGH AVERAGING**

The key observation we make is that in a linear hypothesis space, in certain cases voting-based prediction is equivalent to a single prediction by the average of the models that participate in the voting. Furthermore, updating a set of linear models and then averaging them is sometimes equivalent to averaging the models first, and then updating the resulting single model. These observations are valid in a strict sense only in special circumstances. However, our intuition is that even if this key observation holds only in a heuristic sense, it still provides a valid heuristic explanation of the behavior of the resulting averaging-based merging approach.

In the following we first give an example of a case where there is a strict equivalence of averaging and voting to illustrate the concept, and subsequently we discuss and analyze a practical and competitive algorithm, where the correspondence of voting and averaging is only heuristic in nature.

A. The Adaline Perceptron

We consider here the Adaline perceptron \([23]\) that arguably has one of the simplest update rules due to its linear activation function. Without loss of generality, we ignore the bias term. The error function to be optimized is defined as

\[
E_x(w) = \frac{1}{2}(y - \langle w, x \rangle)^2
\]

where \(w\) is the linear model, and \((x, y)\) is a training example \((x, w) \in \mathbb{R}^n, y \in \{-1, 1\}\). The gradient at \(w\) for \(x\) is given by

\[
\nabla_w x = \frac{\partial E_x(w)}{\partial w} = -(y - \langle w, x \rangle)x
\]

that defines the learning rule for \((x, y)\) by

\[
w^{(k+1)} = w^{(k)} + \eta(y - \langle w^{(k)}, x \rangle)x,
\]

where \(\eta\) is the learning rate. In this case it is a constant.

Now, let us assume that we are given a set of models \(w_1, \ldots, w_m\), and let us define \(\bar{w} = (w_1 + \ldots + w_m)/m\). In the case of a regression problem, the prediction for a given point \(x\) and model \(w = \langle w, x \rangle\). It is not hard to see that

\[
h(x) = \langle \bar{w}, x \rangle = \frac{1}{m}(\sum_{i=0}^{m} w_i, x) = \frac{1}{m} \sum_{i=0}^{m} \langle w_i, x \rangle,
\]

which means that the voting-based prediction is equivalent to prediction based on the average model.

In the case of classification, the equivalence does not hold for all voting mechanisms. But it is easy to verify that in the case of a weighted voting approach, where vote weights are given by \(\langle w, x \rangle\), and the votes themselves are given by \(\text{sgn}(w, x)\), the same equivalence holds:

\[
h(x) = \text{sgn}(\frac{1}{m} \sum_{i=1}^{m} \langle w, x \rangle \text{sgn}(w, x)) = \text{sgn}(\frac{1}{m} \sum_{i=1}^{m} \langle w_i, x \rangle) = \text{sgn}(\bar{w}, x).
\]

---

**Algorithm 2** CREATEMODEL(): three implementations

1: **procedure** CREATEMODELRW  
2: \(m \leftarrow \text{modelQueue.first()}\)  
3: update\((m)\)  
4: return \(m\)  
5: **end procedure**

6:    
7: **procedure** CREATEMODELMU  
8: \(m_1 \leftarrow \text{modelQueue.first()}\)  
9: \(m_2 \leftarrow \text{modelQueue.second()}\)  
10: \(m \leftarrow \text{merge}(m_1, m_2)\)  
11: update\((m)\)  
12: return \(m\)  
13: **end procedure**

14:    
15: **procedure** CREATEMODELUM  
16: \(m_1 \leftarrow \text{modelQueue.first()}\)  
17: update\((m_1)\)  
18: \(m_2 \leftarrow \text{modelQueue.second()}\)  
19: update\((m_2)\)  
20: \(m \leftarrow \text{merge}(m_1, m_2)\)  
21: return \(m\)  
22: **end procedure**

---

The key observation we make is that in a linear hypothesis space, in certain cases voting-based prediction is equivalent to a single prediction by the average of the models that participate in the voting. Furthermore, updating a set of linear models and then averaging them is sometimes equivalent to averaging the models first, and then updating the resulting single model. These observations are valid in a strict sense only in special circumstances. However, our intuition is that even if this key observation holds only in a heuristic sense, it still provides a valid heuristic explanation of the behavior of the resulting averaging-based merging approach.

