Gossip-based Learning under Drifting Concepts in Fully Distributed Networks

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

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

***Abstract*—In fully distributed networks data mining is an important tool for monitoring, control, and for offering personalized services to users. The underlying data model can change as a function of time according to periodic (daily, weakly) patterns, sudden changes, or long term transformations of the environment or the system itself. For a large space of the possible models for this dynamism—when the network is very large but only a few training samples can be obtained at all nodes locally—no efficient fully distributed solution is known. Here we present an approach, that is able to follow concept drift in very large scale and fully distributed networks. The algorithm does not collect data to a central location, instead it is based on online learners taking random walks in the network. To achieve adaptivity the diversity of the learners is controlled by managing the lifespans of the models. We demonstrate through a thorough experimental analysis, that in a well specified range of feasible models of concept drift, where there is little data available locally in a large network, our algorithm outperforms known methods from related work.**

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

I. Introduction

Fully distributed P2P systems and applications have seen a steadily increasing popularity in the past decade. We are currently experiencing a trend, in which these systems are transforming from pure technical platforms into social and feature-rich applications. This includes the recent emergence of smart phone platforms [1]–[3] as well as more traditional P2P applications that are being extended with features such as recommendation, spam filtering, and social networking [4]–[6]. The most important building blocks of these emerging systems are P2P data mining algorithms, that need to function in a large-scale, unreliable network, and that need to take the privacy of users into account as well.

In the systems mentioned above, it is essential that all algorithms work in an *adaptive* manner. These applications are running for a long time, without the possibility of tight central control. At the same time, the network and the environment changes continuously. The user base experiences continuous churn, and the user behavior and the world changes as well at all time scales, in other words, the system experiences *concept drift* at many levels [7]. The data mining algorithms have to follow these changes adaptively and provide up-to-date models at all times.

We are interested in scenarios, where data owned by a node cannot be moved outside the node due to privacy (or efficiency) concerns. In addition, we assume that data is distributed horizontally, but only a very small amount of data is available locally, perhaps only a single record. In addition, only a limited amount of new data can be sampled in order to follow concept drift. This scenario is typical in the systems mentioned above. For example, in a smart phone platform, if people are required to manually enter training data samples, then we cannot expect a huge amount of input at all nodes, especially if these samples correspond to rare events. Similarly, if the training data consists of user profiles, purchasing events, etc, then we cannot expect overwhelming data streams locally.

At the same time, even if local data is scarce, when considering the entire network, we are still provided with immense amounts of data considering that these networks can easily reach hundreds of millions of nodes. This calls for algorithms that can process fully distributed data efficiently and effectively, without collecting it to a central location.

In our previous work we have proposed the gossip learning framework (GoLF) to solve a part of this problem [8], [9]. The basic idea is that models take random walks, while being updated along the way using online algorithms, and while being combined as well. However, GoLF was presented as a one-shot algorithm, without taking adaptivity into account. Here we introduce adaptivity, hence we make GoLF a practical solution for realistic changing environments. The main idea is that we manage the distribution of the lifetime of the models in the network, making sure that we have a diverse set of models at all times.

Our contribution is twofold. First, we extend GoLF with components that allow it to deal with concept drift. With these changes, the algorithm can run indefinitely in a changing environment without any central control. Second, we perform a thorough experimental analysis. We compare the proposed algorithm with several baseline algorithms that idealize the main techniques for achieving adaptivity from related work. We show that in the domain where the sampling rate of the concept to be learned is low relative to the speed of drift, our solution is superior to all the baselines and its performance approximates the theoretical maximum. We also demonstrate the fault tolerance of the method.

II. System Model and Data Distribution

Our system model is a network of computers (peers). Each node in the network has a unique network address and can communicate to other nodes through messages if the address of the target node is locally available. We also assume that a *peer sampling service* is available, that provides addresses of random available peers at any time. Here we use NEWSCAST [10], [11] as our peer sampling service. Messages can be delayed or dropped, moreover, new nodes can join and leave the network without any warning. We assume that when a node rejoins the network it has the same state as at the time of going offline.

Regarding data distribution, we assume that the data records are distributed horizontally, that is, all the nodes store full records. Most importantly, in this paper we also assume that all the nodes store *exactly one record* (although the algorithms can be trivially adapted to a more general case). Having access to a single local record excludes the possibility of any local statistical processing of the data. Another important assumption is that the data never leaves the nodes, that is, the collection of the data at a central location is not possible due to privacy or infrastructural constraints.

III. Background

A. Machine Learning

The problem we tackle in this paper is *supervised classification* that can be defined as follows. We are given a training data set in the form of a set of training examples. Each training instance consists of a feature vector and a corresponding class label coming from an unknown underlying probability distribution $\mathcal{D}$. Let us denote this training database with $S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \mathbb{R}^d \times C$ where d is the *dimension* of the problem and C is the set of class labels. The main goal when solving a classification problem is to find a function $f : \mathbb{R}^d \rightarrow C$ using the observations from S that can classify *any samples* including those that are not in the training set but also generated by the probability distribution $\mathcal{D}$. The above mentioned property is known as *generalization*. Function f is called the *model* of the data. When the training samples are available as a stream then the training process is known as *online learning*.

B. Concept Drift

The distribution $\mathcal{D}$ mentioned above may change over time. For this reason we parameterize the distribution of the samples by the time t, that is, at time t a sample $(x, y) \in \mathbb{R}^d \times C$ comes from $\mathcal{D}_t$. This means that any learned prediction function f might become outdated if new samples are not used to update or replace it. The challenge is to design an *adaptive* algorithm that provides a good model f_t at any given time t.

