CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2012)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.2858

SPECIAL ISSUE PAPER

Gossip learning with linear models on fully distributed data

Róbert Ormándi [1], István Hegedűs [1] and Márk Jelasity [1,2,*,†]

[1]*Research Group on Artificial Intelligence, University of Szeged, Szeged, Hungary*
[2]*Hungarian Academy of Sciences, Budapest, Hungary*

SUMMARY

Machine learning over fully distributed data poses an important problem in peer-to-peer applications. In this model, we have one data record at each network node but without the possibility to move raw data because of 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; the system model offers almost no guarantee for reliability, 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 with independent random walks. We prove the convergence of the method theoretically, and perform extensive experiments on benchmark data sets. Our experimental analysis demonstrates the performance and robustness of the proposed approach. Copyright © 2012 John Wiley & Sons, Ltd.

Received 15 November 2011; Revised 26 April 2012; Accepted 27 April 2012

KEY WORDS: P2P; gossip; bagging; online learning; stochastic gradient descent; random walk

1. 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. For example, for a P2P platform that offers rich functionality to its users including spam filtering, personalized search, and recommendation [1–3], or for P2P approaches for detecting distributed attack vectors [4], complex predictive models have to be built on the basis of fully distributed, and often sensitive, data.

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,

*Correspondence to: Márk Jelasity, Research Group on Artificial Intelligence, University of Szeged, Szeged, P. O. Box 652. H-6701, Hungary.

†E-mail: jelasity@inf.u-szeged.hu

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 because 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. Because 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.

Our specific contributions include the following: (i) we propose gossip learning, a novel and generic approach for P2P learning on fully distributed data, which can be instantiated in various different ways; (ii) we introduce a novel, efficient distributed ensemble learning method for linear models, which 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 2 elaborates on the fully distributed data model. Section 3 summarizes related work and the background concepts. In Section 4, we describe our generic approach and a naive algorithm as an example. Section 5 presents the core algorithmic contributions of the paper along with a theoretical discussion, and Section 6 contains an experimental analysis. Section 7 concludes the paper.

This paper is a significantly extended and improved version of our previous work [5].

2. 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. Because 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., [6–8]). Here, the assumption is that every node stores only a single record, for example, a sensor reading. The motivation for not collecting raw data but processing it in place is mainly to achieve robustness and adaptivity through not relying on any central servers. In some systems, such as 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 smart phone applications [9–11] and in P2P social networking [12], where the key motivation is giving the user full control over personal data. In these applications, it is also common for a user to contribute only a single record, for example, a personal profile, a search history, or a sensor reading by a smart phone.

Clearly, in P2P smart phone applications and 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 with 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. Because 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 because 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.

3. BACKGROUND AND RELATED WORK

We organize the discussion of the background of our work along the generic model components outlined in Section 1 and explained in Section 4: online learning, ensemble learning, and peer sampling. We also discuss related work in P2P data mining. Here, we do not consider parallel data mining algorithms. This field has a large literature, but the rather different underlying system model means it is of little relevance to us here.

Online Learning. The basic problem of *supervised binary classification* can be defined as follows. Let us assume that we are given a labeled database in the form of pairs of feature vectors and their correct classification, that is, $(x_1, y_1), \dots, (x_n, y_n)$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. The constant d is the *dimension* of the problem (the number of features). We are looking for a *model* $f : \mathbb{R}^d \to \{-1, 1\}$ that correctly classifies the available feature vectors, and that can also *generalize* well, that is, which can classify unseen examples too. For testing purposes, the available data is often partitioned into a *training set* and a *test set*, the latter being used only for testing candidate models.

Supervised learning can be thought of as an optimization problem, where we want to maximize prediction performance, which can be measured via, for example, the number of feature vectors that are classified correctly over the training set. The search space of this problem consists of the set of possible models (the *hypothesis space*), and each method also defines a specific search algorithm (often called the *training algorithm*) that eventually selects one model from this space.

Training 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 are called *online learning algorithms*. Gossip learning relies on this type of learning algorithms. Ma *et al.* provide a nice summary of online learning for large scale data [13].

Stochastic gradient search [14, 15] is a generic algorithmic family for implementing online learning methods. Without going into too much detail, the basic idea is that we iterate over the training examples in a random order repeatedly, and for each training example, we calculate the gradient of the error function (which describes classification error) and modify the model along this gradient to reduce the error on this particular example. At the same time, the step size along the gradient is gradually reduced. In many instantiations of the method, it can be proven that the converged model minimizes the *sum* of the errors over the examples [16].

Let us now turn to support vector machines (SVM), the learning algorithm we apply in this paper [17]. In its simplest form, the SVM approach works with the space of linear models to solve the binary classification problem. Assuming a d dimensional problem, we want to find a $d-1$ dimensional separating hyperplane that maximizes the *margin* that separates examples of the two class. The margin is defined by the hyperplane as the sum of the minimal perpendicular distances from both classes.