In the following we first give an example of a case where there is a strict equivalence of averaging and voting to illustrate the concept, and subsequently we discuss and analyze a practical and competitive algorithm, where the correspondence of voting and averaging is only heuristic in nature.

A. The Adaline Perceptron

We consider here the Adaline perceptron \([23]\) that arguably has one of the simplest update rules due to its linear activation function. Without loss of generality, we ignore the bias term. The error function to be optimized is defined as

\[
E_x(w) = \frac{1}{2}(y - \langle w, x \rangle)^2
\]

where \(w\) is the linear model, and \((x, y)\) is a training example \((x, w) \in \mathbb{R}^n, y \in \{-1, 1\}\). The gradient at \(w\) for \(x\) is given by

\[
\nabla_w x = \frac{\partial E_x(w)}{\partial w} = -(y - \langle w, x \rangle)x
\]

that defines the learning rule for \((x, y)\) by

\[
w^{(k+1)} = w^{(k)} + \eta(y - \langle w^{(k)}, x \rangle)x,
\]

where \(\eta\) is the learning rate. In this case it is a constant.

Now, let us assume that we are given a set of models \(w_1, \ldots, w_m\), and let us define \(\bar{w} = (w_1 + \ldots + w_m)/m\). In the case of a regression problem, the prediction for a given point \(x\) and model \(w = \langle w, x \rangle\). It is not hard to see that

\[
h(x) = \langle \bar{w}, x \rangle = \frac{1}{m}(\sum_{i=0}^{m} w_i, x) = \frac{1}{m} \sum_{i=0}^{m} \langle w_i, x \rangle,
\]

which means that the voting-based prediction is equivalent to prediction based on the average model.

In the case of classification, the equivalence does not hold for all voting mechanisms. But it is easy to verify that in the case of a weighted voting approach, where vote weights are given by \(\langle w, x \rangle\), and the votes themselves are given by \(\text{sgn}(w, x)\), the same equivalence holds:

\[
h(x) = \text{sgn}(\frac{1}{m} \sum_{i=1}^{m} \langle w, x \rangle \text{sgn}(w, x)) = \text{sgn}(\frac{1}{m} \sum_{i=1}^{m} \langle w_i, x \rangle) = \text{sgn}(\bar{w}, x).
\]
A similar approach to this weighted voting mechanism has been shown to improve the performance of simple vote counting \[24\]. Our preliminary experiments also support this.

In a very similar manner, it can be shown that updating \(\bar{w}\) using an example \((x, y)\) is equivalent to updating all the individual models \(w_1, \ldots, w_m\) and then taking the average:

\[
\bar{w} + \eta(y - \langle \bar{w}, x \rangle)x = \frac{1}{m} \sum_{i=1}^{m} w_i + \eta(y - \langle w_i, x \rangle)x.
\]

The above properties lead to a rather important observation. If we implement our gossip learning skeleton using Adaline, as shown in Algorithm 3 then the resulting algorithm behaves exactly as if all the models were simply stored and then forwarded, resulting in an exponentially increasing number of models contained in each message, as described in Section VI That is, averaging effectively reduces the exponential message complexity to transmitting a single model in each cycle independent of time, yet we enjoy the benefits of the aggressive, but impractical approach of simply replicating all the models and using voting over them for prediction.

It should be mentioned that—even though the number of "virtual" models is growing exponentially fast—the algorithm is not equivalent to bagging over an exponential number of independent models. In each gossip cycle, there are only \(N\) independent updates occurring in the system overall (where \(N\) is the number of nodes), and the effect of these updates is being aggregated rather efficiently. In fact, as we will see in Section VI bagging over \(N\) independent models actually outperforms the gossip learning algorithms.

### B. Pegasos

Here we discuss the adaptation of Pegasos (a linear SVM gradient method \[8\] used for classification) into our gossip framework. The components required for the adaptation are shown in Algorithm 3 where method UPDATEPEGASOS is simply taken from \[8\]. For a complete implementation of the framework, one also needs to select an implementation of CREATEMODEL from Algorithm 2. In the following, the three versions of a complete Pegasos-based implementation defined by these options will be referred to as P2PEGASOSRW, P2PEGASOSMU, and P2PEGASOSUM.