C. Gossip Learning

The adaptive algorithm we propose here is based on our Gossip Learning Framework (GoLF) [8], [9]. In Algorithm 1 a variant of GoLF is shown, extended with two lines (marked with comments) that represent the modifications related to concept drift handling. Here we focus on the original parts; concept drift handling is explained in Section V.

The basic idea is that multiple models perform random walks over the network in parallel, while applying an online learning algorithm to improve themselves, and getting combined via ensemble learning methods.

At each node in the network the same algorithm is run. The algorithm consists of an active loop of periodic activity, and the message handler method ONRECEIVEMODEL to process incoming models. This method updates the incoming model using the local training example and stores it in its *cache* called *receivedModels*. In the active loop the stored models are sent to random neighbors and are removed from the cache. At anytime the freshest model (that is, CURRENTMODEL, the model added to the cache most recently) is used for prediction.

Incoming models can be combined as well, both locally (e.g., merging the received models, or implementing a local voting mechanism) or globally (e.g., finding the best model in the network) [8], [9]. In this paper we do not discuss the possibilities of local model combination, but in Section VI we study the identification and the performance of the best model in the network.

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 length of the period of the loop Δ is the same at all nodes. There is no explicit failure detection, 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.

The algorithm contains abstract methods that can be implemented in different ways to obtain a concrete learning algorithm. The main placeholders are INITMODEL and UPDATEMODEL. The method SELECTPEER is the interface for the peer sampling service, as described in Section II. Here we use the NEWSCAST algorithm [10], [11], 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.

IV. Related Work

We discuss the state-of-the-art of the techniques for dealing with concept drift in general, as well as the known P2P approaches.

A good overview of concept drift can be found in [7]. Many algorithms apply chunk based learning techniques [12]–[14], that is, they teach a new classifier when a new set of samples (a chunk) becomes available via the stream of samples. This approach could be suitable when the stream of samples produces a large number of samples relative to the speed of concept drift, that is, when the method can collect enough samples quickly enough to build an up-to-date classifier. Moreover, determining the chunk size is not easy, yet this parameter is crucial for the prediction performance.

One improvement of chunk based techniques is to detect the drift, that is, to use some performance related measures to decide when to trigger the drift handling method [13], [15], [16]. The early approaches use a single model which is discarded when drift is detected and a new one is constructed immediately. Recently, ensemble learning has also been proposed [17]. In this case, when the drift is detected, they teach a new model but this new model is added to an ensemble pool that is used to perform prediction, possibly involving a weighting mechanism as well.

Learning in P2P systems is a growing area, some examples include [8], [18]–[24]. Very few works address issues related to concept drift in a P2P network. A fully distributed decision tree induction method was proposed by Ghaduri et al. [25]. The proposal involves drift detection, that triggers a tree update.

Another solution was proposed by Ang et al. [20]. This method implements the so-called RePCoDE framework which detects the drift (reactive behavior) and simultaneously predicts it as well (proactive behavior). The basic learning mechanism is performed by chunk-based learning, but the models taught on previous data chunks are also kept and used during prediction (ensemble based aspect). As the extensive evaluations show the proposed approach works well in various scenarios, although its communication cost is rather high, since it involves network wide model propagation. A number of heuristics are proposed to reduce this cost.

As we show later, our approach is rather different from these, as we do not attempt to detect concept drift at all. Instead, we take advantage of the large size of the network and maintain a diverse pool of models via managing the age distribution in the pool. In addition, these models take random walks, so they are all able to take advantage of the new samples. This is especially important, if new samples arrive only very rarely at any given node, but in the overall network there are enough samples to process.

V. Algorithm

A. Handling Concept Drift

In GoLF many models perform random walks converging to a good classifier. Without drift handling capabilities these models will not be able to adapt when concept drift occurs, that is, when the underlying probability distribution of the data changes. We will demonstrate this phenomena through empirical evaluations in Section VI. There, it is shown that when a model becomes too old, it is not able to adapt anymore to the changing environment. This is a well-know property of most online learning algorithms.

It would therefore be useful to build new models when the underlying distribution changes. Indeed, this is what concept drift techniques attempt to do, as we discussed previously. However, instead of detecting drift, in our approach we achieve adaptivity by controlling the *lifetime distribution* of the models available in the network, which introduces network-wise model diversity in terms of model age.

Algorithm 1 Gossip Learning Framework

```
1:  $c \leftarrow 0$
2:  $m \leftarrow$ initModel()
3:  currentModel $\leftarrow$ initDriftHandler($m$)          ▷ DHC init.
4:  receivedModels.add(currentModel)
5:  loop
6:      if receivedModels = $\emptyset$ then
7:          $c \leftarrow c + 1$
8:      if $c = 10$ then
9:          receivedModels.add(currentModel)
10:     for all $m \in$ receivedModels do
11:         $p \leftarrow$ selectPeer()
12:         send $m$ to $p$
13:         receivedModels.remove($m$)
14:         $c \leftarrow 0$
15:     wait($\Delta$)

16: procedure onReceiveModel($m$)
17:     $m \leftarrow$ driftHandler($m$)          ▷ DHC impl.
18:     currentModel $\leftarrow$ updateModel($m$)
19:     receivedModels.add(currentModel)
```

Algorithm 2 Drift Handling Component

```
1: procedure initDriftHandler($m$)
2:     $m.TTL \leftarrow$ generateTTL()
3:     return $m$

4: procedure driftHandler($m$)
5:     if $m.age = m.TTL$ then
6:         $m \leftarrow$ initModel()
7:         $m \leftarrow$ initDriftHandler($m$)
8:     return $m$
```

In Algorithm 1 we show the skeleton of the GoLF algorithm extended with a drift handling component (DHC). Drift handling is implemented through two abstract functions called in line 3 and line 17. These function calls add the capability of drift handling to GoLF *independently* of the model and learning algorithm applied in GoLF. From now on, we will call the GoLF framework extended with drift handling AdaGoLF.