Equation (1) states a variant of the formal SVM optimization problem, where $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are the parameters of model, namely the norm of the separating hyperplane and the bias parameters, respectively. Furthermore, ξ_i is the slack variable of the ith sample, which can be interpreted as the amount of misclassification error of the ith sample, and C is a trade-off parameter between generalization and error minimization.

$$\begin{aligned} \min_{w,b,\xi_i} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geqslant 1 - \xi_i \quad \text{and} \quad \xi_i \geqslant 0 \quad (\forall i : 1 \leqslant i \leqslant n) \end{aligned} \tag{1}$$

The Pegasos algorithm is an SVM training algorithm on the basis of a stochastic gradient descent approach [18]. It directly optimizes a form of the previously defined, so-called primal optimization task. We will use the Pegasos algorithm as a basis for our distributed method. In this primal form, the desired model w is explicitly represented and is evaluated directly over the training examples.

Because in the context of SVM learning this is an unusual approach, let us take a closer look at why we decided to work in the primal formulation. The standard SVM algorithms solve the dual problem instead of the primal form [17]. The dual form is

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i y_i \alpha_j y_j x_i^T x_j \\ \text{s.t.} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \quad \text{and} \quad 0 \leqslant \alpha_i \leqslant C \quad (\forall i : 1 \leqslant i \leqslant n), \end{aligned} \tag{2}$$

where the variables α_i are the Lagrangian variables. The Lagrangian variables can be interpreted as the weights of the training samples, which specify how important the corresponding sample is from the point of view of the model.

The primal and dual formalizations are equivalent, both in terms of theoretical time complexity and the optimal solution. Solving the dual problem has some advantages; most importantly, one can take full advantage of the kernel-based extensions (which we have not discussed here) that introduce nonlinearity into the approach. However, methods that deal with the dual form require frequent access to the entire database to update α_i, which is unfeasible in our system model. Besides, the number of variables α_i equals the number of training samples, which could be orders of magnitude larger than the dimension of the primal problem, d. Finally, there are indications that applying the primal form can achieve a better generalization on some databases [19].

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 [20]. We apply a method for combining the models in the network that is related to both bagging [21] and 'pasting small votes' [22]: when the models start their random walk, initially they are based on non-overlapping small subsets of the training data because of 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).

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 [23–25]. In our algorithm, we apply gossip-based peer sampling [26] 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.

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. The literature of this field is vast; we mention only two examples: Astrolabe [6] and gossip-based averaging [7]. 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 expectation–maximization algorithm [27], unsupervised learners [28], or the collaborative filtering-based recommender algorithms [29–32]. 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, such as those in [33–39]. 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. Because we assume that data cannot be moved, the models need to visit data instead. In a setting such as this, the main problem we need to solve is to efficiently aggregate the various models that evolve slowly in the system to speed up the convergence of prediction performance.

To the best of our knowledge, there is no other learning approach designed to work in our fully asynchronous and unreliable message passing model and which is capable of producing a large array of state-of-the-art models.

4. 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. On the basis of every incoming model, a new model is created potentially combining it with the previous incoming model. This newly created model is stored in a cache of a fixed size. When the cache is full, the model stored for the longest time is replaced by the newly added model. The cache provides a pool of recent models that can be used to implement, for example, voting-based prediction. We discuss this possibility in Section 6. In the active loop, the freshest model (the model added to the cache most recently) is sent to a random peer.

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 length of the period of the loop Δ is the same

Algorithm 1 Gossip Learning Scheme

```
1: initModel()
2: loop
3:     wait($\Delta$)
4:     $p \leftarrow$ selectPeer()
5:     send modelCache.freshest() to $p$
6: end loop

7: procedure ONRECEIVEMODEL($m$)
8:     modelCache.add(createModel($m$, lastModel))
9:     lastModel $\leftarrow m$
10: end procedure
```

at all nodes. However, during the evaluations, Δ was modeled as a normally distributed random variable with parameters $\mu = \Delta$ and $\sigma^2 = \Delta/10$. 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 3. Here, we use the NEWSCAST algorithm [26], 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 on the basis of locally available information—the two models received most recently 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.

The abstract method UPDATE represents the online learning algorithm—the second main component of our framework besides peer sampling—that updates the model on the basis of one example (the local example of the node). Procedure CREATEMODELRW 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 learning. 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 on the basis of a single model [21, 22], 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 mentioned previously. 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 Δ 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

Algorithm 2 CREATEMODEL: three implementations

```
1:  procedure CREATEMODELRW($m_1, m_2$)
2:      return update($m_1$)
3:  end procedure
4:
5:  procedure CREATEMODELMU($m_1, m_2$)
6:      return update(merge($m_1, m_2$))
7:  end procedure
8:  procedure CREATEMODELUM($m_1, m_2$)
9:      return merge(update($m_1$),update($m_2$))
10: end procedure
```

computational cost is one or two update steps in each Δ time units for the UM or the MU variants, respectively. The exact cost of this step depends on the selected online learner.

5. 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.

