Detecting Concept Drift in Fully Distributed Environments

István Hegedűs[*], Lehel Nyers[†] and Róbert Ormándi[*]
[*]*University of Szeged, Szeged, Hungary*
{ihegedus,ormandi}@inf.u-szeged.hu
[†]*University of Szeged, Szeged, Hungary and Subotica Tech, Subotica, Serbia*
lehel@vts.su.ac.rs

***Abstract*—Applying sophisticated machine learning techniques on fully distributed data is increasingly important in many applications like distributed recommender systems or spam filters. In this type of networked environment the data model can change dynamically over time (concept drift). Identifying when concept drift occurred is a key for several drift handling techniques and important in numerous scenarios. However drift handling approaches exist, no efficient solution for detecting the drift is known in very large scale networks. Here, we propose an approach that can detect the concept drift in large scale and fully distributed networks. In our approach, the learning is performed by applying online learners that take random walks in the network while updating themselves using the samples available at the nodes. The drift detection is based on an adaptive mechanism which uses the historical performances of the models. Through empirical evaluations we demonstrate that our approach handles the drifting concept while additionally detects the occurrence of the concept drift with high accuracy.**

***Keywords*-adaptive classification; concept drift; P2P**

I. Introduction

We are in the middle of big data era. Our devices and applications accumulate more and more data. Smart phones become easily programmable personal sensing devices with accelerometer, gyroscope and GPS, having the capability to communicate through the Internet all the time [1], [2]. The pure P2P file sharing applications are getting more and more social making possible to produce and share high quality content [3], [4]. Paradoxically while getting access to more and more data, we can find less information by applying the original retrieval techniques. This phenomena increases the importance of emerging fully distributed, large-scale data mining algorithms that can work in unreliable networks, take the privacy of users into account and be adaptive to changes in the data.

The above-mentioned applications are usually prepared for long-term running. In this type of applications, it is crucial that the machine learning algorithms work adaptively, since the behavior of the users and the trends—that appears in the data—change continuously.

In this work we are interested in scenarios in which a huge number of connected nodes *learn* global models (e.g. classifiers) by applying local communications from continuously changing data. We assume that the underlying data model changes and that only a limited amount of data becomes available for identifying and following concept drift. These assumptions are quite common in the systems mentioned above.

Previously we proposed the so-called Gossip Learning Framework (GoLF) [5], a general learning framework for performing large-scale fully distributed learning. The basic approach is not designed for dealing with changing data, but recently we extended it by adding drift handling capability to the framework [6]. That approach introduces adaptivity through keeping the model diversity by managing the lifetime of the models without being able to *detect* the concept drift.

Our current contribution is the following. We propose a drift detecting mechanism for GoLF that makes it possible to identify the time moment when the concept changed while the method handles it as well. It is important to highlight that our previous work on drift handling [6] is a quite different approach which completely ignores the drift detection mechanism (proposed here). Our current proposal can be suitable in scenarios in which the drift detection mechanism is important.

II. System and Data Model

As our system model, we consider a network of computers where each node can communicate with any other node by messages if the address of the target node is locally available. We assume that a peer sampling service exists which can provide addresses of uniform randomly selected nodes from the network.

In our data model, the database is horizontally distributed over the whole network. Additionally we assume that each peer in the network has just only one data record which excludes the local statistical processing. Another key assumption is that the record never leaves the node; collecting the data to a central server is not allowed due to privacy considerations.

III. Background

A. Supervised Learning

The main problem of the supervised binary classification can be defined as follows. We have a manually labeled training database $S = \{(x_1, y_1), \ldots, (x_l, y_l)\} \in \mathbb{R}^d \times \{-1, +1\}$. Here $x_i \in \mathbb{R}^d$ called *feature vector* that describes an object of real world (e.g. the content of a textual comment) as a real-valued vector and y_i is the *class label* which assigns the

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).

feature vector to a well-defined class (e.g. spam). We assume that all the samples are generated by an unknown underlying probability distribution $\mathcal{D}$. The goal—through the learning phase—of the classification problem is to find a model $f : \mathbb{R}^d \to \{-1, +1\}$ which can classify *any* sample coming from the same probability distribution ($\mathcal{D}$). We expect that the model has to classify unseen examples too i.e. it has to generalize well. When the training samples are available as a stream, the training process is known as *online learning*.

