Massively Distributed Concept Drift Handling in Large Networks*

István Hegedűs, Róbert Ormándi
University of Szeged
Szeged, H-6720, Hungary
{ihegedus, ormandi}@inf.u-szeged.hu

Márk Jelasity
Univ. Szeged and Hungarian Acad. Sci.
Szeged, H-6720, Hungary
jelasity@inf.u-szeged.hu

Abstract

Massively distributed data mining in large networks such as smart device platforms and peer-to-peer systems is a rapidly developing research area. One important problem here is concept drift, where global data patterns (movement, preferences, activities, etc.) change according to the actual set of participating users, the weather, the time of day, or as a result of events such as accidents or even natural catastrophes. In an important case—when the network is very large but only a few training samples can be obtained at each node locally—no efficient distributed solution is known that could follow concept drift efficiently. This case is characteristic of smart device platforms where each device stores only one local observation or data record related to a learning problem. Here we present two algorithms to handle concept drift. None of the algorithms collects data to a central location, instead models of the data perform random walks in the network, while being improved using an online learning algorithm. The first algorithm achieves adaptivity by maintaining young as well as old models in the network according to a fixed age distribution. The second one measures the performance of models locally, and discards them if they are judged outdated. We demonstrate through a thorough experimental analysis that our algorithms outperform the known competing methods if the number of independent local samples is limited relative to the speed of drift: a typical scenario in our targeted application domains. The two algorithms have different strengths: while the age distribution approach is very simple and efficient, explicit drift detection can be useful in monitoring applications to trigger control action.

1 Introduction

Very large fully distributed systems are now transforming from pure technical platforms into social and feature-rich applications. Examples include the recent emergence of smart phone platforms [2, 27, 36] as well as more traditional P2P applications that are being extended with features such as recommendation, spam filtering, and social networking [8, 11, 37]. An important building block of these emerging systems is data mining [18].

These systems pose a number of challenges for data mining. In this paper, we focus on the problem of *concept drift* [46] that occurs when the data patterns we wish to learn in the network change either continuously or suddenly. This can happen due to different reasons. For example, the set of users of an application can change, or external factors (such as the weather) can vary. This has an effect on how people react to, for example, traffic situations, it can influence what kind of movies they want to watch, it can affect their mobility patterns, and so on. People can also behave differently during their normal daily routine, or during a demonstration, for example. Such external factors cannot always be explicitly captured, and thus

*Electronic version of an article published as Advs. Complex Syst. 16, 1350021 (2013), DOI: 10.1142/S0219525913500215. ©World Scientific Publishing Company, `http://www.worldscientific.com/worldscinet/acs`. M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences. This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013) and 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. This work is based on the revised versions of two previous conference publications [20, 22].

they can be considered a source of uncertainty and dynamism. The applied data mining algorithms have to follow these changes adaptively to 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. Currently the norm is that data is uploaded to data centers, where it is processed [9, 14, 29]. However, this practice raises serious privacy issues [12], and on a very large scale it can also become very expensive, which seriously reduces the set of potential applications.

In addition, we assume that data is distributed horizontally, that is, full data records (e.g. profiles, personal histories of sensor readings, etc) are stored at all nodes, but very few, perhaps only a single record is available at any given node. In addition, only a limited amount of new data can be sampled in order to follow concept drift. These assumptions are natural in the systems mentioned above, where, for example, a mobile device stores information only about its owner like user profiles, purchasing events, or mobility patterns. Besides, in applications, where people are even required to manually enter training data samples we cannot expect a huge amount of input at all nodes, especially if these samples correspond to rare events.

In our previous work we have proposed the gossip learning framework (GoLF) for massively distributed data mining targeted to those environments that are characterized above [34, 35], where models of the data perform random walks in the network, while being improved using an online learning algorithm (see Section 4 for a description). However, one major limitation of GoLF was that it supported only one-shot algorithms, without taking adaptivity into account. This is a problem, because—as we argued above—in a realistic environment we are typically not able to take all relevant features into account when we learn data patterns, thus the validity of a pattern can change continuously or suddenly, in which case the learned data models need to be updated or refreshed. This can be achieved through a manual restart, but on a large scale, an automated method is necessary that allows the system to adapt to changes without human intervention. This would allow the system to scale both in terms of the size of the network, as well as the number of learning tasks supported simultaneously.

We propose two orthogonal approaches to follow concept drift in GoLF. The idea behind our first approach is that we manage the distribution of the lifetime of the models in the network, making sure that we have both young (and thus adaptive) models and old models at all times. This approach provides a very simple and efficient adaptive mechanism which makes it possible to seamlessly follow the changing data patterns. However, if change in the data patterns has to be detected explicitly, this solution is not suitable. For this reason, as our second contribution, we propose a mechanism that estimates and monitors the changes in the performance of the current data models and discards those ones that show a deteriorating performance.

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 algorithms with several baseline algorithms that idealize the main techniques for achieving adaptivity from related work. We show that in the domain where the number of independent samples available locally is low relative to the speed of drift, our solutions are superior to all the baselines and their performance approximates the theoretical maximum. We also demonstrate the fault tolerance of the method.