The implementations of the drift handling components are shown in Algorithm 2. We add a new time-to-live (TTL) field to each model. When a new model is created this field is initialized (at line 2 of Algorithm 2) to a value generated from a predefined probability distribution that we call the Model Lifetime Distribution (MLD). The selection of this probability distribution is crucial as we explain later in this section. The age of the model increases with each hop. When the model age reaches the TTL value the model "dies" and a new one is created (at line 6 of Algorithm 2) with a newly generated, independent TTL value (at line 7 of Algorithm 2).

We would like to characterize the distribution of the age of the models in the network at some time point t. Consider

Algorithm 3 Learner Component

```
1:  procedure INITMODEL
2:      $m.age \leftarrow 0$
3:      $m.w \leftarrow \bar{0}$
4:      return $m$

5:  procedure UPDATEMODEL($m$)                    ▷ $\lambda = 0.0001$
6:      $m.age \leftarrow m.age + 1$
7:      $m.\eta \leftarrow 1/m.age$
8:      $\hat{y} \leftarrow m.$predict($x$)        ▷ $(x, y)$ is the local example
9:      $m.w \leftarrow (1 - m.\eta \cdot \lambda)m.w + m.\eta \cdot (y - \hat{y})x$
10:     return $m$

11: procedure PREDICT($x$)
12:     $p_0 \leftarrow 1/(1 + exp($currentModel$.w^T x))$
13:     $p_1 \leftarrow 1 - p_0$
14:     return $p_0 > p_1$ ? 0 : 1
```

a sequence of models $m_1, m_2, \ldots$ according to a random walk where m_i is removed and m_{i+1} is started by method DRIFTHANDLER. The *lifetime sequence* of these models $m_1.TTL$, $m_2.TTL$, … forms a *renewal process*. Let the age of the model that is "alive" at time t be the random variable A_t; we are interested in the distribution of A_t. More formally, let S_t be the birth time of the model alive at time t:

$$S_t = \max_k \{V_k : V_k = \sum_{i=1}^{k} m_i.TTL \text{ and } V_k < t\}, \qquad (1)$$

in which case $A_t = t - S_t$. Applying results from renewal theory [26] (the renewal equation and the expectation equation) we get the expected model age and its variance as the time tends to infinity:

$$\begin{aligned} \mathbb{E}(A_t) &\xrightarrow{t\to\infty} \frac{\mathbb{E}(X^2)}{2\mathbb{E}(X)} \\ \mathbb{D}^2(A_t) &\xrightarrow{t\to\infty} \frac{\mathbb{E}(X^3)}{3\mathbb{E}(X)} - \left(\frac{\mathbb{E}(X^2)}{2\mathbb{E}(X)}\right)^2 \end{aligned} \qquad (2)$$

We selected the lognormal distribution as our MLD with parameters $\mu = 8$ and $\sigma^2 = 0.5$, that gives us an expected age of $\mathbb{E}(A_t) \approx 3155$, and $\mathbb{D}(A_t) \approx 3454$. The method is very insensitive to the distribution, as long as it provides a diverse-enough set of values with young as well as old models present at the same time.

B. The Learner Component

In our evaluations we applied the logistic regression [27] learner which is a widely used online classification algorithm. Logistic regression looks for a set of parameters (w) that maximizes the logarithm of the conditional data likelihood

$$l(w) = \sum_{i=1}^{n} \ln P(y_i|x_i, w) - \frac{\lambda}{2}\|w\|^2. \qquad (3)$$

Here (x_i, y_i) represents the ith sample from the training database and λ is the regularization parameter. The implementation of the online learning rule for the above mentioned objective function can be seen in Algorithm 3 (in line 9, where η is the learning rate). The prediction of a model on a sample x is performed locally by selecting the most likely class (the implementation for the two-class case can be found in Algorithm 3 starting from line 11).

C. Spreading the Approximated Best Model

Since we have numerous diverse models available in the network, it is a natural idea to try to continuously search for the current best model and share this model among all the nodes. It is non-trivial to find the model with the best prediction performance since we have no independent test data available. One solution is to define an approximate error score that is computable in the learning phase. A possible approach involves using each training example as a test sample before using it for updating the model. We can then compute a moving average of these elementary validation scores to get an aggregated error score of the model. The model with the minimum score can be spread in the network via gossip-based minimum search [28]. We refer to this slightly modified version ADAGOLF as MINADAGOLF.

D. Communication Complexity

The expected communication cost for a single node in a period of Δ time (one cycle) is at most two. To see this, consider, that there are M models in the entire network, and there are $N > M$ nodes (with $N = M$ if there is no message drop and no churn), and that every model performs a random walk with exactly one hop in each cycle. Since we assume a good quality peer sampling service, the number of incoming messages will follow a Poisson distribution with $\lambda = 1$ if $M = N$, and less if $M < N$. Since each incoming message also generates an outgoing message, the overall number of messages will be twice the number of the incoming ones.

MINADAGOLF involves a minimum search that takes $O(\log N)$ messages per node itself. If we assume that the gossip-based minimum spreading messages piggyback the ADAGOLF gossip messages, then no extra messages are involved, however, the message size will double (it will contain two models instead of one) and we learn the minimum only with a logarithmic delay. The space complexity of a model strongly depends on the selected learning algorithm, as well as the dimensionality of the data.