B. Concept Drift

The distribution that generates the training databases ($\mathcal{D}$) may change over the time. This change—especially when it is significant and sudden—makes the already trained models inaccurate. This phenomena is called *concept drift*. The main goal of *concept drift handling* is to design algorithms that provide a good model f_t corresponding to any time moment t in which the actual data distribution is $\mathcal{D}_t$. However, in some scenarios more is needed i.e. we have to identify the time moment t^* when (sudden) drift occurred while we also handle it. This problem is known as the *drift detection* problem.

C. Gossip Learning Framework

The Gossip Learning Framework (GoLF) [5] is a learning framework designed for performing fully distributed learning in large scale networks. The basic idea behind GoLF is that large number of online models take random walks in the network while improving themselves using the training samples contained at the nodes and getting combined by applying ensemble learning techniques.

In Algorithm 1, the skeleton of GoLF is shown extended with an additional line (marked with the comment "Drift Detection") which calls the proposed drift detection subroutine. Here, we explain the original parts of GoLF only; the drift detecting approach is detailed in Section V.

At each peer in the network, the same protocol runs which consists of an active loop of periodic activity, and the message handler method ONRECEIVEMODEL to process incoming models. This method updates the received model using the locally stored training example and stores it in its *cache* called *receivedModels*. In the active loop, the stored models are sent to randomly selected neighbors (proposed by the peer sampling service) and are deleted from the cache. At anytime the freshest model (that is, CURRENTMODEL, the model added to the local cache most recently) is used for performing predictions. This means that no communication is needed for performing predictions. The concrete learning algorithm is implemented in the method UPDATEMODEL which is detailed in Section V.

In the GoLF protocol, we make no assumptions about either the synchrony of the loops at the different nodes or the reliability of the messages. It is assumed only that the distribution of the length of active period, denoted by Δ in the algorithms and modeled with random variable $\mathcal{N}(\Delta, \Delta/10)$, is the same at each node. There is no explicit failure detection mechanism; however, if a node does not receive any models for a certain number of periods (in our case 10), then it will assume that the number of models circulating in the network has decreased, and will send its CURRENTMODEL to a random neighbor. This prevents the network from running out of models due to message drop failures.

IV. Related Work

A. Non-Distributed Concept Drift Handling

The set of non-distributed approaches available in the literature is quite large. The early approaches for handling concept drift applied chunk based learning [7], [8], [9] i.e. they learned a new model when a fixed sized sample set (chunk) became available and discarded the previously learned model. These approaches produce good performance when the sample generation speed is faster relative to the speed of concept drift. In these types of approaches, no drift detection mechanism is introduced.

More sophisticated methods introduce various drift detection methods [8], [10], [11]. These approaches use performance related measures to investigate whether building a new model has to be initiated. The recent approaches apply ensemble techniques as well to improve their performance [12]. Here, the old models are not be discarded instead they are added to an ensemble pool. The models of this pool are often weighted and they are used for making predictions.

B. Handling Concept Drift in Fully Distributed Environment

Addressing machine learning in a fully distributed environment is a relatively new but growing area. Some algorithms were introduced in [13], [14], [15], [16], [17], [18], [19], [20], [5]. However, very few approaches was proposed to tackle concept drift in a fully distributed environment.

In a recent approach proposed by Ang et al. [15], a drift detection (reactive behavior) and simultaneously a drift prediction (proactive behavior) mechanisms were introduced. The basic idea behind the approach is the chunk-based technique extended with a triggering and an ensemble based aspects. The evaluations show that the proposed approach is suitable in many drifting scenarios, but the communication cost of the method is extremely high. Later a number of heuristics were proposed to decrease this cost.

A quite different yet general, gossip-based drift handling technique was introduced in our previous work [6]. This approach completely ignored the drift detection; instead it takes advantage of the large number of models found in the network and maintain a diverse pool of models by managing the age distribution of the models in the pool. Additionally—based on the mechanism proposed by GoLF—the models continuously take random walks in the network, since they can learn from new training samples. This mechanism is crucial in those scenarios in which the sample generation speed per node is low, since in the whole network there should be enough samples for learning.