5.1. The Adaline perceptron

We consider here the Adaline perceptron [40] that arguably has one of the simplest update rules because of its linear activation function. Without loss of generality, we ignore the bias term. The error function to be optimized is defined as

$$E_x(w) = \frac{1}{2}(y - \langle w, x\rangle)^2, \tag{3}$$

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

$$\nabla_w = \frac{\partial E_x(w)}{\partial w} = -(y - \langle w, x\rangle)x \tag{4}$$

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

$$w^{(k+1)} = w^{(k)} + \eta(y - \langle w^{(k)}, x\rangle)x, \tag{5}$$

where η 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 is $\langle w, x\rangle$. It is not hard to see that

$$h(x) = \langle \bar{w}, x\rangle = \frac{1}{m}\left\langle \sum_{i=0}^{m} w_i, x \right\rangle = \frac{1}{m}\sum_{i=0}^{m}\langle w_i, x\rangle, \tag{6}$$

which means that the voting-based prediction is equivalent to prediction on the basis of 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 $\operatorname{sgn}\langle w, x\rangle$, the same equivalence holds:

$$h(x) = \operatorname{sgn}\left(\frac{1}{m}\sum_{i=1}^{m}|\langle w, x\rangle|\operatorname{sgn}\langle w, x\rangle\right) = \operatorname{sgn}\left(\frac{1}{m}\sum_{i=1}^{m}\langle w_i, x\rangle\right) = \operatorname{sgn}\langle \bar{w}, x\rangle. \tag{7}$$

A similar approach to this weighted voting mechanism has been shown to improve the performance of simple vote counting [41]. Our preliminary experiments also support this.

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

$$\bar{w} + \eta(y - \langle \bar{w}, x\rangle)x = \frac{1}{m}\sum_{i=1}^{m} w_i + \eta(y - \langle w_i, x\rangle)x. \tag{8}$$

Algorithm 3 Pegasos and Adaline updates, initialization, and merging

```
1: procedure UPDATEPEGASOS($m$)
2:     $m.t \leftarrow m.t + 1$
3:     $\eta \leftarrow 1/(\lambda \cdot m.t)$
4:     if $y\,\langle m.w, x\rangle < 1$ then
5:         $m.w \leftarrow (1 - \eta\lambda)m.w + \eta y x$
6:     else
7:         $m.w \leftarrow (1 - \eta\lambda)m.w$
8:     end if
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: procedure INITMODEL
17:     lastModel.$t \leftarrow 0$
18:     lastModel.$w \leftarrow (0, \ldots, 0)^T$
19:     modelCache.add(lastModel)
20: end procedure
21:
22: procedure MERGE($m_1$,$m_2$)
23:     $m.t \leftarrow \max(m_1.t, m_2.t)$
24:     $m.w \leftarrow (m_1.w + m_2.w)/2$
25:     return $m$
26: end procedure
```

The aforementioned properties lead to a rather important observation. If we implement our gossip learning skeleton by 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 4. That is, averaging effectively reduces the exponential message complexity to transmitting a *single* model in each cycle independently 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 6, bagging over N-independent models actually outperforms the gossip learning algorithms.

5.2. Pegasos

Here, we discuss the adaptation of Pegasos (a linear SVM gradient method [18] 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 [18]. 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. Because of 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 6, 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 carried out using independent samples, whereas 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.

Let $S = \{(x_i, y_i) : 1 \leqslant i \leqslant n, x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}\}$ be a distributed training set with one data point at each network node. Let $f : \mathbb{R}^d \to \mathbb{R}$ be the objective function of the SVM learning problem (applying the L1 loss in the more general form proposed in Equation (1)):

$$\begin{aligned} f(w) &= \min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{n} \sum_{(x,y)\in\mathcal{S}} \ell(w;(x,y)), \\ &\text{where } \ell(w;(x,y)) = \max\{0, 1 - y\langle w, x\rangle\}. \end{aligned} \tag{9}$$

Note that f is strongly convex with a parameter λ [18]. Let $w^\star$ denote the global optimum of f. For a fixed data point (x_i, y_i), we define

$$f_i(w) = \frac{\lambda}{2}\|w\|^2 + \ell(w;(x_i, y_i)), \tag{10}$$

which is used to derive the update rule for the Pegasos algorithm. Obviously, f_i is λ strongly convex as well because 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 $w_1^{(i)}$ and $w_2^{(i)}$. 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

$$w^{(i)} = \arg\max_{w\in\{w_1^{(i)}, w_2^{(i)}\}} \|w - w^\star\|, \;\; i = 0, \ldots, t-1. \tag{11}$$

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

*Theorem 1 (*P2PEGASOSMU *convergence)*
We assume that (i) each node receives an incoming message after any point in time within a finite time period (eventual update assumption), and (ii) there is a subgradient ∇ of the objective function such that $\|\nabla_w\| \leqslant G$ for every w. Then,

$$\frac{1}{t}\sum_{i=1}^{t} f_i(\bar{w}^{(i)}) - f_i(w^\star) \leqslant \frac{G^2(log(t) + 1)}{2\lambda t}, \tag{12}$$

where $\bar{w}^{(i)} = (w_1^{(i)} + w_2^{(i)})/2$.