2 System Model and Data Distribution

As our system model, we consider a network of nodes (peers)—where the number of nodes potentially can be very large—that are typically personal computing devices such as PCs or mobile devices. 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 an additional middleware (*peer sampling service*) is also available, that can provide addresses of probably available peers at any time in the network selected uniformly at random. Many implementations of the peer sampling service are known, in this paper we use NEWSCAST [25,41]. The messages in the network can be delayed or dropped, moreover, nodes can leave and join the network again without prior notice. We assume that when re-joining the network, a node has the same state as at the time of going offline.

Regarding data distribution, we assume that the records of the database are distributed horizontally, that is, all the nodes store full records. Additionally, in this paper we assume that all the nodes store *exactly* *one record* (although the algorithms can be trivially adapted to—and in fact benefit from—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.

3 Related Work

We discuss the state-of-the-art related to concept drift in general, as well as in the area of P2P learning.

3.1 Non-Distributed Concept Drift Handling

A good overview of concept drift can be found in [46]. Many algorithms apply chunk based learning techniques [26, 40, 43], 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 drift, that is, to use some performance related measures to decide when to trigger the drift handling method [7, 17, 26]. 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 [31]. In this case, when drift is detected, a new model is created but this new model is added to an ensemble pool that also contains older models. This pool is used to perform prediction, possibly involving a weighting mechanism as well.

Due to our system model, we are not able to collect chunks since there is not enough local data available. However, as we explain later, we will use a form of drift detection in which nodes cooperate with each other to detect drift.

3.2 Handling Concept Drift in Fully Distributed Environments

Learning in P2P systems is a growing area, some examples include [3–5, 13, 23, 30, 34, 39]. Very few works address issues related to concept drift in a P2P network. A fully distributed decision tree induction method was proposed by Bhaduri et al. [10]. The proposal involves drift detection, that triggers a tree update. The proposed solution is a distributed adaptive threshold detection algorithm that—although elegant—is a special purpose approach that does not generalize to arbitrary learning algorithms like our approaches do.

Another solution was proposed by Ang et al. [5]. This method implements the so-called RePCoDE framework which detects 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. In this paper we assume that chunk based approaches are not viable due to the lack of sufficient amounts of local data.

Our main contribution w.r.t. related work is to propose two *generic* methods to efficiently deal with the scenario, when samples arrive only very rarely at any given node, but in the overall network there are enough samples to learn high quality models.

4 Background

Here we briefly introduce the main concepts that we build our work on, both the most basic notions from machine learning as well as our own previous work.

Algorithm 1 Skeleton of the original GoLF learning protocol

```
1: initModel()
2: loop
3:     wait($\Delta$)
4:     $p \leftarrow$ selectPeer()
5:     send currentModel to $p$

6: procedure ONRECEIVEMODEL($m$)
7:     $m \leftarrow$ updateModel($m$)
8:     currentModel $\leftarrow m$
```

4.1 Machine Learning

In this paper we tackle *supervised classification* that can be defined as follows. We are given a training data set that consists of the training examples. Each training example is a feature vector with a corresponding class label coming from an unknown underlying probability distribution $\mathcal{D}$. Let us denote this training data set by $S = \{(x_1, y_1), \dots, (x_n, y_n)\} \subset \mathbb{R}^d \times C$ where d is the *dimension* of the problem and C is the domain of the class labels. In the case of binary classification, $C = \{-1, 1\}$. The main goal when solving a classification problem is to find a function $f : \mathbb{R}^d \to C$ using the observations from S that can classify *any samples* including those that are not in the training set but that are also generated by the probability distribution $\mathcal{D}$. This 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*.

4.2 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. However, in some scenarios more might be required, for example, we might have to identify the time moment t^* when a sudden drift occurs. This problem is known as the *drift detection* problem.

Let us now elaborate on the types of concept drift that one can observe in the real world. The first type occurs when we are not using all the relevant and important features that are needed to provide accurate prediction. The reason can be that we do not know that the given feature is useful, or we might know it, but we might be unable to have access to the feature value. Examples include the age of users that is known to have an influence [38, 44] or sudden changes in weather conditions [45]. This type of drift can have very diverse time-scales from very quick (an environmental disaster) to very slow (the change of fashion) drift.

The second type of drift has to do with “arms race” situations when an adversary constantly changes strategy to overcome some security measures. The typical examples are the problem of spam filtering [15] and IT security [28]. In these cases, the models used to detect suspicious behavior or spam need to be updated constantly. This type of drift is typically rather slow.

4.3 Gossip Learning: the Vanilla Version

The adaptive algorithm we propose here is based on our Gossip Learning Framework (GoLF) [34, 35]. The skeleton of the original version of GoLF is shown in Alg. 1. This algorithm runs on each node. The algorithm consists of an *active loop* that runs periodically and an event handler (ONRECEIVEMODEL) which is called when a new model arrives.

The models take random walks over the network by selecting a random node (Alg. 1 line 4) and moving there (Alg. 1 line 5). Procedure ONRECEIVEDMODEL updates the received model using the training sample stored by the node (Alg. 1 line 7). It then stores the model as the current model (Alg. 1 line 8). In this skeleton the model is an abstract class which has abstract methods that can be implemented in different ways to instantiate several learning algorithms. The main placeholders are INITMODEL and UPDATEMODEL.

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.

The method SELECTPEER is the interface of the peer sampling service, as described in Section 2. As we mentioned earlier, we use the NEWSCAST algorithm, 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.