Algorithm 1 CDDGoLF

```
$c \leftarrow 0$
currentModel $\leftarrow$ initModel()
receivedModels.add(currentModel)
loop
    if receivedModels = $\emptyset$ then
        $c \leftarrow c + 1$
    if $c = 10$ then
        receivedModels.add(currentModel)
    for all $m \in$ receivedModels do
        $p \leftarrow$ selectPeer()
        send $m$ to $p$
        receivedModels.remove($m$)
        $c \leftarrow 0$
    wait($\Delta$)

procedure onReceiveModel($m$)
    $m \leftarrow$ driftHandler($m$)                ▷ Drift Detection
    currentModel $\leftarrow$ updateModel($m$)
    receivedModels.add(currentModel)
```

V. Algorithm

In our proposal, referred as CDDGoLF, we extend the original GoLF protocol described in Section III-C with concept drift detection capability. This extension is attached to the original GoLF through two modifications. First, we added a method call at the line 16 in Alg. 1. This method, called DRIFTHANDLER, is responsible for detecting drift. Second, we extended the models with a bounded size queue, called *history*, that stores some performance related data from the model based on the previously seen examples.

The main idea behind our concept drift detection algorithm is that we can use the samples stored at the nodes for evaluating the models before we use them for training. Then using the results got from this evaluation we can properly characterize the models, i.e. we can decide whether the concept drifted.

Concretely when a node receives an incoming model it calls the method DRIFTHANDLER at line 16 in Alg. 1. This subroutine is presented at Alg. 2. First, it uses the locally stored sample as a test sample and evaluates the model (line 7). Then it updates the model history by storing the error score (value 1 if the predicted and real class label are different; 0 otherwise) measured on the locally stored sample (line 8). Applying this technique we can accumulate a bounded size series of independent error scores in the history. We have to take two important notes about the history. First, the error sequence stored in the history is biased in the sense that the model we measure is continuously changing while we collect the error scores. But we expect that this bias will not cause any huge failure for characterizing the models. Second, we have to use a bounded size queue to keep the message size constant, since the history is a part of the model and it is sent through the network. In our case we used a history size of 100.

Algorithm 2 Procedures

```
procedure initModel
    $m.w \leftarrow (0, 0, \ldots)^T$
    $m.age \leftarrow 0$
    $m.history \leftarrow \emptyset$
    return $m$

procedure driftHandler($m$)
    $\hat{y} \leftarrow m$.predict($x$)
    $m.history$.add($\hat{y} = y$ ? 0 : 1)          ▷ history update
    if driftOccurred($m$) then
        $m \leftarrow$ initModel()
    return $m$

procedure driftOccurred($m$)
    $errorRates \leftarrow$ smooth($m.history$)
    $slope \leftarrow$ linearReg($errorRates$)
    if rand([0; 1]) < $\sigma_{c,d}(slope)$ then
        return $true$
    return $false$

procedure updateModel($m$)
    $m.age \leftarrow m.age + 1$
    $\hat{y} \leftarrow m$.predict($x$)           ▷ $(x, y)$ is stored locally
    $m.w \leftarrow (1 - \frac{1}{m.age})m.w + \frac{1}{m.age \cdot \lambda}(y - \hat{y})x$
    return $m$

procedure predict($x$)
    $p_0 \leftarrow 1/(1 + exp(\text{currentModel}.w^T x))$
    $p_1 \leftarrow 1 - p_0$
    return $p_0 > p_1$ ? 0 : 1
```

After the history was updated, the DRIFTOCCURRED method shown in Alg. 2 is called by the DRIFTHANDLER to decide whether a concept drift occurred (line 9). In this method, an analysis of the history is performed. Concretely, first we perform a preprocessing step (line 13) by applying a sliding window based averaging with window size *history*.size()/2 on the raw data. This step is necessary for noise reduction reasons. Second, we apply linear regression [21] on the smoothed error rates (line 14) to catch the main slope of the data. Finally, using the normalized version of this slope value, we make a decision about the drift (line 15). That is, we apply a sigmoid function on the slope value (normalization) and alert concept drift with a probability proportional to this value. Here, the sigmoid function used is $\sigma_{c,d}(x) = \frac{1}{1+e^{-c(x-d)}}$, with parameters $c = 20$ and $d = 0.5$.