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 because of the eventual update assumption. Let the model currently stored at this node be $w^{(t+1)}$.

We know that $w^{(t+1)} = \bar{w}^{(t)} - \nabla^{(t)}/(\lambda t)$, where $\bar{w}^{(t)} = (w_1^{(t)} + w_2^{(t)})/2$ and $\nabla^{(t)}$ is the subgradient of f_t. From the λ-convexity of f_t, it follows that

$$f_t(\bar{w}^{(t)}) - f_t(w^\star) + \frac{\lambda}{2}\|\bar{w}^{(t)} - w^\star\|^2 \leqslant \langle \bar{w}^{(t)} - w^\star, \nabla^{(t)}\rangle. \tag{13}$$

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

$$\left\langle \bar{w}^{(t)} - w^{\star}, \nabla^{(t)} \right\rangle \leqslant \frac{\lambda t}{2} \|\bar{w}^{(t)} - w^{\star}\|^2 - \frac{\lambda t}{2} \|w^{(t+1)} - w^{\star}\|^2 + \frac{G^2}{2\lambda t}. \tag{14}$$

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

$$\|\bar{w}^{(t)} - w^{\star}\|^2 = \left\| \frac{w_1^{(t)} - w^{\star}}{2} + \frac{w_2^{(t)} - w^{\star}}{2} \right\|^2 \leqslant \|w^{(t)} - w^{\star}\|^2. \tag{15}$$

From (13), (14), (15), and the bound on the subgradients, we derive

$$f_t(\bar{w}^{(t)}) - f_t(w^{\star}) \leqslant \frac{\lambda(t-1)}{2} \|w^{(t)} - w^{\star}\|^2 - \frac{\lambda t}{2} \|w^{(t+1)} - w^{\star}\|^2 + \frac{G^2}{2\lambda t}. \tag{16}$$

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

$$\sum_{i=1}^{t} f_i(\bar{w}^{(i)}) - f_i(w^{\star}) \leqslant -\frac{\lambda t}{2} \|w^{(t+1)} - w^{\star}\|^2 + \frac{G^2}{2\lambda} \sum_{i=1}^{t} \frac{1}{i} \leqslant \frac{G^2(log(t)+1)}{2\lambda}, \tag{17}$$

from which the theorem follows after division by t. □

The bound in (17) is analogous to the bound presented in [18] in the analysis of the PEGASOS algorithm. It basically means that the average error tends to be zero. 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 [18]. 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 because of lack of space).

6. 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 [42].

6.1. Experimental setup

6.1.1. 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 because 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 (WB1) is defined as

$$h_{WB1}(x,t) = \text{sgn}\left(\sum_{i=1}^{N} \langle x, w_i^{(t)} \rangle \right), \tag{18}$$

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 (WB2):

$$h_{WB2}(x) = \operatorname{sgn}\left(\sum_{i=1}^{\min(2^t,N)} \langle x, w_i \rangle\right). \tag{19}$$

The weighted bagging variants described previously 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 P2PEGASOSRW 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 P2PEGASOSRW 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 WB2 quite well by using only a single message per node in each cycle because of the technique of merging models.

The last baseline algorithm we experiment with is 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—because this variant increases the efficiency of mixing—it will maintain a higher diversity of models, and so, a better performance can be expected because of the 'virtual bagging' effect we explained previously. Note that this algorithm is not intended to be practical either.

6.1.2. Data sets. We used three different data sets, Reuters [43], Spambase, and the Malicious URLs [13], which were obtained from the University of California, Irvine, database repository [44]. 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 10 features with the maximal absolute values. If necessary, this calculation can also be carried out in a gossip-based fashion [7], 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.

6.1.3. Using the local models for prediction. An important aspect of our protocol is that every node has at least one model available locally, and thus, all the nodes can perform a prediction. Moreover, because the nodes can remember the models that pass through them at no communication cost, we cheaply implement a simple voting mechanism, where nodes will use more than one model to make

Table I. The main properties of the data sets and the prediction error (0–1 error) of the baseline sequential algorithm.

	Reuters	SpamBase	Malicious URLs (10)
Training set size	2000	4140	2,155,622
Test set size	600	461	240,508
Number of features	9947	57	10
Class label ratio	1300:1300	1813:2788	792,145:1,603,985
Pegasos 20,000 iter.	0.025	0.111	0.080 (0.081)

In the case of Malicious URLs data set, the results of the full feature set are shown in parentheses.

