Asynchronous Peer-to-peer Data Mining with Stochastic Gradient Descent*

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

[1] University of Szeged, Hungary
{ormandi,ihegedus}@inf.u-szeged.hu
[2] University of Szeged and Hungarian Academy of Sciences, Hungary
jelasity@inf.u-szeged.hu

Abstract. Fully distributed data mining algorithms build global models over large amounts of data distributed over a large number of peers in a network, without moving the data itself. In the area of peer-to-peer (P2P) networks, such algorithms have various applications in P2P social networking, and also in trackerless BitTorrent communities. The difficulty of the problem involves realizing good quality models with an affordable communication complexity, while assuming as little as possible about the communication model. Here we describe a conceptually simple, yet powerful generic approach for designing efficient, fully distributed, asynchronous, local algorithms for learning models of fully distributed data. The key idea is that many models perform a random walk over the network while being gradually adjusted to fit the data they encounter, using a stochastic gradient descent search. We demonstrate our approach by implementing the support vector machine (SVM) method and by experimentally evaluating its performance in various failure scenarios over different benchmark datasets. Our algorithm scheme can implement a wide range of machine learning methods in an extremely robust manner.

1 Introduction

Data aggregation has long been considered an important aspect of a peer-to-peer (P2P) system. In the past decade, an extensive literature has accumulated on the subject. Research has mainly focused on very simple statistics over fully distributed databases, such as the average of a distributed set of numbers [18, 15], separable functions [22], or network size [20]. General SQL queries have also been implemented in this fashion [25]. The main attraction of the known fully distributed (mostly gossip-based) algorithms for data aggregation is their impressive simplicity and efficiency, combined with robustness to benign failure.

* in Proc. Euro-Par 2011, Springer LNCS 6852, pp 528–540, doi:10.1007/978-3-642-23400-2_49. 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).

Simple statistics or queries are very useful, but often more is needed. For example, for a P2P platform that offers rich functionality to its users including spam filtering, personalized search, and recommendation [24, 1, 3], or for P2P approaches for detecting distributed attack vectors [5], complex predictive models have to be built based on fully distributed, and often sensitive, data. At the same time, it would be highly desirable to build these models without sacrificing any of the nice properties of the aggregation algorithms mentioned above.

In sum, we need to find fully distributed, efficient, and lightweight data mining algorithms that make no or minimal assumptions about the synchrony and reliability of communication, work on fully distributed datasets without collecting the data to a central location, and make the learned models available to all participating nodes. Our contribution is that we propose a method based on stochastic gradient search that meets these requirements. In stochastic gradient search, the model of the data is gradually evolved as it is exposed to random records from the training dataset. A wide range of models—including artificial neural networks, and support vectors—can be evolved in this fashion. Stochastic gradient methods can naturally be implemented in a gossip fashion, where models perform a random walk over the network, while converging to an optimal model. Furthermore, we can even improve the performance of sequential stochastic gradient methods, exploiting the fact that there are many interacting models making random walks at the same time.

2 System and Data Model

We assume that the system consists of a potentially very large number of nodes, typically personal computing devices such as PCs or mobile devices. Every node has a network address. Every node can send messages to every other node, provided the address of the target node is available. We assume that messages can have arbitrary delays, and messages can be lost as well. In addition, nodes can join and leave at any time without warning, thus leaving nodes and crashed nodes are treated identically. Leaving nodes can join again, and while offline, they may retain their state information.

The only middleware service our algorithm relies on is the peer sampling service [16]. Through this service, each node can request uniform random samples of the nodes in the network that are likely to be online at the time of the request. The API of the service consists of a local function GETRANDOMPEER(), which returns a random node address. Many implementations of the peer sampling service are known. In this paper we apply the NEWSCAST protocol, a gossip based implementation [16]. The overhead of NEWSCAST consists of sending one message of a constant size to a random node periodically. This protocol has been extended to deal with uneven request rates at different nodes, as well as uneven distributions of message drop probabilities [30].