We have to note that this decision has a clear geometrical interpretation. That is, if the slope is negative or 0, the learning is probably in the convergence or in the already converged phase, respectively. Otherwise, if the slope is positive, drift probably occurred.

When drift occurred (i.e. the method DRIFTOCCURRED returned with *true*), we reinitialize the model by calling the INITMODEL method in the method DRIFTHANDLER the (line 10). Here, we can implement arbitrary drift handling behavior; however—since we want to investigate the effect

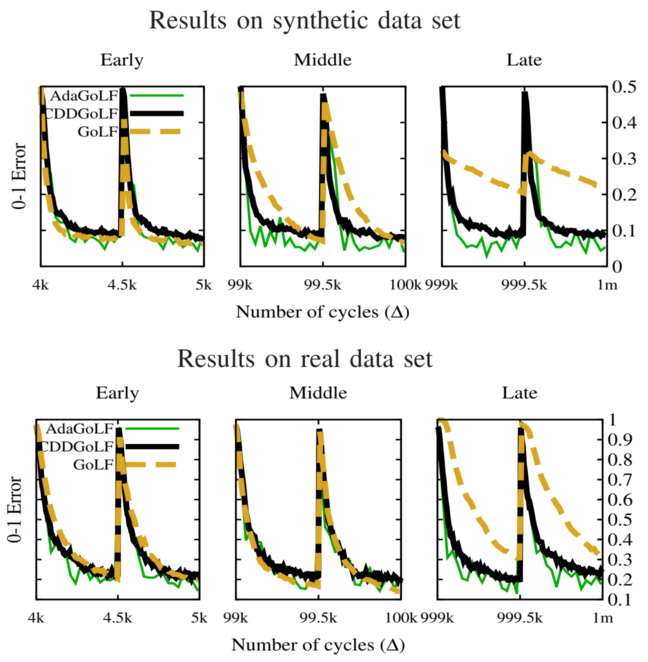

Figure 1. Adaptive methods avoid the burn in effect.

of CDDGoLF—we ignored this possibility. After the drift detection procedure, the onReceiveModel updates the previously drift handled model using the locally stored sample (line 17 in Alg. 1).

A. Online Learner

The CDDGoLF is *independent* of the applied online learner. In our evaluation we applied the Logistic Regression method [21], which is a commonly used online learner algorithm. This method seeks the parameter vector (w) that maximizes the logarithm of the conditional data likelihood

$$l(w) = \sum_{i=1}^{n} \ln P(y_i|x_i, w) - \frac{\lambda}{2}\|w\|^2, \tag{1}$$

where the (x_i, y_i) is the ith sample in the training set and the λ is regularization parameter (we used the $\lambda = 0.0001$). Applying this learner, the update rule for the parameter vector w using the training sample (x, y) can be seen in Alg. 2 (method updateModel started at the line 18). The prediction algorithm of the learner for a sample x is shown in Alg. 2 (method predict started at the line 23). The method initModel (presented in Alg. 2 started at line 1) is used to reinitialize the model when drift occurred i.e. it clears the model and its history.

VI. Results

A. Evaluation Settings

We implemented our algorithm in the PeerSim [22] P2P network simulation environment. We used the GoLF API [23] for performing evaluations and applied the Newscast [24], [25] protocol as the peer sampling service (the current implementation is part of the PeerSim Extras package).

In the experiments we used a *synthetically* generated database and a *real world* database. The synthetic database was generated by drawing uniform random points from the d dimension uniform cube (in our simulations d was set to 5). The labeling of these points is defined by a hyperplane.