Algorithm 4 Local prediction procedures

```
1: procedure PREDICT($x$)
2:     $w \leftarrow$ modelCache.freshest()
3:     return sign($\langle w, x\rangle$)
4: end procedure

5: procedure VOTEDPREDICT($x$)
6:     pRatio $\leftarrow 0$
7:     for $m \in$ modelCache do
8:         if sign($\langle m.w, x\rangle$) $\geq 0$ then
9:             pRatio $\leftarrow$ pRatio $+1$
10:        end if
11:    end for
12:    return sign(pRatio/modelCache.size()$-0.5$)
13: end procedure
```

predictions. Algorithm 4 shows the procedures used for prediction in the original case and in the case of voting. Here, the vector x is the unseen example to be classified. In the case of linear models, the classification is simply the sign of the inner product with the model, which essentially describes on which side of the hyperplane the given point lies. In our experiments, we used a cache of size 10.

6.1.4. Evaluation metric. The evaluation metric we focus on is prediction error. To measure prediction error, we need to split the data sets into training sets and test sets. The proportions of this splitting are shown in Table I. In our experiments with P2PEGASOSMU and P2PEGASOSUM, 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.

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 by 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 under uniform sampling, it is known that all models converge to an optimal model.

6.1.5. 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 Δ is the gossip period in Algorithm 1. This is also an extreme delay, orders of magnitudes higher than what can be expected in a realistic scenario, except if Δ is very small. We also model realistic churn on the basis of probabilistic models in [45]. Accordingly, we approximate online session length with a lognormal distribution, and we approximate the parameters of the distribution by using a maximum likelihood estimate on the basis of a trace from a private BitTorrent community called FileList.org obtained from Delft University of Technology [46]. We set the offline session lengths so that at any moment, 90% of the peers are online. In addition, we assume that when a peer comes back online, it retains its state that it had at the time of leaving the network.

6.2. Results and discussion

The experimental results for prediction without local voting are shown in Figures 1 and 2. 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 1, 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, whereas the two variants of weighted bagging are the fastest. P2PEGASOSMU approximates WB2 quite well, with some delay, so we can use WB2 as a heuristic model of the behavior of the algorithm. Note

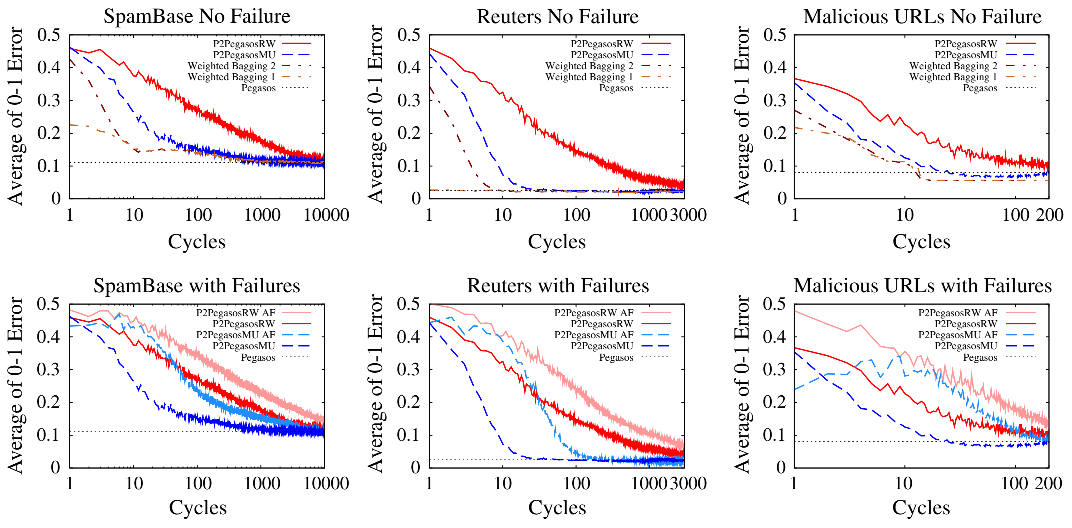

Figure 1. Experimental results without failure (upper row) and with extreme failure (lower row). AF means all possible failures are modeled.