As for the data distribution model, we assume that each node stores exactly one data record. These records are of the same type (contain the local values of the same features) at each node. This extreme distribution model allows us to

support applications that require extreme privacy where, for example, the profile of a user never leaves the computer of the user.

3 Background

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, i.e. $(x_1, y_1), \ldots, (x_n, y_n)$, 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.

Stochastic gradient search is one such generic search algorithm. 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 [8].

Let us now turn to support vector machines (SVM), the learning algorithm we apply in this paper [6]. 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 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.

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

The Pegasos algorithm is an SVM training algorithm, based on a stochastic gradient descent approach [27]. It directly optimizes a form of the above 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. Since 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 [6]. The dual form is

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

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 [4].

4 Related Work

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. Apart from the related work mentioned in the Introduction, we focus here on fully distributed data mining algorithms.

We divide fully distributed data mining algorithms into two main groups. In the first group we can find approaches that do not build models or that build minimal models like the unsupervised learners [28] or the collaborative filtering based recommender algorithms [23, 2, 12, 31]. These types of approaches mainly use other well-studied P2P services like aggregation [18, 15] with perhaps some kind of overlay support like T-Man [14]. We stress that these algorithms do not implement optimization or generative probability modeling like most of the state-of-the-art machine learning algorithms do.

In the second group there are algorithms that do build models, but require services such as round-based synchronization, and other reliability assumptions (e.g. [7]). As for SVM algorithms, we are aware of only one comparable P2P SVM

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p
6: procedure onReceiveModel(m)
7:     m ← updateModel(m)
8:     currentModel ← m
9:     modelQueue.add(m)
```

implementation called Gadget SVM [13]. This algorithm applies the Push-Sum algorithm [18], but it requires round synchronization as well.

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

5 The Algorithm

The skeleton of the algorithm we propose is shown in Algorithm 1. This algorithm is run by every node in the network. When joining the network, each node generates a model via initModel(). After the initialization each node starts to periodically send its current model to a random neighbor that is selected using the peer sampling service (see Section 2). When receiving the model, the node updates it using a stochastic gradient descent step based on the training sample it stores, and subsequently it stores the model. The model queue can be used for voting, as we will explain later.

Recall that we assumed that each node stores exactly one training sample. This is a worst case scenario; if more samples are available locally, then we can use them all to update the model without any network communication, thus speeding up convergence.

In this skeleton, we do not specify what kind of models are used and what algorithms operate on them. For example, a model is a $d-1$ dimensional hyperplane in the case of SVM, as described earlier, which can be characterized by a d dimensional real vector. In other learning paradigms other model types are possible. To instantiate the framework, we need to implement initModel() and updateModel(). This can be done based on any learning algorithm that utilizes the stochastic gradient descent approach. In this paper we will focus on the Pegasos algorithm [27], which implements the SVM method. The two procedures are shown in Algorithm 2.

We assume that the model m has two fields: $m.t \in \mathbb{N}$, which holds the number of times the model was updated, and $m.w \in \mathbb{R}^d$ that holds the linear model. The parameter $\lambda \in \mathbb{R}$ is the learning rate. In our experiments we used the setting $\lambda = 10^{-4}$. Vector $x \in \mathbb{R}^d$ is the local feature vector at the node, and $y \in \{-1, 1\}$ is its correct classification. The operator $< \cdot, \cdot >$ calculates the inner product. Line 4 gets executed if the local example x is misclassified by the model $m.w$.

The effect of the algorithm will be that the models will perform a random walk in the network while being updated using the update rule of the Pegasos

Algorithm 2 P2Pegasos