Here, the drift was modeled by moving this hyperplane as a function of time according to some predefined pattern (moving hyperplane approach) [26]. The exact state of the moving hyperplane at time moment t is defined by $f_t(x) = (1-\alpha_t)f_s(x) + \alpha_t f_d(x)$, where $f_s(x)$ and $f_d(x)$ are the source and the destination hyperplanes, respectively, which are chosen randomly but kept orthogonal to each other. The value of $0 \le \alpha_t \le 1$ is defined by Eq. 2, where the concept speed is denoted by v and $[.]$ denotes the *round* function i.e. it returns the closest integer to its parameter. This selection of the drift model results a sudden change between $f_s(x)$ and $f_t(x)$ hyperplanes.

$$\alpha_t = \begin{cases} [1-(tv - \lfloor tv \rfloor)] & \text{if } \lfloor tv \rfloor \mod 2 = 1 \\ [tv - \lfloor tv \rfloor] & \text{otherwise} \end{cases} \tag{2}$$

We used the Segmentation [27] database taken from the UCI repository as well. This is a complex database describing various images with 19 high-level numeric-valued attributes. It contains 2310 samples that are classified into 7 distinct classes. We used the train-validation split proposed by the author of the database.

As a drift model, we applied a class label rotation mechanism here that is similar to the one proposed in [28], [29]. This mechanism results a variant of sudden drift.

For both databases we modeled the drift by changing the labeling over the time. The nodes get random samples iteratively from that training pool with the correct labeling i.e. with the label that the sample has in the time moment of the selection. We applied 0.1 $sample/\Delta$ sample generation speed and 0.001 $tick/\Delta$ drift speed, which defines an extremely hard scenario.

In the simulations we performed evaluations applying equidistant time step with length Δ. Here, we measured the average error rate, the so-called *0-1 error*, of the models held by the nodes on an independent validation set which was labeled corresponding to the time moment of the measurement. In the results we averaged these elementary error scores over the nodes.

B. Experimental Evaluations

1) Drift Handling: Algorithms (even the online learners) without concept drift handling capabilities show *burn in* effect, i.e. after a certain time they cannot adapt to the changing concepts. In Fig. 1, we present this phenomena on the original GoLF protocol and we also point out that CDDGoLF avoids the burn in effect. Concretely, here we demonstrate the prediction error, averaged over the network, achieved by the original GoLF (baseline), the AdaGoLF [6], and the CDDGoLF algorithms as a function of time (divided into three distinct figures: beginning, middle and end of a long-term run from left to right). It is easy to see that on both the synthetic (upper figs.) and real world