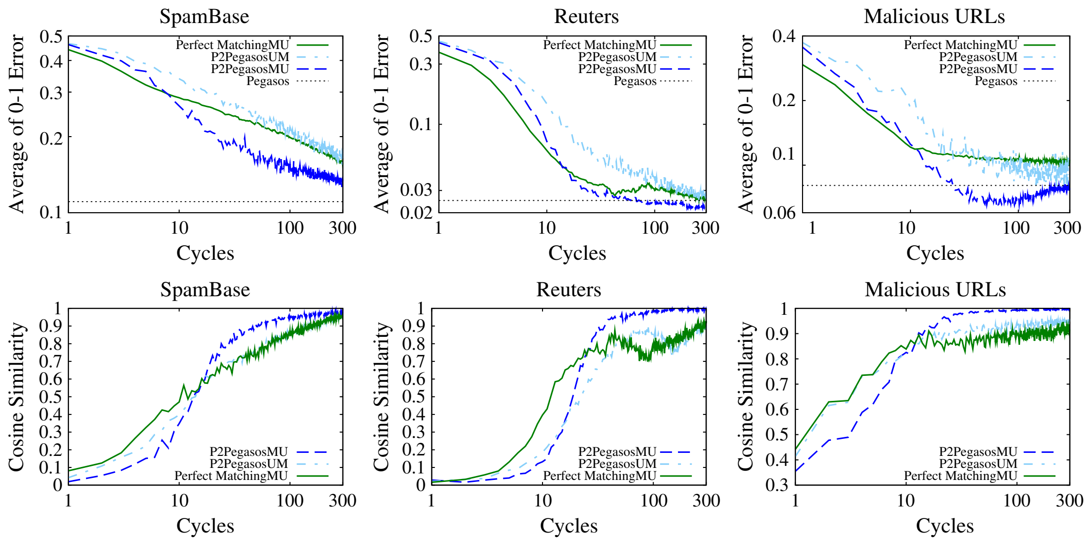

Figure 2. Prediction error (upper row) and model similarity (lower row) with PERFECT MATCHING and P2PEGASOSUM.

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 five 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. Here, we model no failures. In Section 5.2, 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

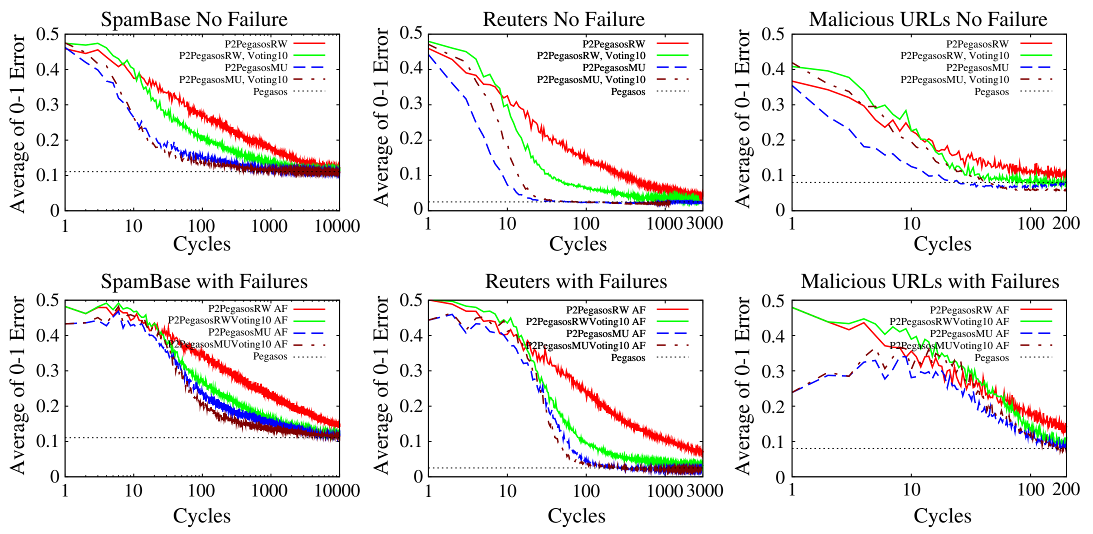

Figure 3. Experimental results applying local voting without failure (upper row) and with extreme failure (lower row).

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 Section 5.2.

The results with local voting are shown in Figure 3. The main conclusion is that voting results in a significant improvement when applied along with P2PEGASOSRW, the learning algorithm that does not apply merging. When merging is applied, the improvement is less dramatic. In the first few cycles, voting can result in a slight degradation of performance. This could be expected because the models in the local caches are trained on fewer samples on average than the freshest model in the cache. Overall, because voting is for free, it is advisable to use it.

7. CONCLUSIONS

We proposed gossip learning as a generic approach to learn models of fully distributed data in large scale P2P systems. The basic idea of gossip learning is that many models perform a random walk over the network, while being updated at every node they visit and while being combined (merged) with other models they encounter. We presented an instantiation of gossip learning on the basis the Pegasos algorithm. The algorithm was shown to be extremely robust to message drop and message delay; furthermore, a very significant speedup was demonstrated with regard to the baseline Pegasos algorithm because of the model merging technique and the prediction algorithm that is based on local voting.

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: every node sends one model in each gossip cycle. The main features that differentiate this approach from related work are the focus on fully distributed data and its modularity, generality, and simplicity.

An important promise of the approach is the support for privacy preservation because data samples are not observed directly. Although in this paper we did not focus on this aspect, it is easy to see that the only feasible attack is the multiple forgery attack [47], where the local sample is guessed on the basis of sending specially crafted models to nodes and observing the result of the update step.

This is very hard to perform even without any extra measures, given that models perform random walks on the basis of local decisions and that merge operations are performed as well.

This short informal reasoning motivates our ongoing work towards understanding and enhancing the privacy-preserving properties of gossip learning.

ACKNOWLEDGEMENTS

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).