The main difference between the Adaline perceptron and Pegasos is the context dependent update rule that is different for correctly and incorrectly classified examples. Due to this difference, there is no strict equivalence between averaging and voting, as in the case of the previous section. To see this, consider two models, \(w_1\) and \(w_2\), and an example \((x, y)\), and let \(\bar{w} = (w_1 + w_2)/2\). In this case, updating \(w_1\) and \(w_2\) first, and then averaging them results in the same model as updating \(\bar{w}\) if and only if both \(w_1\) and \(w_2\) classify \(x\) in the same way (correctly or incorrectly). This is because when updating \(\bar{w}\), we virtually update both \(w_1\) and \(w_2\) in the same way, irrespective of how they classify \(x\) individually.

This seems to suggest that P2PEGASOSUM is a better choice. We will test this hypothesis experimentally in Section VII where we will show that, surprisingly, it is not always true. The reason could be that P2PEGASOSMU and P2PEGASOSUM are in fact very similar when we consider the entire history of the distributed computation, as opposed to a single update step. The histories of the models define a directed acyclic graph (DAG), where the nodes are merging operations, and the edges correspond to the transfer of a model from one node to another. In both cases, there is one update corresponding to each edge: the only difference is whether the update occurs on the source node of the edge or on the target. Apart from this, the edges of the DAG are the same for both methods. Hence we see that P2PEGASOSMU has the favorable property that the updates that correspond to the incoming edges of a merge operation are done using independent samples, while for P2PEGASOSUM they are performed with the same example. Thus, P2PEGASOSMU guarantees a greater independence of the models.

In the following we present our theoretical results for both P2PEGASOSMU and P2PEGASOSUM. We note that these results do not assume any coordination or synchronization; they are based on a fully asynchronous communication model. First let us formally define the optimization problem at hand, and let us introduce some notation.

---

**Algorithm 3** Pegasos and Adaline updates, initialization, and merging

```
1: procedure UPDATEPEGASOS(m)
2: \[ \eta \leftarrow 1/(\lambda \cdot m.t) \]
3: if \( y \langle m.w, x \rangle < 1 \) then
4: \[ m.w \leftarrow (1 - \eta\lambda)m.w + \eta y x \]
5: else
6: \[ m.w \leftarrow (1 - \eta\lambda)m.w \]
7: end if
8: m.t \leftarrow m.t + 1
9: return m
10: end procedure
11:
12: procedure UPDATEADALINE(m)
13: \[ m.w \leftarrow m.w + \eta(y - \langle m.w, x \rangle)x \]
14: return m
15: end procedure
16:
17: procedure INITMODEL
18: currentModel.t \leftarrow 0
19: currentModel.w \leftarrow (0, \ldots, 0)^T
20: modelQueue.add(currentModel)
21: modelQueue.add(currentModel)
22: end procedure
23:
24: procedure MERGE(m_1, m_2)
25: \[ m.t \leftarrow \min(m_1.t, m_2.t) \]
26: \[ m.w \leftarrow (m_1.w + m_2.w)/2 \]
27: return m
28: end procedure
```
Let \( S = \{(x_i, y_i) : 1 \leq i \leq m, x_i \in \mathbb{R}^n, y_i \in \{+1, -1\}\} \) be a distributed training set with one data point at each network node. Let \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) be the objective function of the SVM learning problem:

\[
f(w) = \min_{w} \frac{1}{2} \|w\|^2 + \frac{1}{m} \sum_{(x,y) \in S} \ell(w; (x,y)),
\]

where \( \ell(w; (x,y)) = \max\{0, 1 - y(w,x)\} \) \( \odot \) Note that \( f \) is strongly convex with a parameter \( \lambda \). Let \( w^\star \) denote the global optimum of \( f \). For a fixed data point \((x_i, y_i)\) we define

\[
f_i(w) = \frac{\lambda}{2} \|w\|^2 + \ell(w; (x_i, y_i)),
\]

which is used to derive the update rule for the Pegasos algorithm. Obviously, \( f_i \) is \( \lambda \) strongly convex as well, since it has the same form as \( f \) with \( m = 1 \).

The update history of a model can be represented as a binary tree, where the nodes are models, and the edges are defined by the direct ancestor relation. Let us denote the direct ancestors of \( w^{(i+1)} \) as \( w^{(i)} \) and \( w^{(2)} \). These ancestors are averaged and then updated to obtain \( w^{(i+1)} \) (assuming the MU variant). Let the sequence \( w^{(0)}, \ldots, w^{(t)} \) be defined as the path in this history tree, for which

\[
w^{(i)} = \arg\max_{w \in \{w^{(i)}, w^{(2)}\}} \|w - w^\star\|, \quad i = 0, \ldots, t - 1.
\]

This sequence is well defined. Let \((x_i, y_i)\) denote the training example, that was used in the update step that resulted in \( w^{(i)} \) in the series defined above.