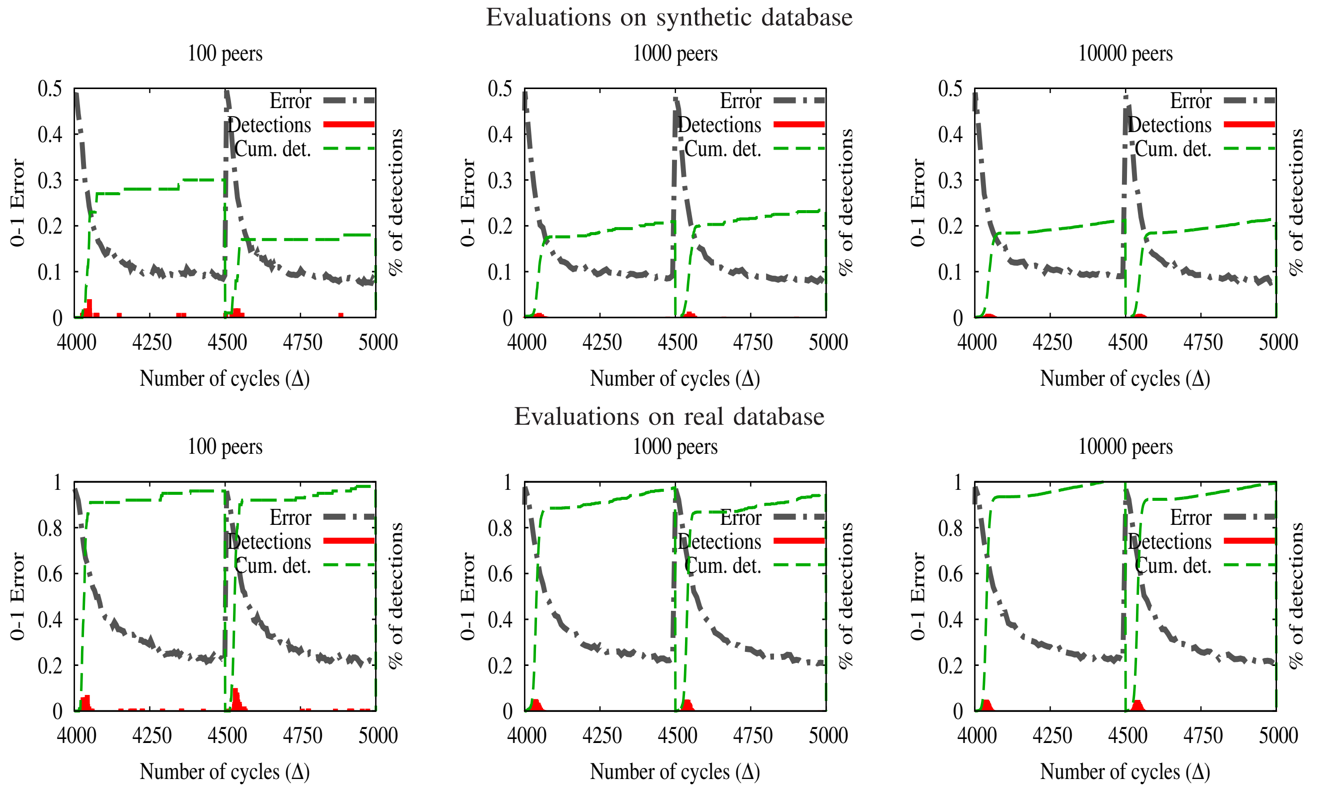

Figure 2. The drift detection and the classification performance of the proposed method on synthetic and real datasets.

database (lower figs.) the drift handling capable algorithms (AdaGoLF and CDDGoLF) do not show the burn in effect while the original protocol does. Moreover, we can see that the performance of the two adaptive algorithms (AdaGoLF and CDDGoLF) are quite similar; or perhaps the performance of the AdaGoLF is slightly better. But we note the fact that the CDDGoLF algorithm has an additional feature of detecting the drift. This minimal difference in the performance can be interpreted as the ”cost“ that we pay for this additional feature.

2) Drift Detection: In Fig. 2, we show the drift detection capability of CDDGoLF. Here, the rows represent different databases, while the columns represent different network sizes. Now we are focusing on the rows. In each figure we present the averaged (over the nodes) errors (Error), the percentage of drift detections of models (Detections) and the cumulative percentage of drift detections (Cum. det.) as a function of time. In the case of real world database, the speed of drift detections is extremely fast, i.e. the cumulative detection curve increases very quickly after drifts; while in the case of the synthetic database it is much slower. A possible reason for this could be that the synthetic database is more easily learnable, which results that the models are more robust to the change. Based on both databases, these results indicate that the CDDGoLF detects the drift *accurately* and *very quickly*.

3) Scalability: Let us now turn to the discussion of the effect of network size changes. In Fig. 2, we present the achieved results of CDDGoLF algorithm under different network sized scenarios. These scenarios are grouped in the columns of the figures, since here we are focusing on the columns now. With both database types we cannot see any significant difference in the error rate between the simulations. This observation implies that the performance of the CDDGoLF algorithm is *independent* of the network size.

VII. Conclusion

In this work we proposed a novel algorithm for detecting concept drifts in a fully distributed network on the top of our previously proposed learning framework (GoLF). The approach introduces a cache, called history, for collecting data about the performance of the models. Later in the protocol a mechanism decides whether drift occurred.

Through empirical evaluations we investigated the performance (both for drift handling and detection capabilities) and scalability of the method. Based on these, we pointed out that our current proposal is a suitable choice when we need the additional feature of drift detection, since the drift handling performance of our approach is similar to the state-of-the-art approaches.

References

[1] A. S. Pentland, “Society’s nervous system: Building effective government, energy, and public health systems,” *Computer*, vol. 45, no. 1, pp. 31–38, January 2012.

[2] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell, “A survey of mobile phone sensing,” *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 140–150, September 2010.

[3] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips, "TRIBLER: a social-based peer-to-peer system," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 2, pp. 127–138, 2008.

[4] X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy, "Gossiping personalized queries," in *Proceedings of the 13th International Conference on Extending Database Technology (EBDT'10)*, 2010.

[5] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, 2012, to appear.