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) [34,35]. In this paper we do not discuss these possibilities for clarity but they can be generalized for our adaptive scenario as well.

4.4 Diversity Preserving GoLF

In GoLF, every model that is performing a random walk is theoretically guaranteed to converge so long as we assume that peer sampling works correctly. However, since in each iteration some nodes will receive more than one model, while others will not receive any, and since the number of models in the network is kept constant if there is no failure (since in each iteration all the nodes send exactly one model) it is clear that the *diversity* of models will decrease. That is, some models get replicated, while others "die out". Introducing failure decreases diversity even further, since we can lose models due to message loss, delay, and churn as well, which speeds up homogenization.

This is a problem, because diversity is important when we want to apply techniques such as combination or voting [34, 35]. More importantly, diversity is key to achieve adaptation as well, the goal of this paper. Algorithm 2 contains techniques to deal with this problem. These ideas to maintain diversity were first introduced in [21].

Apart from the two commented lines that deal with concept drift (to be discussed later), the changes w.r.t. the original version are the following. A node sends models in an active cycle only in two cases: it sends the last received model if there was no incoming model during the last 10 active cycles, otherwise it sends all the models received during the last cycle. If there is no failure, then this protocol is guaranteed to keep the diversity of models, since all the models in the network will perform independent random walks. Due to the Poisson distribution of the number of incoming models in one cycle, the probability of bottlenecks is diminishing, and for the same reason the probability that a node does not receive messages for 10 cycles is also practically negligible.

If the network experiences message drop failures or churn, then the number of models circulating in the network will converge to a smaller value due to the 10 cycle waiting time, and the diversity can also decrease, since after 10 cycles a model gets replicated. Interestingly, this is actually useful because if the diversity is low, it makes sense to circulate fewer models and to wait most of the time, since information is redundant anyway. Besides, with reliable communication channels that eliminate message drop (but still allow for delay), diversity can still be maintained.

Finally, note that if there is no failure, Alg. 2 has the same total message complexity as Alg. 1 except for the extremely rare messages triggered by the timeout. In case of failure, the message complexity decreases as a function of failure rate; however, the remaining random walks do not get slower relative to Alg. 1, so the convergence rate remains the same on average, at least if no model-combination techniques are used.

The discussion of the components of Alg. 2 that deal with concept drift is delayed until Section 5.

5 Algorithms

In this section we present two alternative extensions to the original GoLF algorithm. The first one is an extremely simple approach that is independent of the drift pattern. Without trying to detect drift, we maintain a fixed age-distribution in order to keep the adaptivity and diversity of the models at a certain level. In the second approach, we explicitly detect drift via monitoring models and discarding the ones that perform badly. While this is a more complex approach (and therefore potentially more sensitive to the actual drift pattern), it is able to provide information about the actual drift in the system as well.

Algorithm 2 GoLF with drift handling

```
1:  $c \leftarrow 0$
2:  $m \leftarrow$ initModel()
3:  currentModel $\leftarrow$ initDriftHandler($m$)        ▷ drift handling: initialization
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($m$)        ▷ main drift handling method
18:     currentModel $\leftarrow$ updateModel($m$)
19:     receivedModels.add(currentModel)
```

Algorithm 3 AdaGoLF

```
1: procedure INITDRIFTHANDLER($m$)
2:     $m.TTL \leftarrow$ generateTTL()
3:     return $m$
4: procedure DRIFTHANDLER($m$)
5:     $m.age \leftarrow m.age + 1$
6:     if $m.age = m.TTL$ then
7:         $m \leftarrow$ initModel()
8:         $m \leftarrow$ initDriftHandler($m$)
9:     return $m$
```

In Algorithm 2 we show the skeleton of the GoLF algorithm extended with two abstract methods to handle drift: INITDRIFTHANDLER and DRIFTHANDLER. The two approaches mentioned above are both implemented in this framework.

5.1 AdaGoLF: Maintaining a Fixed Age Distribution

It is well known that online algorithms must be less and less sensitive to new samples with time, otherwise they are unable to converge. However, this also means that after a certain point they are not able to adapt to a changing data distribution. To avoid models becoming too old, in our approach we achieve adaptivity by controlling the *lifetime distribution* of the models available in the network.

The implementation of age-based drift handling is shown in Algorithm 3. Note that this approach works *independently* of the learning algorithm applied in GoLF, and independently of the drift pattern as well. From now on, we will call the GoLF framework extended with age-based drift handling ADAGOLF.

The algorithm works by adding 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 3) to a value generated from a predefined probability distribution that we call the Model Lifetime Distribution (MLD). The age of the model increases with each hop. When the model age reaches the TTL value the model is discarded and a new one is created (at line 7 of Algorithm 3) with a newly generated, independent TTL value (at line 8 of Algorithm 3).

The MLD can be selected arbitrarily by a particular implementation to achieve optimal adaptivity. If

Algorithm 4 CDDGoLF

```
procedure INITDRIFTHANDLER($m$)
    $m.history \leftarrow ()$
    return $m$
procedure DRIFTHANDLER($m$)
    $\hat{y} \leftarrow m$.predict($x$)                        ▷ $(x, y)$ is the local sample
    $m.history$.add($\hat{y} = y$ ? 0 : 1)                    ▷ history update
    if driftOccurred($m$) then
        $m \leftarrow$ initModel()
        $m \leftarrow$ initDriftHandler($m$)
    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 $false$
```

the drift pattern is not known then the MLD should have a reasonably heavy tail, so that we always have old models in the system as well as new ones. Such distributions are more robust to the speed and the pattern of concept drift given that a wide range of model ages are always available.