```
1: procedure UPDATEMODEL(m)
2:     η ← 1/(λ · m.t)
3:     if y ⟨m.w, x⟩ < 1 then
4:         m.w ← (1 − ηλ)m.w + ηyx
5:     else
6:         m.w ← (1 − ηλ)m.w
7:     m.t ← m.t + 1
8:     return m
9: procedure INITMODEL
10:    m.t ← 0
11:    m.w ← (0, . . . , 0)^T
12:    send model(m) to self
```

Algorithm 3 P2Pegasos prediction procedures

```
1: procedure PREDICT(x)
2:     w ← currentModel
3:     return sign(⟨w, x⟩)
4: procedure VOTEDPREDICT(x)
5:     pRatio ← 0
6:     for m ∈ modelQueue do
7:         if sign(⟨m.w, x⟩) ≥ 0 then
8:             pRatio ← pRatio +1
9:     return sign(pRatio/modelQueue.size()−0.5)
```

algorithm. In this sense, each model corresponds to an independent run of the sequential Pegasos, hence the theoretical results of the Pegasos algorithm are applicable. Accordingly, we know that all these models will converge to an optimal solution of the SVM primal optimization problem [27]. For the same reason, the algorithm does *not need any synchronization or coordination.* Although we do not give a formal discussion of asynchrony, it is clear that as long as each node can contact at least one new uniform random peer in a bounded time after each successful contact, the protocol will converge to the optimal solution.

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, since there are N models in the network (where N is the network size), we can apply additional techniques to achieve a higher predictive performance than that of an output model of a simple sequential implementation. Here we implement a simple voting mechanism, where nodes will use more than one model to make predictions. Algorithm 3 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. We note, that MODELQUEUE is assumed to be of a bounded size. When storing a new model in it, an old one will be removed if the queue is full. In our experiments we used a queue implementation, where the queue holds the 10 latest added models.

6 Experimental results

We selected data sets of different types including small and large sets containing a small or large number of features. Our selection includes the commonly used

Table 1. The main properties of the data sets, and the prediction error of the baseline sequential algorithms.

	Iris1	Iris2	Irirs3	Reuters	SpamBase	Malicious10
Training set size	90	90	90	2000	4140	2155622
Test set size	10	10	10	600	461	240508
Number of features	4	4	4	9947	57	10
Classlabel ratio	50/50	50/50	50/50	1300/1300	1813/2788	792145/1603985
Pegasos 20000 iter.	0	0	0	0.025	0.111	0.080 (0.081)
Pegasos 1000 iter.	0	0	0.4	0.057	0.137	0.095 (0.060)
SVMLight	0	0	0.1	0.027	0.074	0.056 (–)

Fiser's Iris data set [9]. The original data set contains three classes. Since the SVM method is designed for the binary (two-class) classification problem, we transformed this database into three two-class data sets by simply removing each of the classes once, leaving classes 1 and 2 (Iris1), classes 1 and 3 (Iris2), and classes 2 and 3 (Iris3) in the data set. In addition, we included the Reuters [11], the Spambase, and the Malicious URLs [19] data sets as well. All the data sets were obtained from the UCI database repository [10]. Table 1 shows the main properties of these data sets, as well as the prediction performance of the baseline algorithms. SVMLight [17] is an efficient SVM implementation. Note that the Pegasos algorithm can be shown to converge to the same value as SVMlight [27].