**Theorem 1 (P2PegasosMU convergence):** We assume that (1) each node receives an incoming message after any point in time within a finite time period (eventual update assumption), (2) there is a subgradient \( \nabla \) of the objective function such that \( \|\nabla_w\| \leq G \) for every \( w \). Then,

\[
\frac{1}{t} \sum_{i=1}^{t} f_i(\bar{w}^{(i)}) - f_i(w^\star) \leq \frac{G^2(\log(t) + 1)}{2\lambda t}
\]

where \( \bar{w}^{(i)} = (w^{(i)}_1 + w^{(i)}_2)/2 \).

**Proof:** During the running of the algorithm, let us pick any node on which at least one subgradient update has been performed already. There is such a node eventually, due to the eventual update assumption. Let the model currently stored at this node be \( w^{(i+1)} \).

We know that \( w^{(i+1)} = \bar{w}^{(i)} - \nabla^{(i)}/(\lambda t) \), where \( \bar{w}^{(i)} = (w^{(i)}_1 + w^{(i)}_2)/2 \) and where \( \nabla^{(i)} \) is the subgradient of \( f_i \). From the \( \lambda \)-convexity of \( f_i \) it follows that

\[
f_i(\bar{w}^{(i)}) - f_i(w^\star) + \frac{\lambda}{2} \|\bar{w}^{(i)} - w^\star\|^2 \leq \langle \bar{w}^{(i)} - w^\star, \nabla^{(i)} \rangle.
\]

On the other hand, the following inequality is also true, following from the definition of \( \bar{w}^{(i+1)} \), \( G \) and some algebraic rearrangements:

\[
\frac{M}{2} \|\bar{w}^{(i)} - w^\star\|^2 - \frac{M}{2} \|w^{(i+1)} - w^\star\|^2 \leq G^2/2\lambda t.
\]

Moreover, we can bound the distance of \( \bar{w}^{(i)} \) from \( w^\star \) with the distance of the ancestor of \( w^{(i)} \) that is further away from \( w^\star \) with the help of the Cauchy–Bunyakovsky–Schwarz inequality:

\[
\|\bar{w}^{(i)} - w^\star\|^2 \leq \frac{w^{(i)} - w^\star}{2} \frac{w^{(i+1)} - w^\star}{2} \leq \|w^{(i)} - w^\star\|^2.
\]

From (11), (12), (13) and the bound on the subgradients, we derive

\[
\sum_{i=1}^{t} f_i(\bar{w}^{(i)}) - f_i(w^\star) \leq \frac{\lambda(t - 1)}{2} \|w^{(i)} - w^\star\|^2 - \frac{\lambda t}{2} \|w^{(i+1)} - w^\star\|^2 + \frac{G^2}{2\lambda t}.
\]

Note that this bound also holds for \( w^{(i)}, 1 \leq i \leq t \). Summing up both sides of these \( t \) inequalities, we get the following bound:

\[
\sum_{i=1}^{t} f_i(\bar{w}^{(i)}) - f_i(w^\star) \leq \frac{\lambda t}{2} \|w^{(i+1)} - w^\star\|^2 + \frac{G^2}{2\lambda} \sum_{i=1}^{t} \frac{1}{t} \leq \frac{G^2(\log(t) + 1)}{2\lambda t},
\]

from which the theorem follows after division by \( t \).

The bound in (15) is analogous to the bound presented in \( \odot \) in the analysis of the Pegasos algorithm. It basically means that the average error tends to zero. To be able to show that the limit of the process is the optimum of \( f \), it is necessary that the samples involved in the series are uniform random samples. Investigating the distribution of the samples is left to future work; but we believe that the distribution closely approximates uniformity for a large \( t \), given the uniform random peer sampling that is applied.

For P2PegasosUM, an almost identical derivation leads us to a similar result (omitted due to lack of space).