We would like to characterize the distribution of the age of the models in the network at some time point t, given the MLD. Consider 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 $m_1.TTL$, $m_2.TTL, \ldots$ 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:

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

in which case $A_t = t - S_t$. Applying results from renewal theory [19] (the renewal equation and the expectation equation) we get the expected model age and its variance as the time tends to infinity:

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

where random variable X is from the MLD.

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 $\mathbb{D}(A_t) \approx 3454$. This distribution has a reasonably long tail, so we are guaranteed to have old as well as new models at all times. For this reason we expect this distribution to perform well in a wide range of drift scenarios.

5.2 CDDGoLF: Detecting Concept Drift

We propose Algorithm 4 for detecting drift explicitly. Note that the algorithm is still independent of the applied learning algorithm so it can be used along with any GoLF implementation.

We extended the models with a queue of a bounded size, called *history*, that stores performance related data based on the previously seen examples. The main idea is that—when a node receives a model—we can use the sample stored at the node for evaluating the model before we use the sample to actually update

Algorithm 5 Logistic Regression for GoLF

```
1: procedure INITMODEL                                   ▷ model initalization
2:     $m.w \leftarrow (0, \dots, 0)^T$
3:     return $m$
4: procedure UPDATEMODEL($m$)
5:     $\hat{y} \leftarrow m.\text{predict}(x)$                ▷ $(x, y)$ is stored locally
6:     $m.w \leftarrow (1 - \frac{1}{m.age})m.w + \frac{1}{m.age \cdot \lambda}(y - \hat{y})x$
7:     return $m$
8: procedure PREDICT($x$)                                ▷ predicting label
9:     $p_0 \leftarrow 1/(1 + exp(\text{currentModel}.w^T x))$
10:    $p_1 \leftarrow 1 - p_0$
11:    return $p_0 > p_1$ ? 0 : 1
```

the model. Storing these evaluations in the history, we can detect the trend of the performance of the model, in particular, we can detect whether the performance is decreasing.

Let us explain the algorithm in more detail. Procedure DRIFTHANDLER uses the locally stored sample as a test sample and evaluates the model (line 5 of Algorithm 4). Then it updates the model history by storing an error score (value 1 if the predicted and real class labels are different; 0 otherwise) in line 6 of Algorithm 4. Applying this technique we can accumulate a bounded size series of independent error scores in the history. We have to make two important observations 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. In other words, this is merely a heuristic approach to characterize the performance trend. Second, we have to use a bounded size queue to keep the message size constant, since the history is a part of the model, so it is sent through the network each time we pass the model on. In our case we used a history size of 100.

After updating the history, method DRIFTOCCURRED is used to decide whether a concept drift occurred (line 7 of Algorithm 4). In this method we first perform a preprocessing step (line 12 of Algorithm 4) 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 [32] on the smoothed error rates (line 13 of Algorithm 4) to extract the trend in the performance history using the slope of the fitted linear approximation. Finally, we convert the slope value into a drift probability by applying a sigmoid function mapping it onto the interval $(0, 1)$. We then issue a concept drift alert with this probability. The sigmoid function we used is $\sigma_{c,d}(x) = (1 + e^{-c(x-d)})^{-1}$, with parameters $c = 20$ and $d = 0.5$. These parameters were not fine-tuned (noting that $d = 0.5$ is needed for a symmetric mapping).

The geometrical interpretation is that if the slope is negative or 0, then the model in question is still improving (learning phase) or stable (converged phase), respectively. Otherwise, if the slope is positive, then the performance of the model is decreasing, which is most likely due to concept drift.

When drift occurs (i.e. method DRIFTOCCURRED returns *true*), we reinitialize the model, that is, we discard the old one and start a fresh one.

5.3 The Learner Component

So far we have discussed only abstract algorithms that were independent of the actual machine learning algorithm that is implemented in GoLF. However, to evaluate our proposals, we need to implement an actual learning algorithm. In this paper, we opted for logistic regression. In our previous work we have proposed support vector machine (SVM) and boosting implementations as well [21, 35], but any online learning algorithm is suitable that has a constant or slowly growing model size.

Algorithm 5 shows the GoLF implementation of logistic regression [32], a 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

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

Here (x_i, y_i) represents the ith sample from the training database and λ is the regularization parameter (we used $\lambda = 0.0001$). 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 5 starting from line 8).

5.4 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 $N \geq M$ 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.

The space complexity of a model, which directly determines message size, strongly depends on the selected learning algorithm, as well as the dimensionality of the data samples. In the case of a linear model we apply in this paper, the size of a model is the same as the size of a data sample. In addition, a model might also contain drift handling information such as the age value or a performance history of a bounded size.

6 Experimental Setup

6.1 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 unit time interval. The sampling rate defines how many new *independent* samples become available within a unit time interval. We need to stress that only independent samples count in this metric, that is, samples that are drawn from the underlying distribution independently at random.

A third speed-related parameter is the cycle length Δ of GoLF. 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 [46].