The original Malicious URLs data set has about 3,000,000 features, hence we first reduced the number of features so that we could carry out simulations. 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 used a simple and well-known method, namely we calculated the correlation coefficient of 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 [15], but we performed it offline. The effect of this dramatic reduction on the prediction performance is shown in Table 1, where the results of Pegasos on the full feature set are shown in parentheses (SVMlight could not be run due to the large size of the database).

6.1 Scenarios

The experiments were carried out in the event based engine of the PeerSim simulator [21]. The peer sampling service was provided by the NewsCast protocol. The network size is the same as the database size; each node has exactly one sample. Each node starts running the protocol at the same time. The protocol does not require a synchronized startup, but we need it here to analyze convergence in a clearly defined setting.

In our experimental scenarios we modeled message drop, message delay, and churn. The drop probability of each message was 0.5. This can be considered an extremely large drop rate. Message delay was modeled as a uniform random delay from the interval $[\Delta, 10\Delta]$, where Δ is the gossip period, as shown in

Algorithm 1. This is also an extreme delay, which is orders of magnitudes higher than what can be expected in a realistic scenario.

We also modeled realistic churn based on probabilistic models in [29]. Accordingly, we approximated the online session length with a lognormal distribution, and we approximated the parameters of the distribution using a maximum likelihood estimate based on a trace from a private BitTorrent community called FileList.org, obtained from Delft University of Technology [26]. We set the offline session lengths so that at any moment in time 90% of the peers were online. In addition, we assumed that when a peer came back online, it retained its state that it had at the time of leaving the network. We now list the scenarios we experimented with: *No failure*: there is no message drop, no delay and no churn; *Drop only*: we simulate message drop as described, but no other types of failure; *Delay only*: we simulate message delay only; *Churn only*: we simulate node churn only; *All failures*: we apply message drop, delay and churn at the same time.

6.2 Metrics

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 ratios of this splitting are shown in Table 1. At a given point in time, we select 100 peers at random (or all the peers, if there are fewer than 100) and we calculate the average misclassification ratio of these 100 peers over the test set using the current models of the 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.

Moreover, we calculated the similarities between the models circulating in the network using the cosine similarity measure. This was done only for the Iris databases, where we calculated the similarity between all pairs of models, and calculated the average. This metric is useful for studying the speed at which the actual models converge. Note that under uniform sampling it is known that all models converge to an optimal model.

6.3 Results