VI. EXPERIMENTAL SETUP

A. Drift Dynamics and Drift Types

Our algorithm is not the optimal choice in all possible concept drift scenarios, however, in certain important cases we will show it to be the most favorable option. To be able to characterize the different drift scenarios. Let us first identify a few key features of drift.

As of dynamics, there are two important properties that characterize an environment involving concept drift: the *speed of drift*, and the *sampling rate*. The speed of drift defines how much the underlying concept changes within a

Table I
THE MAIN PROPERTIES OF THE BASELINE AND THE ADAPTIVE ALGORITHMS.

Name	Computational complexity of learning / cycle	models used for prediction	Communication complexity / cycle
LOCALLEARNER	O(chunkSize)	1	0
CACHEBASEDLEARNER	O(sampleCacheSize)	1	0
VOTELEARNER	O(chunkSize)	N	$O(N)$
CACHEDVOTELEARNER	O(sampleCacheSize)	N	$O(N)$
VOTELEARNERWITHDELAY	O(chunkSize)	modelCacheSize	$O(1)$
CACHEDVOTELEARNERWITHDELAY	O(sampleCacheSize)	modelCacheSize	$O(1)$
GLOBAL	O(chunkSize*N)	1	$O(N)$
ADAGOLF	$O(1)$	1	$O(1)$
MINADAGOLF	$O(1)$	1	$O(\log N)$

unit time interval. The sampling rate defines how many new samples become available within a unit time interval.

A third speed-related parameter is the cycle length Δ of ADAGOLF. However, these three parameters can be considered redundant, since in the range of reasonable cycle lengths Δ (where Δ is significantly greater than message transmission time) we can always chose a Δ that keeps drift speed (or sampling rate) constant. For this reason, we will chose Δ as the unit of time, and we will define drift speed and sampling rate in terms of Δ, leaving us with two remaining free parameters.

However, in this paper we do not investigate the speed of drift independently, since the interesting scenarios are differentiated more by the ratio of drift speed and sampling rate. If drift is too fast relative to the sampling rate, then there is no chance to learn a reasonable model with any method. If drift is too slow relative to the sampling rate, then the problem is not very challenging, since even the simplest baselines can achieve a very good performance [7].

The type of the drift is another important property. In our scenarios drift can be *incremental* or *sudden* [7]. The actual drift types are described later in this section.

B. Baseline Algorithms

We selected our baseline algorithms so as to represent the most typical approaches from related work with a simpler, but optimistic version, that is guaranteed to perform better by construction than the corresponding published algorithm.

Our simplest baseline is LOCALLEARNER where each node simply collects its local samples during a cycle and builds a model based on this sample set at the end of each cycle. If a node does not observe any training samples during a cycle then the previous model is used for prediction. This solution involves no communication, but with low sample rates it performs poorly.

CACHEBASEDLEARNER is a more sophisticated baseline which uses a limited size memory, a FIFO queue (called cache) in which it collects samples. For maximal fairness, the memory size was optimized during preliminary experiments, and was set to 100. Note that our test datasets are learnable from 100 samples very well. Due to the optimized cache size, this baseline represents the the chunk based as well as trigger based approaches mentioned in Section IV. The cache size can be considered an optimized chunk size which is the optimal solution of the trigger based approaches as well assuming continuous drift. Thus the result of this baseline can be considered as an upper bound of the performance of the local chunk and trigger based approaches. This baseline still uses no communication, but it performs better than LOCALLEARNER.

The baselines VOTELEARNER and CACHEDVOTELEARNER are natural extensions of the previous baselines: they use collaboration within the network. All the nodes send their models they create at the end of the cycle to every other node, and the prediction is performed by voting. These are powerful baselines in terms of prediction performance, although their communication costs are extremely large due to a full broadcast of all the models in each cycle. It is easy to see that these voting-based baselines represent the ensemble based approaches mentioned in Section IV, hence the performance of these idealized variants can be considered an upper bound of the result of the ensemble based approaches.

VOTELEARNERWITHDELAY and CACHEDVOTELEARNERWITHDELAY are the communication-effective versions of VOTELEARNER and CACHEDVOTELEARNER, respectively. They work exactly the same way like their ancestors but the model spreading process is slowed down: in each cycle each node sends its model to exactly one neighbor. These models are collected in a FIFO queue of a fixed size (the model cache) and prediction is based on the voting of the models in the model cache. We set the model cache size to be 100, similarly to the sample cache described above.

The last baseline called GLOBAL is an algorithm that can observe all the samples observed in the network in a cycle and can build a model based on them. The implementation of this algorithm is infeasible and its result can be considered as a theoretical upper bound on the performance of the distributed baselines and ADAGOLF.

The complexity of these algorithms is summarized in Table I. The network size is denoted by N, CHUNKSIZE is the number of samples observed by a node in a cycle, which depends on the sampling rate. The last two lines show the same properties for our algorithms. In the case of MINADAGOLF we assumed that the minimum is searched for in each cycle. Note that if we allow for $O(\log N)$ cycles of delay for learning the minimum, which is a very reasonable choice, then a constant number of messages would suffice.

C. Data Sets

In our evaluations we used synthetically generated as well as real world data sets. In both cases we modeled drift by changing the labeling of the data set. That is, drift itself was synthetic even in the case of the real world database. The nodes can get random samples according to the given sampling rate from a training pool.