The type of the drift is another important property. In our scenarios drift can be *incremental* or *sudden* [46]. The actual drift types are described later in this section.

6.2 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.

Table 1: The main properties of the baseline and the adaptive algorithms.

Name	Computational complexity of learning / cycle	# models used for prediction	Communication complexity / cycle
LOCAL	O(chunkSize)	1	0
CACHEBASED	O(sampleCacheSize)	1	0
VOTE	O(chunkSize)	N	$O(N)$
CACHEDVOTE	O(sampleCacheSize)	N	$O(N)$
VOTE WD	O(chunkSize)	modelCacheSize	$O(1)$
CACHEDVOTE WD	O(sampleCacheSize)	modelCacheSize	$O(1)$
GLOBAL	O(chunkSize*N)	1	$O(N)$
ADAGOLF	$O(1)$	1	$O(1)$
CDDGOLF	$O(1)$	1	$O(1)$

Our simplest baseline is LOCAL 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.

CACHEBASED 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 chunk based as well as the trigger based approaches mentioned in Section 3. 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 LOCAL.

The baselines VOTE and CACHEDVOTE 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 3, hence the performance of these idealized variants can be considered an upper bound of the result of the ensemble based approaches.

VOTE WD and CACHEDVOTE WD are the communication-effective versions of VOTE and CACHED-VOTE (WITH DELAY), 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 adaptive algorithms.

The complexity of these algorithms is summarized in Table 1. 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.

6.3 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 hypercube. 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 moving the hyperplane periodically over time [24]. A hyperplane at time t is defined by its normal

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

where $w_t, w_s, w_d \in \mathbb{R}^d$, $0 \leq \alpha_t \leq 1$, 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

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

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 [6] 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 [16, 42] 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.

6.4 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 and CDDGOLF, 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.

6.5 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. While offline, the nodes retain their state. The length of the online sessions was drawn at random from a lognormal distribution. The parameters of this distribution were given by the maximum likelihood estimation over a private BitTorrent trace called the FileList.org collected by Delft University of Technology [1]. For example, the average session length is around 5 hours. Note, that in mobile phone networks, one can expect similarly long, or even longer sessions on average.

So far our unit of time was Δ (the cycle length), but here we need to specify Δ in order to be able to model churn in simulation. A normal value of Δ in an implementation would be around 10 seconds. However, with an average session length of 5 hours, this would result in practically no churn from our point of view, since convergence occurs at a much faster timescale. So, in order to push our algorithms to the limit, we set Δ to be the average session length, that results in an extreme churn scenario with nodes joining and leaving very frequently. In most of the experiments, except those in Section 7.6, we set the offline session lengths so that at any moment in time 10% of the nodes are offline. In Section 7.6 we experiment also with scenarios where 50% or 80% of the peers are offline.

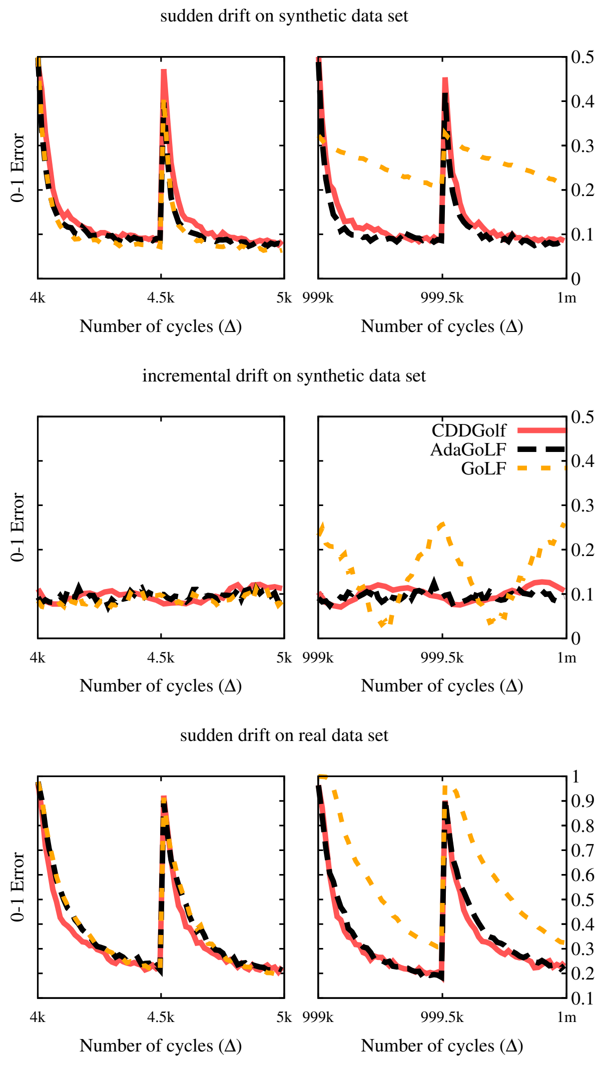

Figure 1: The burn in effect as a motivation for adaptivity.

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.

7 Experimental Results

7.1 Adaptivity

First we 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 GoLF (without drift handling), ADAGOLF, and CDDGOLF as a function of time. We can see that ADAGOLF and CDDGOLF show no burn in effect (these two algorithms have very similar performance producing mostly overlapping curves).