**VI. EXPERIMENTAL RESULTS**

We experiment with two algorithms: P2PegasosUM and P2PegasosMU. In addition, to shed light on the behavior of these algorithms, we include a number of baseline methods as well. To perform the experiments, we used the PeerSIM event based P2P simulator [25].

**A. Experimental Setup**

\( \odot \) Baseline Algorithms: The first baseline we use is P2PegasosRW. If there is no message drop or message delay, then this is equivalent to the Pegasos algorithm, since in cycle \( t \) all peers will have models that are the result of Pegasos learning on \( t \) random examples. In case of message
delay and message drop failures, the number of samples will be less than \( t \), as a function of the drop probability and the delay.

We also examine two variants of weighted bagging. The first variant (\( WB_1 \)) is defined as

\[
h_{WB_1}(x, t) = \text{sgn}(\sum_{i=1}^{N} \langle x, w_i(t) \rangle),
\]

where \( N \) is the number of nodes in the network, and the linear models \( w_i(t) \) are learned with Pegasos over an independent sample of size \( t \) of the training data. This baseline algorithm can be thought of as the ideal utilization of the \( N \) independent updates performed in parallel by the \( N \) nodes in the network in each cycle. The gossip framework introduces dependencies among the models, so its performance can be expected to be worse.

In addition, in the gossip framework a node has influence from only \( 2^t \) models on average in cycle \( t \). To account for this handicap, we also use a second version of weighted bagging (\( WB_2 \)):

\[
h_{WB_2}(x) = \text{sgn}(\sum_{i=1}^{\min(2^t, N)} \langle x, w_i \rangle).
\]

The weighted bagging variants described above are not practical alternatives, these algorithms serve as a baseline only. The reason is that an actual implementation would require \( N \) independent models for prediction. This could be achieved by \( P2P \) \textit{Pegasos} \( RW \) with a distributed prediction, which would impose a large cost and delay for every prediction. This could also be achieved by all nodes running up to \( O(N) \) instances of \( P2P \) \textit{Pegasos} \( RW \), and using the \( O(N) \) local models for prediction; this is not feasible either. In sum, the point that we want to make is that our gossip algorithm approximates \( WB_2 \) quite well using only a single message per node in each cycle, due to the technique of merging models.

The last baseline algorithm we experiment with is \textit{Perfect Matching}. In this algorithm we replace the peer sampling component of the gossip framework: instead of all nodes picking random neighbors in each cycle, we create a random perfect matching among the peers so that every peer receives exactly one message. Our hypothesis was that—since this variant increases the efficiency of mixing—it will maintain a higher diversity of models, and so a better performance can be expected due to the “virtual bagging” effect we explained previously. Note that this algorithm is not intended to be practical either.

\( f \) Data Sets.: We used three different data sets: Reuters [26], Spambase, and the Malicious URLs [5] data sets, which were obtained from the UCI database repository [27]. These data sets are of different types including small and large sets containing a small or large number of features. Table I shows the main properties of these data sets, as well as the prediction performance of the Pegasos algorithm.

The original Malicious URLs data set has a huge number of features (\( \sim 3,000,000 \)), therefore we first performed a feature reduction step so that we can carry out simulations. Note that the message size in our algorithm depends on the number of features, therefore in a real application this step might also be useful in such extreme cases. We applied the well-known correlation coefficient method for each feature with the class label, and kept the ten features with the maximal absolute values. If necessary, this calculation can also be carried out in a gossip-based fashion [2], but we performed it offline. The effect of this dramatic reduction on the prediction performance is shown in Table I where Pegasos results on the full feature set are shown in parenthesis.

\( g \) Evaluation metric.: The evaluation metric we focus on is prediction error. To measure prediction error, we need to split the datasets into training sets and test sets. The proportions of this splitting are shown in Table I. In our experiments with \( P2PGPega sosMU \) and \( P2PGPegasosUM \) we track the misclassification ratio over the test set of 100 randomly selected peers. The misclassification ratio of a model is simply the number of the misclassified test examples divided by the number of all test examples, which is the so-called 0-1 error. Prediction at a given peer is performed using its local model \textit{Current Model}.

For the baseline algorithms we used all the available models for calculating the error rate, which equals the number of training samples. From the Malicious URLs database we used only 10,000 examples selected at random, to make the evaluation computationally feasible. Note, that we found that increasing the number of examples beyond 10,000 does not result in a noticeable difference in the observed behavior.