The synthetic database was generated by drawing uniform random points from the d dimensional uniform cube. The labeling of these points is defined by a hyperplane that naturally divides the examples into a positive and negative subset. Drift is modeled by moves the hyperplane periodically over time [29]. A hyperplane at time t is defined by its normal

$$w_t = (1 - \alpha_t)w_s + \alpha_t w_d, \tag{4}$$

where $w_t, w_s, w_d \in \mathbb{R}^d$, $0 \le \alpha_t \le 1$, and w_s and w_d are the source and the destination hyperplanes, respectively, which are orthogonal to each other. The normal w_s is generated at random with all coordinates drawn from $[0, 1]$ uniformly, and $w_d = (1, \ldots, 1)^T - w_s$.

In the case of incremental drift the hyperplane is moved smoothly back and forth between w_s and w_d. This can formally be defined as

$$\alpha_t = \begin{cases} 1 - (tv - \lfloor tv \rfloor) & \text{if } \lfloor tv \rfloor \mod 2 = 1 \\ tv - \lfloor tv \rfloor & \text{otherwise} \end{cases} \tag{5}$$

where v is the speed of drift. When sudden drift was modeled, we used the same dynamics for α_t but rounded it to implement discontinuity: $\alpha'_t = \text{round}(\alpha_t)$. This results in switching back and forth between w_s and w_d with a period of $2/v$ time units.

We used one real world data set, namely the Image Segmentation [30] data set from the UCI Machine Learning Repository. The database has 19 real valued features and 7 class labels. The class label ratio is balanced. Originally this data set does not support the evaluation of concept drift. We implemented a mechanism proposed by [31], [32] to add synthetic drift to this data set. This simple mechanism consists of rotating the class labels periodically. This results in a variant of sudden drift.

D. Evaluation Metrics

For performing evaluations we split all the databases into a training and an evaluation set. The proportion of this splitting was 80/20% (training/evaluation) except in the case of the real data set where the training/evaluation split was provided by the owners of the database. In all cases the splitting was performed before the simulations since all the evaluation sets are obviously independent from the training sets.

Our main evaluation metric is prediction error. In the case of AdaGoLF, 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 also called the 0-1 error.

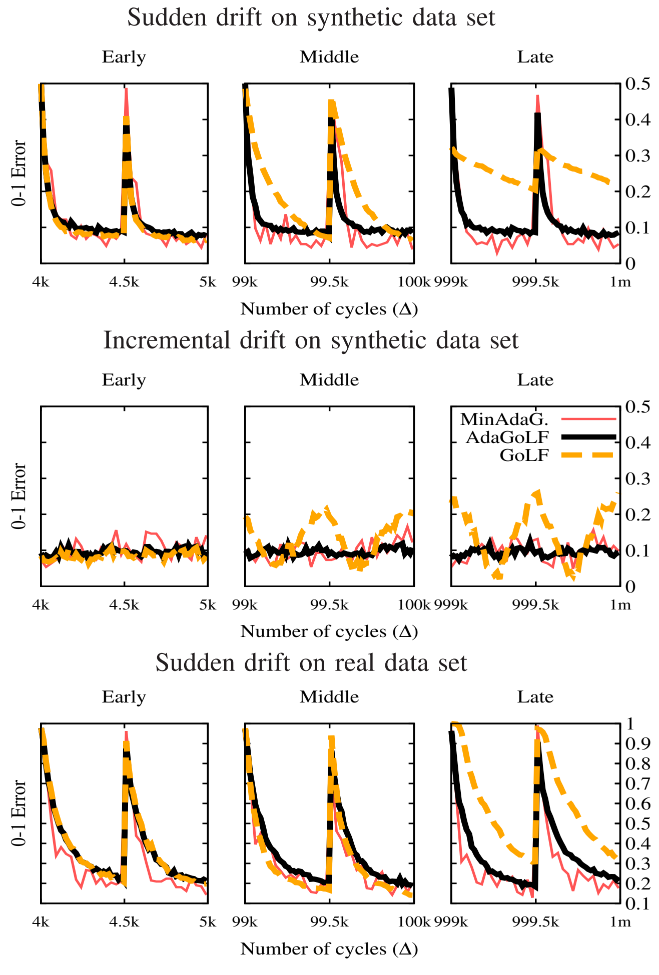

Figure 1. The burn in effect as a motivation for adaptivity.

E. Simulation scenarios

The experiments were performed using the event-based engine of PeerSim [33]. Apart from experiments involving no failure, we model the effect of message drop, message delay, and churn. In all the failure scenarios we modeled realistic churn, that is, the nodes were allowed to join and leave the network. The length of the online sessions was estimated by applying maximum likelihood estimation considering lognormal distribution based on a private BitTorrent trace called the FileList.org collected by Delft University of Technology [34]. We set the offline session lengths so that at any moment in time 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.

In the scenario we call *AF mild* (all failures mild), message drop probability was set to be 0.2, and message delay is modeled as a uniform random delay from the interval $[\Delta, 2\Delta]$ where Δ is the cycle length. Moreover we applied a very heavy failure scenario as well where the message drop probability was 0.5 and message delay was uniform random from $[\Delta, 10\Delta]$. We refer to this scenario as *AF hard*.

The default value for the sampling rate parameter was $1/\Delta$, and the default for network size is $N = 100$. Both of these values are explored in our experimental study.

VII. Experimental Results

A. Adaptivity

Here we briefly illustrate the problem that is caused by the lack of adaptivity. Online learners exhibit a burn in effect when run for a long time, as demonstrated by Figure 1, where we present the prediction error, averaged over the network, of the original GoLF (baseline), AdaGoLF, and MinAdaGoLF as a function of time. We can see that AdaGoLF shows no burn in effect. The plots suggest that MinAdaGoLF might have a slightly better performance than AdaGoLF, but with a higher variance.