REFERENCES

1. Pouwelse JA, Garbacki P, Wang J, Bakker A, Yang J, Iosup A, Epema DHJ, Reinders M, van Steen MR, Sips HJ. TRIBLER: a social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience* 2008; **20**(2):127–138. DOI: 10.1002/cpe.1189.
2. Bai X, Bertier M, Guerraoui R, Kermarrec AM, Leroy V. Gossiping personalized queries. *Proceedings of the 13th International Conference on Extending Database Technology (EBDT'10)*, Lausanne, 2010.
3. Buchegger S, Schiöberg D, Vu LH, Datta A. PeerSoN: P2P social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems (SNS'09)*. ACM: New York, NY, USA, 2009; 46–52, DOI: 10.1145/1578002.1578010.
4. Cheetancheri SG, Agosta JM, Dash DH, Levitt KN, Rowe J, Schooler EM. A distributed host-based worm detection system. In *Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense (LSAD'06)*. ACM: New York, NY, USA, 2006; 107–113, DOI: 10.1145/1162666.1162668.
5. Ormándi R, Hegedűs I, Jelasity M. Asynchronous peer-to-peer data mining with stochastic gradient descent. In *17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011)*, Vol. 6852, Lecture Notes in Computer Science. Springer-Verlag: Berlin Heidelberg, 2011; 528–540.
6. van Renesse R, Birman KP, Vogels W. Astrolabe: a robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems* 2003; **21**(2):164–206. DOI: 10.1145/762483.762485.
7. Jelasity M, Montresor A, Babaoglu O. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems* 2005; **23**(3):219–252. DOI: 10.1145/1082469.1082470.
8. Boyd S, Ghosh A, Prabhakar B, Shah D. Randomized gossip algorithms. *IEEE Transactions on Information Theory* 2006; **52**(6):2508–2530.
9. Pentland AS. Society's nervous system: building effective government, energy, and public health systems. *Computer* 2012; **45**(1):31–38. DOI: 10.1109/MC.2011.299.
10. Abdelzaher T, Anokwa Y, Boda P, Burke J, Estrin D, Guibas L, Kansal A, Madden S, Reich J. Mobiscopes for human spaces. *Pervasive Computing, IEEE* 2007; **6**(2):20–29. DOI: 10.1109/MPRV.2007.38.
11. Lane N, Miluzzo E, Lu H, Peebles D, Choudhury T, Campbell A. A survey of mobile phone sensing. *Communications Magazine, IEEE* 2010; **48**(9):140–150. DOI: 10.1109/MCOM.2010.5560598.
12. Diaspora. https://joindiaspora.com/.
13. Ma J, Saul LK, Savage S, Voelker GM. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)*. ACM: New York, NY, USA, 2009; 681–688, DOI: 10.1145/1553374.1553462.
14. Bottou L. The tradeoffs of large-scale learning, 2007. Tutorial at the 21st Annual Conference on Neural Information Processing Systems (NIPS), http://leon.bottou.org/talks/largescale.
15. Bottou L, LeCun Y. Large scale online learning. In *Advances in Neural Information Processing Systems 16*, Thrun S, Saul L, Schölkopf B (eds). MIT Press: Cambridge, MA, 2004.
16. Duda RO, Hart PE, Stork DG. *Pattern Classification*, (2nd edn). Wiley-Interscience: New York, 2000.
17. Cristianini N, Shawe-Taylor J. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press: Cambridge, 2000.
18. Shalev-Shwartz S, Singer Y, Srebro N, Cotter A. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming B* 2010. DOI: 10.1007/s10107-010-0420-4.
19. Chapelle O. Training a support vector machine in the primal. *Neural Computation* 2007; **19**:1155–1178. DOI: 10.1162/neco.2007.19.5.1155.
20. Rokach L. Ensemble-based classifiers. *Artificial Intelligence Review* 2010; **33**(1):1–39. DOI: 10.1007/s10462-009-9124-7.
21. Breiman L. Bagging predictors. *Machine Learning* 1996; **24**(2):123–140. DOI: 10.1007/BF00058655.
22. Breiman L. Pasting small votes for classification in large databases and on-line. *Machine Learning* 1999; **36**(1–2): 85–103. DOI: 10.1023/A:1007563306331.
23. King V, Saia J. Choosing a random peer. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC'04)*. ACM Press: New York, 2004; 125–130, DOI: 10.1145/1011767.1011786.