We also calculated the similarities between the models circulating in the network, using the cosine similarity measure. We calculated the similarity between all pairs of models, and calculated the average. This metric is useful to study the speed at which the actual models converge. Note that a uniform sampling it is known that all models converge to an optimal model.

\( h \) Modeling failure.: In a set of experiments we model extreme message drop and message delay. Drop probability is set to be 0.5. This can be considered an extremely large drop rate. Message delay is modeled as a uniform random delay from the interval \([\Delta, 10\Delta] \), where \( \Delta \) is the gossip period in Algorithm I. This is also an extreme delay, orders of magnitudes higher than what can be expected in a realistic scenario, except if \( \Delta \) is very small.

B. Results and Discussion

The experimental results are shown in Figures I and II. Note that all variants can be mathematically proven to converge to the same result, so the difference is in convergence speed only. Bagging can temporarily outperform a single instance of Pegasos, but after enough training samples, all models become almost identical, so the advantage of voting disappears.

In Figure I we can see that our hypothesis about the relationship of the performance of the gossip algorithms and the baselines is validated: the standalone Pegasos algorithm is
the slowest, while the two variants of weighted bagging are the fastest. P2PEGASOSMU approximates WB_2 quite well, with some delay, so we can use WB_2 as a heuristic model of the behavior of the algorithm. Note that the convergence is several orders of magnitude faster than that of Pegasos (the plots have a logarithmic scale).

Figure 1 also contains results from our extreme failure scenario. We can observe that the difference in convergence speed is mostly accounted for by the increased message delay. The effect of the delay is that all messages wait 5 cycles on average before being delivered, so the convergence is proportionally slower. In addition, half of the messages get lost too, which adds another factor of about 2 to the convergence speed. Apart from slowing down, the algorithms still converge to the correct value despite the extremely unreliable environment, as was expected.

Figure 2 illustrates the difference between the UM and MU variants. In Section V-B we pointed out that—although the UM version looks favorable when considering a single node—when looking at the full history of the learning process P2PEGASOSMU maintains more independence between the models. Indeed, the MU version clearly performs better according to our experiments. We can also observe that the UM version shows a lower level of model similarity in the system, which probably has to do with the slower convergence.

In Figure 2 we can see the performance of the perfect matching variant of P2PEGASOSMU as well. Contrary to our expectations, perfect matching does not clearly improve performance, apart from the first few cycles. It is also interesting to observe, that model similarity is correlated to prediction performance also in this case. We also note, that in the case of the Adaline-based gossip learning implementation perfect matching is clearly better than random peer sampling (not shown). This means that this behavior is due to the context-dependence of the update rule discussed in V-B.

VII. CONCLUSIONS

We proposed gossip learning as a generic approach to learn models of fully distributed data in large scale P2P systems. We presented an instantiation of gossip learning based on the Pegasos algorithm. The algorithm was shown to be extremely robust to message drop and message delay, furthermore, a very significant speedup was demonstrated w.r.t. the baseline Pegasos algorithm.
The algorithm makes it possible to compute predictions locally at every node in the network at any point in time, yet the message complexity is acceptable. The main features that differentiate this approach from related work are the focus on fully distributed data, the modularity and generality, the simplicity, and in general, its design philosophy.

A main part of this design philosophy is that the algorithm should run continuously, hence we did not focus on stopping conditions at all. Instead, we were interested in the cost of the algorithm as a one-shot algorithm with a synchronized start, this was merely to be able to characterize its convergence properties in a clear manner. However, a main direction we are pursuing is adaptivity to changing environments. We are also investigating the strength of the privacy preservation gossip learning could offer. This is also an aspect that had a strong influence on the design philosophy from the start, but its impact on fully distributed data, the modularity and generality, the simplicity, and in general, its design philosophy.

Although here we considered linear models only, linear algorithms obviously cover many interesting nonlinear representations through nonlinear feature transformations. Besides, the generic gossip learning scheme can be instantiated with an arbitrary online learner, if CREATEMODELRW is applied as an implementation of CREATEMODEL, which does not use the merge operator. Still, it would be very interesting to characterize the model spaces that have a merge operator that is capable of providing a speedup similar to the linear case. Investigating this question is part of our future work.

REFERENCES

[16] W. Kowalczyk and N. Vlassis, “Newscast EM,” in 17th Advances in