Figure 1 shows the results over the Iris datasets for algorithm variants that do not apply voting for prediction. The plots show results as a function of cycles. One cycle is defined as a time interval of one gossip period Δ. Although the size of each data set is the same, the dynamics of the convergence are rather different. The reason is that the learning complexity of a database depends primarily on the inner structure of the patterns of the data, and not on the size of data set. In trivially learnable patterns a few examples are enough to construct a good model, while under complex patterns a large number of samples as well as many iterations might be required. Since Pegasos also has a similar convergence behavior, we can be sure that this is not an artifact of parallelization.

Let us now turn to the analysis of the individual effects of the different failures we modeled, comparing them to two baseline algorithms. The first baseline

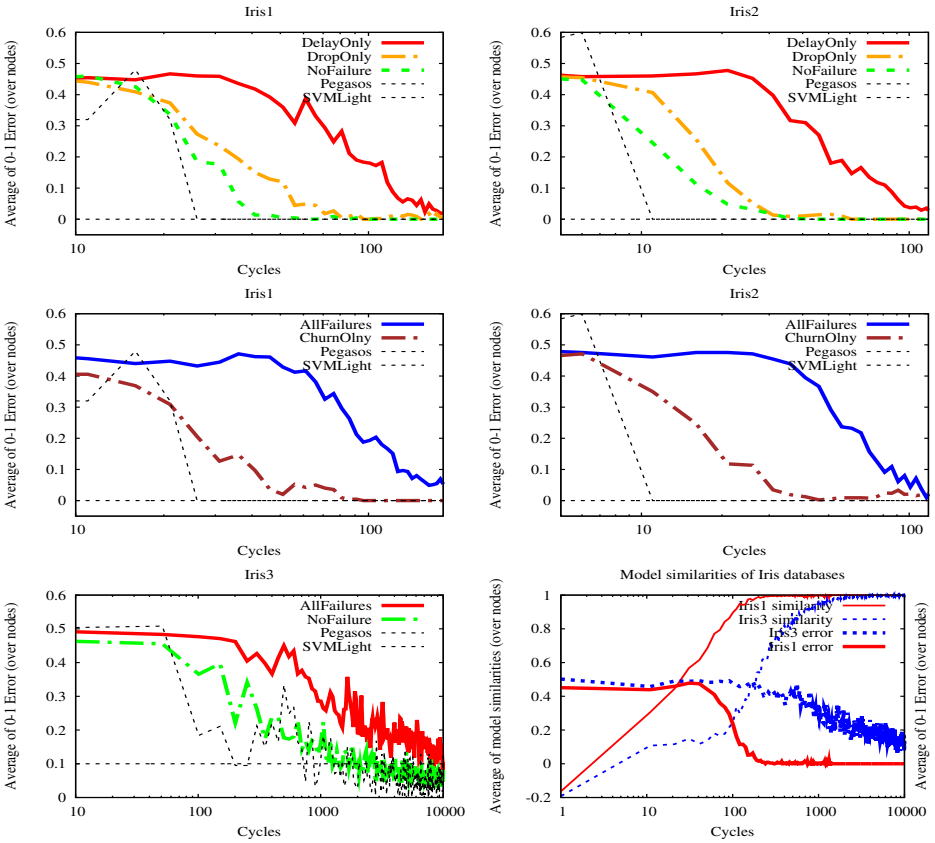

Fig. 1. Experimental results over the Iris databases

algorithm is SVMLight, a sequential efficient SVM solver [17] that optimizes the dual SVM problem given in (2). It is independent of the cycles, hence its performance is shown as a horizontal line. The second baseline algorithm is Pegasos. We ran Pegasos 100 times, and show the average error at each cycle. Note that for Pegasos each cycle means visiting another random teaching example.

Clearly, the best performance is observed under no failure. This performance is very close to that of Pegasos, and converges to SVMlight (like Pegasos does). The second best performance is observed with churn only. Adding churn simply introduces an extra source of delay since models do not get forgotten as mentioned in 6.1. The situation would be different in an adaptive scenario, which we do not consider here. In the scenario with message drop only, the performance is still very close to the ideal case. Considering the extremely large drop rates, this result is notable. This extreme tolerance to message drop comes from the fact that the algorithm is fully asynchronous, and a 25% drop rate on average causes only at most a proportional slowdown of the convergence. Among the individual failure types, extreme message delay is the most significant factor. On average, each message takes as much as 5 cycles to reach its destination. The resulting

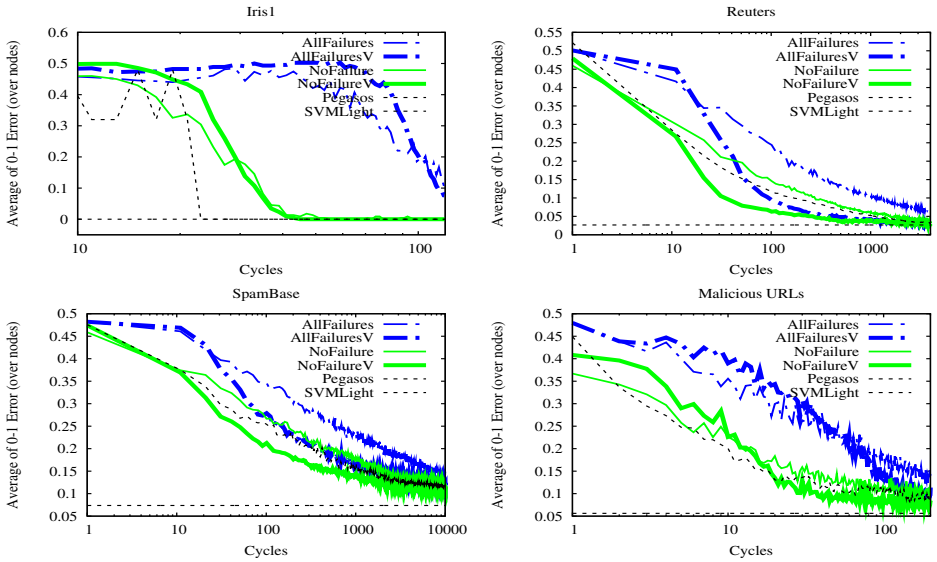

Fig. 2. Experimental results over the large databases, and the Iris1 database. Labels marked with a 'V' are variants that use voting.

slowdown is less then a factor of 5, since some messages do get through faster, which speeds up the convergence of the prediction error.

In Figure 1 we also present the convergence of the averaged cosine similarities over the nodes together with their prediction performance under no failures, without voting. We can see that in the case of each data set the models converge, so the observed learning performance is due to good models as opposed to random influences.

Although, as mentioned above, in our case convergence speed depends mainly on data patterns, and not on the database size, to demonstrate scalability we performed large scale simulations as well with our large data sets. The results can be seen in Figure 2. Here we plotted just the two scenarios with no failures and with all the failures. The figure also shows results for the variants that use voting. A general observation regarding the distinction between the P2Pegasos variants with and without voting is that voting results in a better performance in all scenarios, after a small number of cycles. In the first few cycles, the version without voting outperforms voting because there is insufficient time for the queues to be filled with models that are mature enough. On some of the databases the improvement due to voting can be rather dramatic. We note that where the test data sets were larger (see Table 1) we obtained smoother convergence curves.