24. Stutzbach D, Rejaie R, Duffield N, Sen S, Willinger W. On unbiased sampling for unstructured peer-to-peer networks. *IEEE/ACM Transactions on Networking* 2009; **17**(2):377–390. DOI: 10.1109/TNET.2008.2001730.
25. Hall C, Carzaniga A. Uniform sampling for directed P2P networks. In *Euro-Par 2009*, Vol. 5704, Sips H, Epema D, Lin HX (eds), Lecture Notes in Computer Science. Springer: Berlin Heidelberg, 2009; 511–522, DOI: 10.1007/978-3-642-03869-3_49.
26. Jelasity M, Voulgaris S, Guerraoui R, Kermarrec AM, van Steen M. Gossip-based peer sampling. *ACM Transactions on Computer Systems* 2007; **25**(3):8. DOI: 10.1145/1275517.1275520.
27. Kowalczyk W, Vlassis N. Newscast EM. In *17th Advances in Neural Information Processing Systems (NIPS)*, Saul LK, Weiss Y, Bottou L (eds). MIT Press: Cambridge, MA, 2005; 713–720.
28. Siersdorfer S, Sizov S. Automatic document organization in a p2p environment. In *Advances in Information Retrieval*, Vol. 3936, Lalmas M *et al.* (eds), LNCS. Springer: Berlin Heidelberg, 2006; 265–276.
29. Ormándi R, Hegedűs I, Jelasity M. Overlay management for fully distributed user-based collaborative filtering. In *Euro-Par 2010*, Vol. 6271, D'Ambra P, Guarracino M, Talia D (eds), Lecture Notes in Computer Science. Springer-Verlag: Berlin Heidelberg, 2010; 446–457, DOI: 10.1007/978-3-642-15277-1_43.
30. Bakker A, Ogston E, van Steen M. Collaborative filtering using random neighbours in peer-to-peer networks. In *Proceeding of the 1st ACM International Workshop on Complex Networks Meet Information and Knowledge Management (CNIKM '09)*. ACM: New York, NY, USA, 2009; 67–75, DOI: 10.1145/1651274.1651288.
31. Han P, Xie B, Yang F, Wang J, Shen R. A novel distributed collaborative filtering algorithm and its implementation on P2P overlay network. In *Advances in Knowledge Discovery and Data Mining*, Vol. 3056, Dai H, Srikant R, Zhang C (eds), LNCS. Springer: Berlin Heidelberg, 2004; 106–115.
32. Tveit A. Peer-to-peer based recommendations for mobile commerce. In *Proceedings 1st Intl. Workshop on Mobile Commerce (WMC '01)*. ACM: New York, 2001; 26–29.
33. Luo P, Xiong H, Lü K, Shi Z. Distributed classification in peer-to-peer networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*. ACM: New York, NY, USA, 2007; 968–976, DOI: 10.1145/1281192.1281296.
34. Ang H, Gopalkrishnan V, Ng W, Hoi S. Communication-efficient classification in P2P networks. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, Vol. 5781, Buntine W, Grobelnik M, Mladenic D, Shawe-Ta ylor J (eds), Lecture Notes in Computer Science. Springer: Berlin Heidelberg, 2009; 83–98, DOI: 10.1007/978-3-642-04180-8_23.
35. Ang H, Gopalkrishnan V, Ng W, Hoi S. On classifying drifting concepts in P2P networks. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, Vol. 6321, Balcázar J, Bonchi F, Gionis A, Sebag M (eds), Lecture Notes in Computer Science. Springer: Berlin Heidelberg, 2010; 24–39, DOI: 10.1007/978-3-642-15880-3_8.
36. Ang H, Gopalkrishnan V, Hoi S, Ng W. Cascade RSVM in peer-to-peer networks. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, Vol. 5211, Daelemans W, Goethals B, Morik K (eds), Lecture Notes in Computer Science. Springer: Berlin Heidelberg, 2008; 55–70, DOI: 10.1007/978-3-540-87479-9_22.
37. Datta S, Bhaduri K, Giannella C, Wolff R, Kargupta H. Distributed data mining in peer-to-peer networks. *IEEE Internet Computing* 2006; **10**(4):18–26. DOI: 10.1109/MIC.2006.74.
38. Siersdorfer S, Sizov S. Automatic document organization in a P2P environment. In *Advances in Information Retrieval*, Vol. 3936, Lalmas M, MacFarlane A, Rüger S, Tombros A, Tsikrika T, Yavlinsky A (eds), Lecture Notes in Computer Science. Springer: Berlin Heidelberg, 2006; 265–276, DOI: 10.1007/11735106_24.
39. Hensel C, Dutta H. GADGET SVM: a gossip-based sub-gradient svm solver. *International Conference on Machine Learning (ICML), Numerical Mathematics in Machine Learning Workshop*, Montreal, 2009.
40. Widrow B, Hoff ME. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*. IRE: New York, 1960; 96–104.
41. Bauer E, Kohavi R. An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning* 1999; **36**(1):105–139. DOI: 10.1023/A:1007515423169.
42. PeerSim. http://peersim.sourceforge.net/.
43. Guyon I, Hur AB, Gunn S, Dror G. Result analysis of the nips 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17*. MIT Press: Cambridge, 2004; 545–552.
44. Frank A, Asuncion A. UCI machine learning repository, 2010.
45. Stutzbach D, Rejaie R. Understanding churn in peer-to-peer networks. In *Proceedings 6th ACM Conference on Internet Measurement (IMC'06)*. ACM: New York, 2006; 189–202, DOI: 10.1145/1177080.1177105.
46. Roozenburg J. Secure decentralized swarm discovery in Tribler. *Master's Thesis*, Parallel and Distributed Systems Group, Delft University of Technology, 2006.
47. McGrew DA, Fluhrer SR. Multiple forgery attacks against message authentication codes. *IACR Cryptology ePrint Archive* 2005; **2005**:161.