[6] I. Hegedűs, R. Ormándi, and M. Jelasity, "Gossip-based learning under drifting concepts in fully distributed networks," in *Proceedings of the 2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems*, ser. SASO '12, Lyon, France, 2012, to appear.

[7] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 226–235.

[8] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *J. Mach. Learn. Res.*, vol. 8, pp. 2755–2790, December 2007.

[9] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '01. New York, NY, USA: ACM, 2001, pp. 377–382.

[10] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldá, and R. Morales-Bueno, "Early drift detection method," *Fourth International Workshop on Knowledge Discovery from Data Streams*, vol. 6, pp. 77–86, 2006.

[11] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence, Proceedings of SBIA 2004*, ser. LNCS, vol. 3171. Springer, 2004, pp. 286–295.

[12] L. Minku, A. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 5, pp. 730 –742, may 2010.

[13] P. Luo, H. Xiong, K. Lü, and Z. Shi, "Distributed classification in peer-to-peer networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'07)*. New York, NY, USA: ACM, 2007, pp. 968–976.

[14] H. Ang, V. Gopalkrishnan, W. Ng, and S. Hoi, "Communication-efficient classification in P2P networks," in *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, ser. Lecture Notes in Computer Science, W. Buntine, M. Grobelnik, D. Mladenic, and J. Shawe-Ta ylor, Eds., vol. 5781. Springer, 2009, pp. 83–98.

[15] H. H. Ang, V. Gopalkrishnan, W. K. Ng, and S. Hoi, "On classifying drifting concepts in p2p networks," in *Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases: Part I*, ser. ECML PKDD'10. Berlin, Heidelberg: Springer, 2010, pp. 24–39.

[16] H. Ang, V. Gopalkrishnan, S. Hoi, and W. Ng, "Cascade RSVM in peer-to-peer networks," in *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, ser. Lecture Notes in Computer Science, W. Daelemans, B. Goethals, and K. Morik, Eds., vol. 5211. Springer, 2008, pp. 55–70.

[17] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta, "Distributed data mining in peer-to-peer networks," *IEEE Internet Computing*, vol. 10, no. 4, pp. 18–26, July 2006.

[18] S. Siersdorfer and S. Sizov, "Automatic document organization in a P2P environment," in *Advances in Information Retrieval*, ser. Lecture Notes in Computer Science, M. Lalmas, A. MacFarlane, S. Rüger, A. Tombros, T. Tsikrika, and A. Yavlinsky, Eds. Springer, 2006, vol. 3936, pp. 265–276.

[19] C. Hensel and H. Dutta, "GADGET SVM: a gossip-based sub-gradient svm solver," in *International Conference on Machine Learning (ICML), Numerical Mathematics in Machine Learning Workshop*, 2009.

[20] R. Ormándi, I. Hegedűs, and M. Jelasity, "Asynchronous peer-to-peer data mining with stochastic gradient descent," in *17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011)*, ser. Lecture Notes in Computer Science, vol. 6852. Springer, 2011, pp. 528–540.

[21] T. M. Mitchell, *Machine Learning*, 2nd ed., E. M. Munson, Ed. New York: McGraw-Hill, 1997. [Online]. Available: http://www.cs.cmu.edu/~tom/mlbook.html

[22] A. Montresor and M. Jelasity, "Peersim: A scalable P2P simulator," in *Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P 2009)*. Seattle, Washington, USA: IEEE, September 2009, pp. 99–100, extended abstract.

[23] "Golf api," http://github.com/RobertOrmandi/Gossip-Learning-Framework, 2011.

[24] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems*, vol. 25, no. 3, p. 8, August 2007.

[25] N. Tölgyesi and M. Jelasity, "Adaptive peer sampling with newscast," in *Euro-Par 2009*, ser. LNCS, vol. 5704. Springer, 2009, pp. 523–534.

[26] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '01. New York, NY, USA: ACM, 2001, pp. 97–106.

[27] A. Frank and A. Asuncion, "UCI machine learning repository," 2010.

[28] J. Vreeken, M. van Leeuwen, and A. Siebes, "Characterising the difference," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, pp. 765–774.

[29] A. Dries and U. Rückert, "Adaptive concept drift detection," *Stat. Anal. Data Min.*, vol. 2, no. 56, pp. 311–327, December 2009.