B. The Effect of Sampling Rate

We performed evaluations using all our database and drift-type configurations (synthetic-sudden, synthetic-incremental and real world-sudden). For each of these configurations we study the effect of different sampling rates. The results are shown in Figures 2, 3 and 4. In each result set we selected four distinct sampling rates: 0.01, 0.1, 1, 10 samples per cycle.

When the sampling rate is 0.01, all the nodes receive only a single new sample on average in every 100 iterations. While the baseline methods build models only based on local samples, AdaGoLF can take advantage of many more samples due to the models performing a random walk. This allows AdaGoLF to approximate the performance of Global that has the best possible performance by construction. Increasing the sampling rate results in a gradually decreasing difference between the baseline algorithms and AdaGoLF. In fact, with high sampling rates, AdaGoLF is outperformed by most of the baselines in the case of the sudden change scenarios.

We need to note here, that in the present version of the algorithm all models use exactly one sample for the update in each hop, even if more samples are available. This means that there are lots of possibilities to enhance AdaGoLF to deal with high sample rates better. Nevertheless, AdaGoLF is clearly the best option if the sampling rate is low.

In the incremental drift scenario, we should mention the remarkable stability of AdaGoLF under each sampling rate. Here, AdaGoLF remains competitive even in the highest sampling rate scenario. This is rather interesting, given that AdaGoLF ignores most of the samples in that case, as mentioned before.

C. Fault Tolerance

We performed simulations with the failure scenarios described previously. Figure 5 contains the results. From this we can observe that the effect of the failures is a slower convergence speed. This effect can mostly be accounted for by the message delay, since all the random walks will be proportionally slower. This has the same effect as if the cycle length Δ was proportionally larger in a failure-free scenario.

D. Scalability

In Figure 6 we present the results of AdaGoLF in different network sizes. We cannot identify any significant effect of the network size in most of the scenarios. In the case of the real dataset (that is harder to learn) we can realize that larger networks result in a slightly better performance, which is most likely due to the fact that more independent samples are available in larger networks.

VIII. Conclusion

In this paper we proposed an adaptive version of the GoLF framework, called AdaGoLF. Adaptivity is implemented in a very simple manner through the management of the age distribution of the models in the network, making sure that there is a sufficient diversity of different ages in the pool. This is not a usual approach, as most of the related work focuses on building and combining local models, and on detecting and predicting drift explicitly.

We performed a thorough experimental study in which we compared AdaGoLF with a set of baseline algorithms that represented idealized versions of the main techniques that are applied in related work. Our main conclusion is that in those scenarios, where the sampling rate from the underlying distribution is low relative to the speed of drift, AdaGoLF clearly outperforms all the baseline solutions, approximating the "God's Eye view" model, that represents the best possible performance. AdaGoLF was designed to deal with exactly this scenario. We also indicated, that AdaGoLF can be enhanced to deal with (or rather, be robust to) higher sample rates as well, although in that case purely local model building can also be sufficient.

Acknowledgment

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). The authors would like to thank Gábor Szűcs for the consultations about renewal processes.

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] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, and J. Reich, "Mobiscopes for human spaces," *Pervasive Computing, IEEE*, vol. 6, no. 2, pp. 20–29, april-june 2007.

[3] 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.

[4] 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.

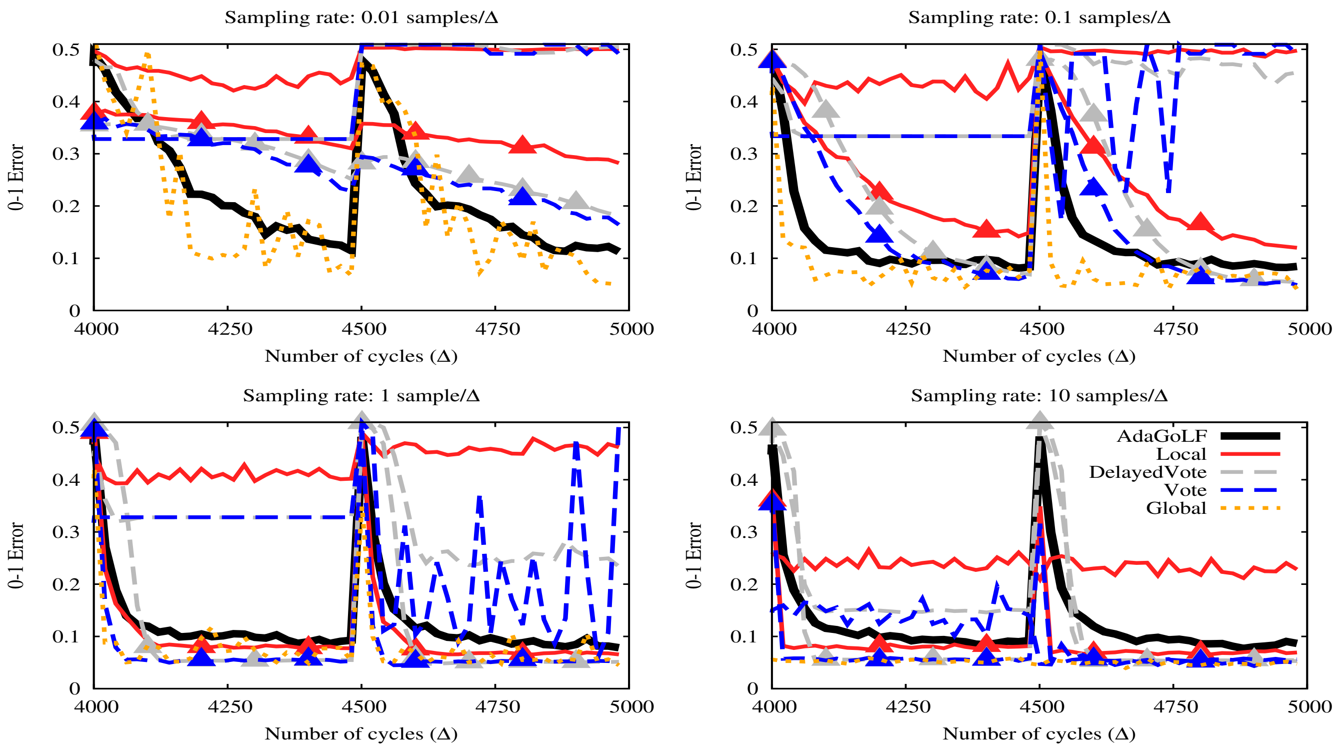

Figure 2. The effect of sampling rate under sudden drift (the lines marked with △ belong to cache based baselines).