7 Conclusions

In this paper we have proposed a generic framework for fully distributed data mining, which implements stochastic gradient search. Nodes in the network gossip models that are continuously updated at each node along their random walk.

We experimented with an instantiation of this framework using the Pegasos algorithm, that is a stochastic gradient descent implementation of the SVM method. Our main conclusion is that the approach is able to produce SVM models in a very hostile environment, with extreme message drop rates and delays, with very limited assumptions about the communication network. The only service that is needed is uniform peer sampling. The quality of the models are very similar to that of the sequential Pegasos algorithm. Furthermore, we can also outperform Pegasos with the help of a voting technique that makes use of the fact that there are many independent models in the network passing through each node. The models are available at each node, so all the nodes can perform predictions as well. At the same time, nodes never reveal their data, so this approach is a natural candidate for privacy preserving solutions.

References

1. Bai, X., Bertier, M., Guerraoui, R., Kermarrec, A.M., Leroy, V.: Gossiping personalized queries. In: Proc. 13th Intl. Conf. on Extending Database Technology (EBDT'10) (2010)
2. Bakker, A., Ogston, E., van Steen, M.: Collaborative filtering using random neighbours in peer-to-peer networks. In: Proc. 1st ACM Intl. workshop on Complex networks meet information & knowledge management (CNIKM'09). pp. 67–75. ACM (2009)
3. Buchegger, S., Schiöberg, D., Vu, L.H., Datta, A.: PeerSoN: P2P social networking: early experiences and insights. In: Proc. Second ACM EuroSys Workshop on Social Network Systems (SNS'09). pp. 46–52. ACM (2009)
4. Chapelle, O.: Training a support vector machine in the primal. Neural Computation 19, 1155–1178 (May 2007)
5. Cheetancheri, S.G., Agosta, J.M., Dash, D.H., Levitt, K.N., Rowe, J., Schooler, E.M.: A distributed host-based worm detection system. In: Proc. 2006 SIGCOMM workshop on Large-scale attack defense (LSAD'06). pp. 107–113. ACM (2006)
6. Cristianini, N., Shawe-Taylor, J.: An introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press (2000)
7. Datta, S., Giannella, C., Kargupta, H.: Approximate distributed k-means clustering over a peer-to-peer network. IEEE Trans. on Knowl. and Data Eng. 21, 1372–1388 (October 2009)
8. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley-Interscience, second edn. (2000)
9. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annals of Eugenics 7(7), 179–188 (1936)
10. Frank, A., Asuncion, A.: UCI machine learning repository (2010)
11. Guyon, I., Hur, A.B., Gunn, S., Dror, G.: Result analysis of the nips 2003 feature selection challenge. In: Advances in Neural Information Processing Systems 17. pp. 545–552. MIT Press (2004)
12. Han, P., Xie, B., Yang, F., Wang, J., Shen, R.: A novel distributed collaborative filtering algorithm and its implementation on p2p overlay network. In: Dai, H., Srikant, R., Zhang, C. (eds.) Advances in Knowledge Discovery and Data Mining, LNCS, vol. 3056, pp. 106–115. Springer (2004)