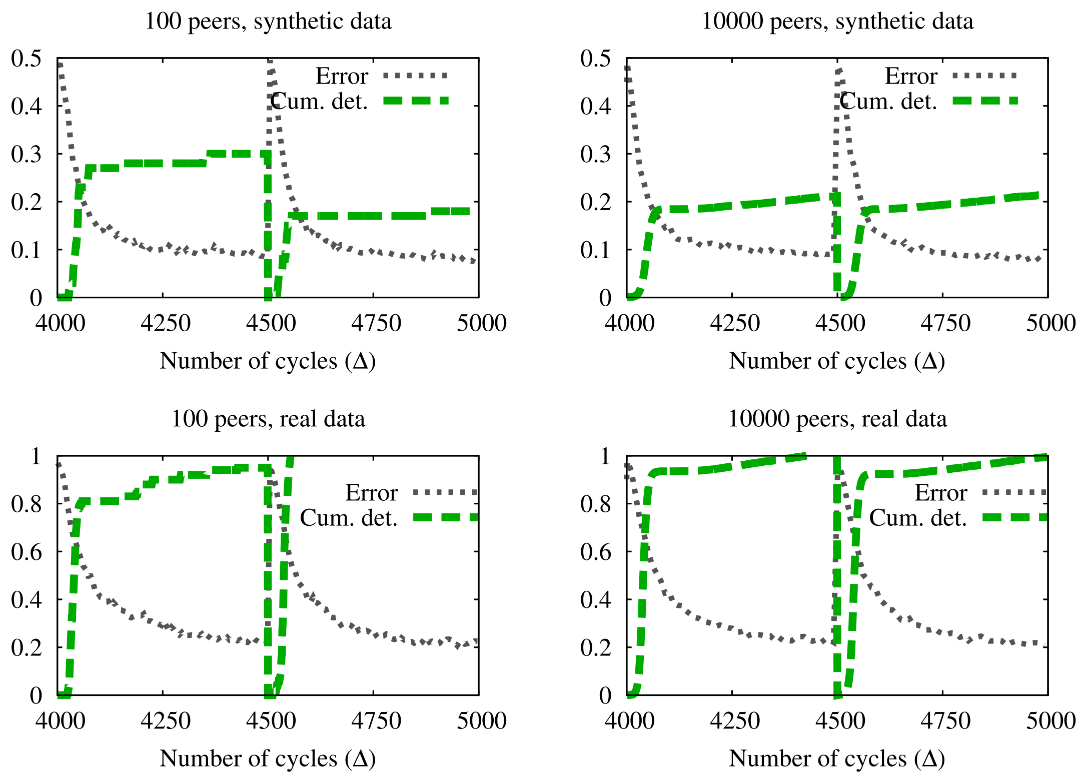

Figure 2: The drift detection and the classification performance of the proposed method on synthetic and real datasets.

7.2 Drift Detection

We start with evaluating the drift detection heuristic of CDDGoLF. Here we focus on the sudden drift scenarios. The results are shown in Figure 2 where the rows represent different databases and the columns represent different network sizes. In each figure we present the average 0-1 error (Error), and the cumulative proportion of drift alarms over all the models that are processed in the network (Cum. det.) as a function of time. This metric is computed by calculating the proportion of those calls to DRIFTHANDLER that result in an alarm in each cycle, and summing these proportions over the cycles up to the next underlying sudden drift, when we reset the sum to zero.

In the case of real world database, the speed of drift detection is extremely fast: the cumulative detection curve increases very sharply when drift occurs. In the case of the synthetic database detection slower. A possible reason is that drift is less dramatic (note that in the synthetic case the maximal error is 0.5, while in the real database it is 1) since half of the labels remain correct after the sudden change. Based on both databases, these results indicate that CDDGoLF detects the drift accurately and quickly.

Let us now turn to the discussion of the effect of network size: We cannot see any significant difference in the error rate between the simulations for sizes 100 and 10,000 (left and right column, respectively). This observation implies the scalability of CDDGoLF. This is not a surprise, since GoLF actually only benefits from scale, since there are more independent learning samples and there is a higher diversity of models.

7.3 The Effect of Sampling Rate

We now turn to the problem of identifying those scenarios, in which GoLF is preferable over the alternative approaches for dealing with concept drift. We focus on sampling rate, as described in Section 6. We