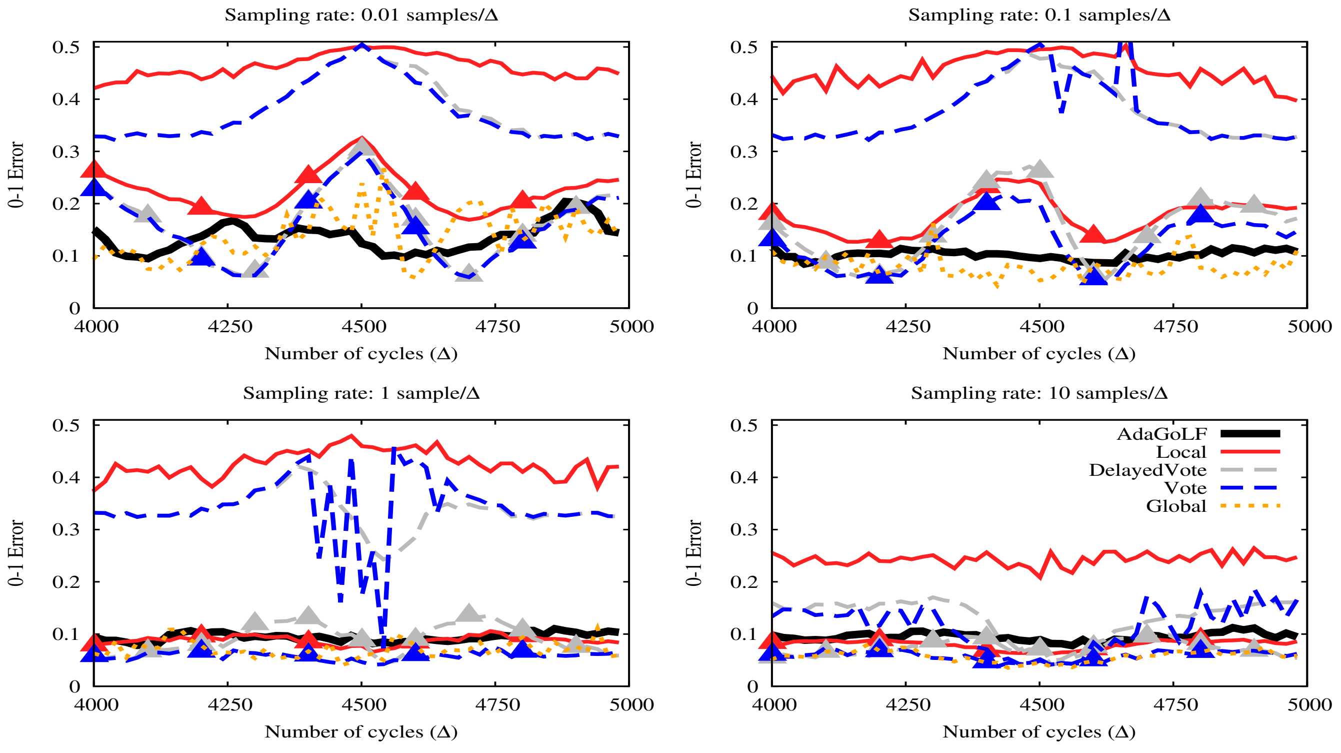

Figure 3. The effect of sampling rate under incremental drift (the lines marked with △ belong to cache based baselines).