13. Hensel, C., Dutta, H.: GADGET SVM: a gossip-based sub-gradient svm solver. In: Intl. Conf. on Machine Learning (ICML), Numerical Mathematics in Machine Learning Workshop (2009)
14. Jelasity, M., Babaoglu, O.: T-Man: Gossip-based overlay topology management. In: Brueckner, S.A., Di Marzo Serugendo, G., Hales, D., Zambonelli, F. (eds.) Engineering Self-Organising Systems: Third Intl. Workshop (ESOA 2005), Revised Selected Papers. LNCS, vol. 3910, pp. 1–15. Springer (2006)
15. Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Transactions on Computer Systems 23(3), 219–252 (2005)
16. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems 25(3), 8 (2007)
17. Joachims, T.: Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods - Support Vector Learning, chap. 11, pp. 169–184. MIT Press (1999)
18. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03). pp. 482–491. IEEE Computer Society (2003)
19. Ma, J., Saul, L.K., Savage, S., Voelker, G.M.: Identifying suspicious urls: an application of large-scale online learning. In: Proc. 26th Annual Intl. Conf. on Machine Learning. pp. 681–688. ICML '09, ACM (2009)
20. Massoulié, L., Merrer, E.L., Kermarrec, A.M., Ganesh, A.: Peer counting and sampling in overlay networks: random walk methods. In: Proc. 25th annual ACM symposium on Principles of distributed computing (PODC). pp. 123–132. ACM (2006)
21. Montresor, A., Jelasity, M.: Peersim: A scalable P2P simulator. In: Proc. 9th IEEE Intl. Conf. on Peer-to-Peer Computing (P2P 2009). pp. 99–100. IEEE (September 2009), extended abstract
22. Mosk-Aoyama, D., Shah, D.: Fast distributed algorithms for computing separable functions. IEEE Transactions on Information Theory 54(7), 2997–3007 (2008)
23. Ormándi, R., Hegedűs, I., Jelasity, M.: Overlay management for fully distributed user-based collaborative filtering. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010. LNCS, vol. 6271, pp. 446–457. Springer (2010)
24. Pouwelse, J.A., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D.H.J., Reinders, M., van Steen, M.R., Sips, H.J.: TRIBLER: a social-based peer-to-peer system. Concurrency and Computation: Practice and Experience 20(2), 127–138 (2008)
25. van Renesse, R., Birman, K.P., Vogels, W.: Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Transactions on Computer Systems 21(2), 164–206 (May 2003)
26. Roozenburg, J.: Secure Decentralized Swarm Discovery in Tribler. Master's thesis, Parallel and Distributed Systems Group, Delft University of Technology (2006)
27. Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A.: Pegasos: primal estimated sub-gradient solver for SVM. Mathematical Programming B (2010)
28. Siersdorfer, S., Sizov, S.: Automatic document organization in a p2p environment. In: Lalmas, M et al. (ed.) Advances in Information Retrieval, LNCS, vol. 3936, pp. 265–276. Springer (2006)
29. Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: Proc. 6th ACM Conf. on Internet measurement (IMC'06). pp. 189–202. ACM (2006)
30. Tölgyesi, N., Jelasity, M.: Adaptive peer sampling with newscast. In: Sips, H., Epema, D., Lin, H.X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 523–534. Springer
31. Tveit, A.: Peer-to-peer based recommendations for mobile commerce. In: Proc. 1st Intl. workshop on Mobile commerce (WMC '01). pp. 26–29. ACM (2001)