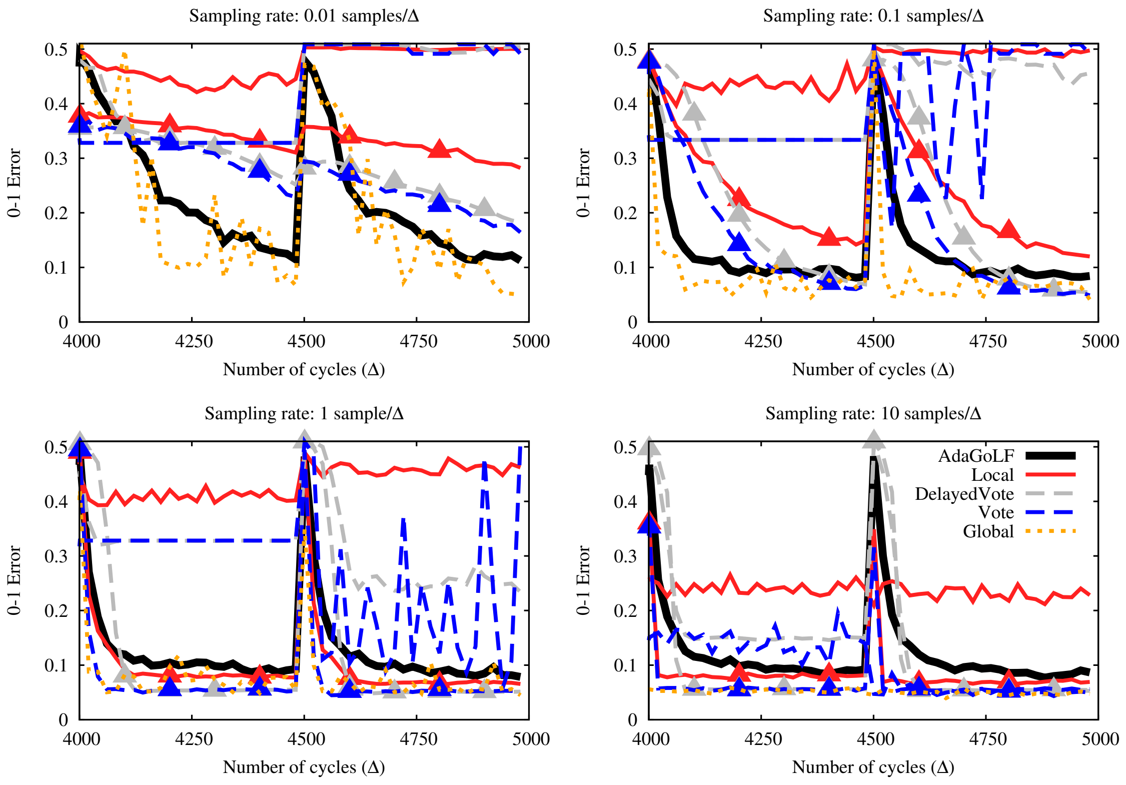

Figure 3: The effect of sampling rate under sudden drift (the lines marked with △ belong to cache based baselines).

performed evaluations using all our database and drift-type configurations (synthetic-sudden, synthetic-incremental and real world-sudden). The results are shown in Figures 3, 4 and 5. In each result set we selected four distinct sampling rates: 0.01, 0.1, 1, 10 samples per cycle.

We have performed these experiments both with ADAGOLF and CDDGOLF. Since these two algorithms show almost identical performance, from now on we include only ADAGOLF in the discussion.

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.

We would like to stress again, that the scenario, in which there are relatively few local *independent* samples relative to the speed of drift is practically very relevant. In the majority of important applications in P2P or mobile networks there is practically one sample throughout the entire participation of a certain user (in which case drift originates mostly from the changing membership). For example, when the data consists of relatively stable personal preferences, stable mobility patterns (work-home), and so on. It is

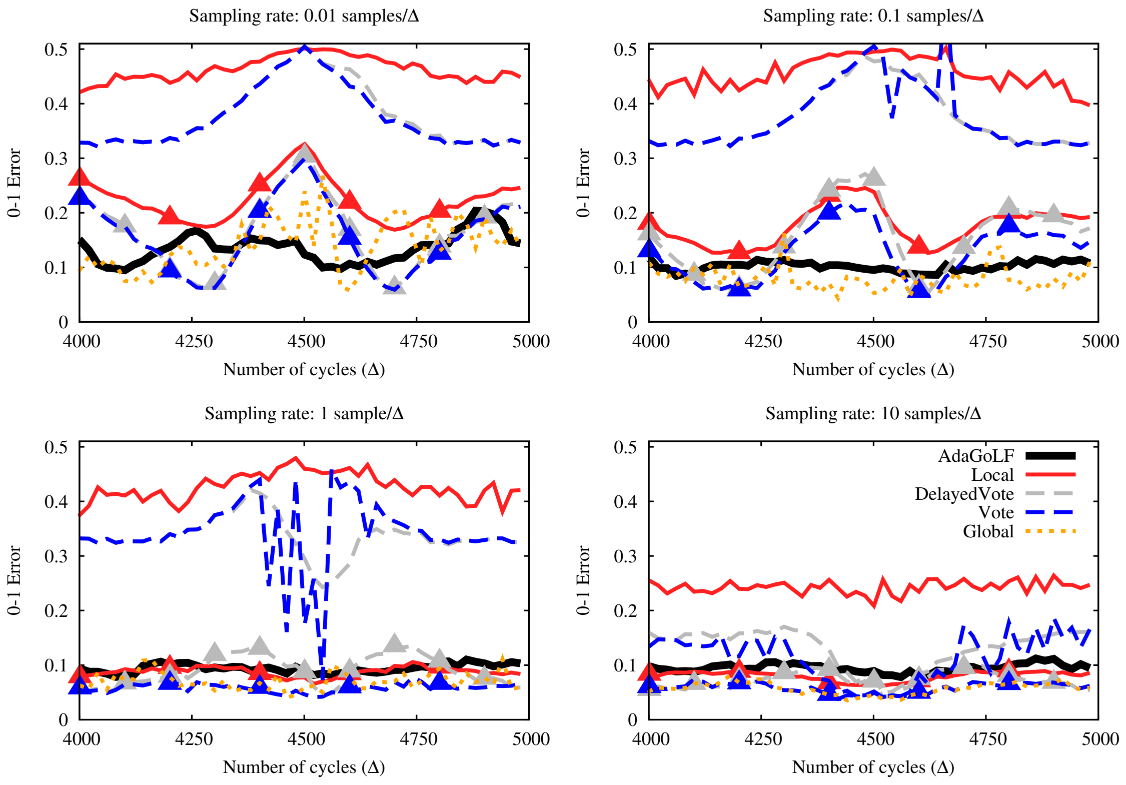

Figure 4: The effect of sampling rate under incremental drift (the lines marked with △ belong to cache based baselines).