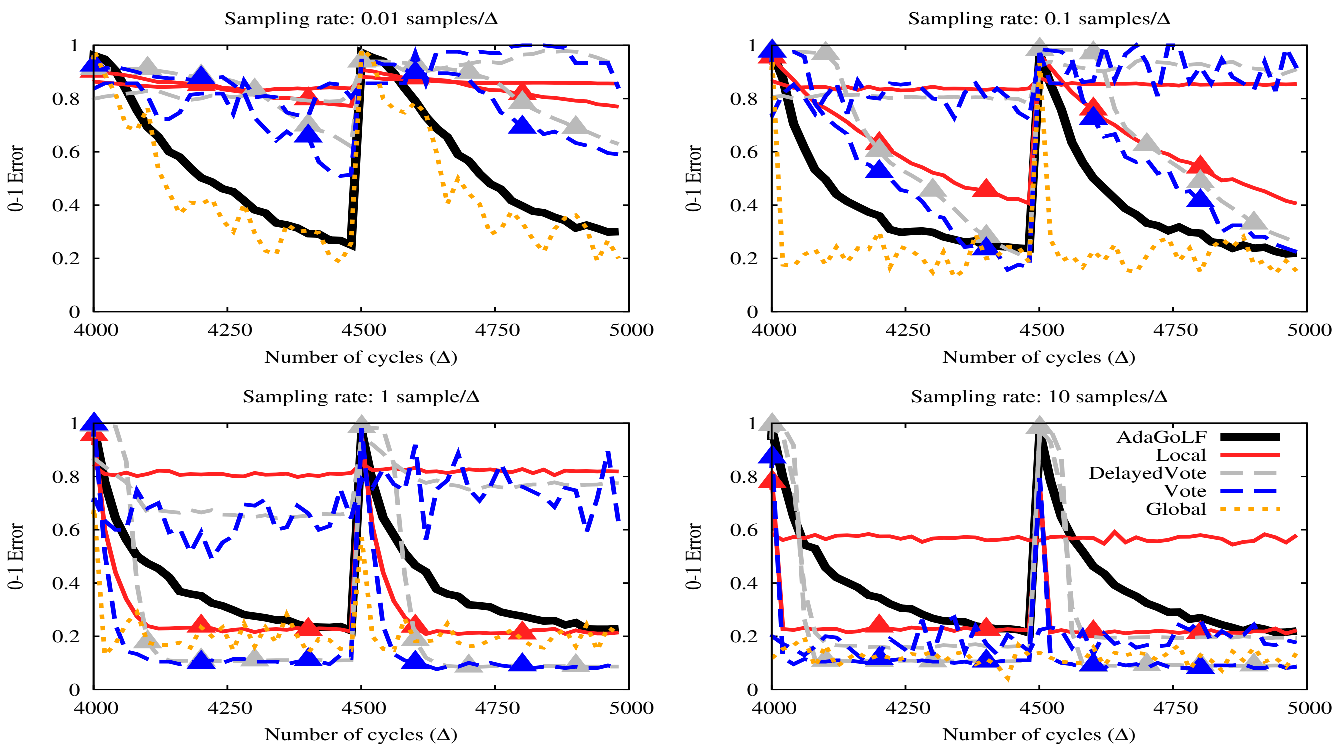

Figure 4. The effect of sampling rate over the real database (the lines marked with △ belong to cache based baselines).

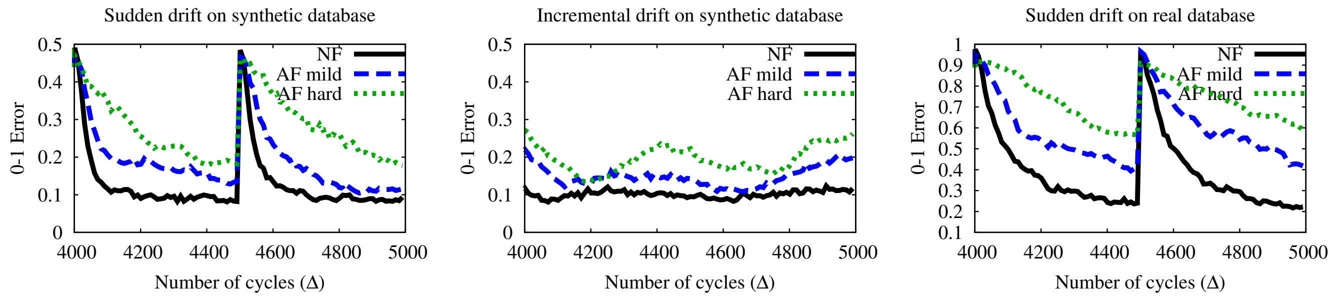

Figure 5. Prediction performance under failure.

[5] 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.

[6] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "PeerSoN: P2P social networking: early experiences and insights," in *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems (SNS'09)*. New York, NY, USA: ACM, 2009, pp. 46–52.

[7] I. Zliobaite, "Learning under concept drift: an overview," arxiv.org, Tech. Rep. 1010.4784, 2010. [Online]. Available: http://arxiv.org/abs/1010.4784

[8] 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.

[9] 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.

[10] 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.

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

[12] 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*

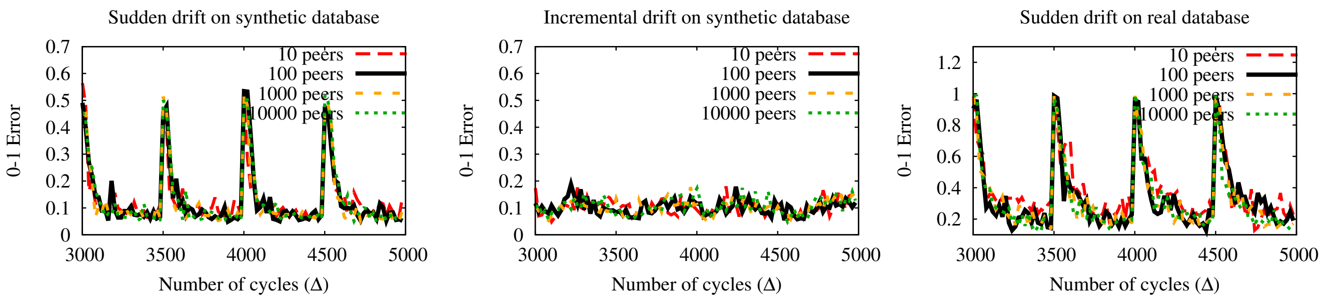

Figure 6. The effect of network size (with sampling rate $1/\Delta$).

Knowledge discovery and data mining, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 226–235.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta, "Distributed decision-tree induction in peer-to-peer systems," *Stat. Anal. Data Min.*, vol. 1, pp. 85–103, June 2008.

[26] G. Grimmett and D. Stirzaker, *Probability and Random Processes*, ser. Texts from Oxford University Press. Oxford University Press, 2001.

[27] 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

[28] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219–252, August 2005.

[29] 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.

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

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

[32] 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.

[33] 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.

[34] "Filelist," http://www.filelist.org, 2005.