important to note that although one covers the work-home path almost each day, we still have only one example, since these repeated examples cannot be considered independent samples from the underlying distribution of all the work-home paths. In addition, many user related events, for example, car accidents, are also very rare relative to drift speed, which can be rather fast due to, for example, weather conditions. With our method—in a very large network—one can still produce and up-to-date accident prediction model based on these examples. Finally, in cases when users need to enter data explicitly, it is very likely that the average rate of samples will be relatively very low.

7.4 Fault Tolerance

We performed simulations with the failure scenarios described previously. Figure 6 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.

7.5 Scalability

In Figure 7 we present the results of ADAGOLF in different network sizes. For CDDGOLF we obtain identical results (not shown). 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.

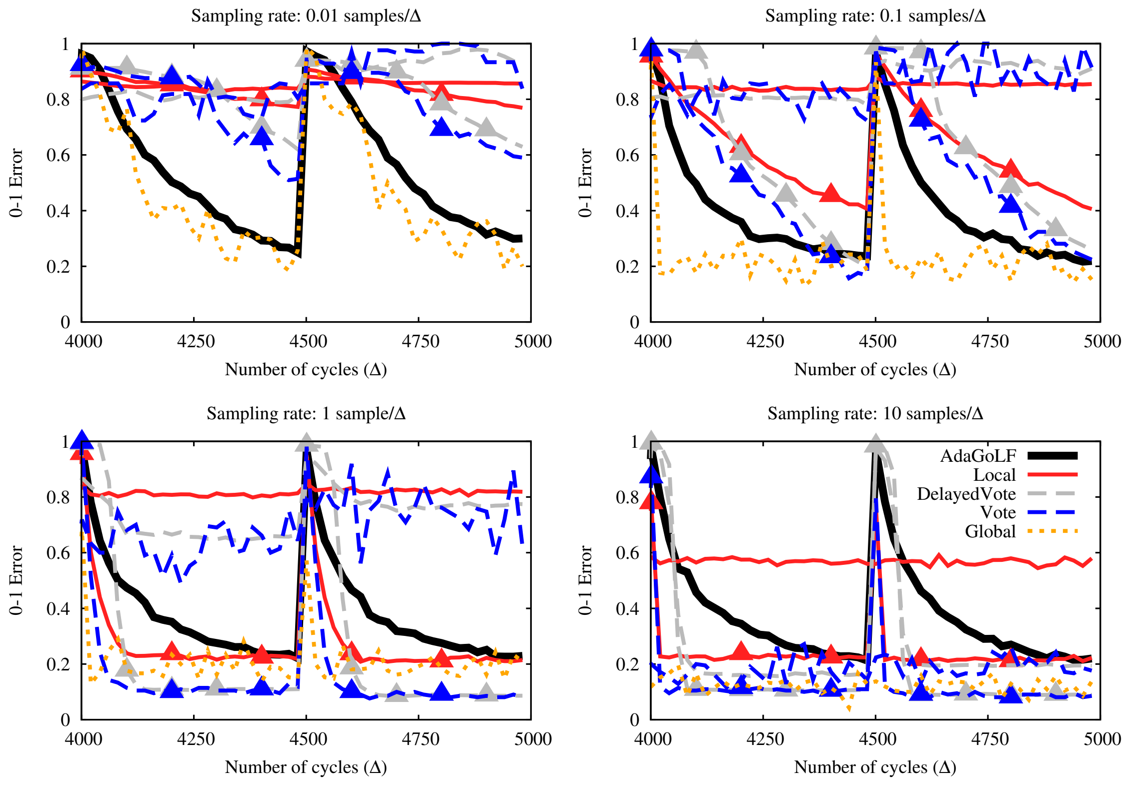

Figure 5: The effect of sampling rate over the real database (the lines marked with △ belong to cache based baselines).

7.6 Churn revisited

Our last set of experiments focuses on the effect of churn on ADAGOLF. Our approach to modeling churn is described in Section 6.5. We briefly mentioned there that we opted for modeling extremely short online session lengths as a worst case scenario. Here we justify this decision by showing that indeed if session lengths are longer (closer to the realistic value) then the performance is very close to that of the scenario without churn. For this reason, we added a scenario where the average session length was 100Δ. Note, that this still results in $\Delta = 3$ minutes, which is still very long.

Figure 8 shows the results. In this figure we show our experiments with 0%, 10%, 50% and 80% of the peers offline on average at any given time. In addition, we also present the 0-1 error over both the set of all the nodes (including those that are offline) and over the set of online nodes.

It is clear from the plot, that increasing the proportion of the nodes that are offline results in a slower convergence if we consider also the nodes that are offline. In the case of long session lengths, this is because nodes that are offline are unable to adapt to concept drift, and hence will have a very poor prediction performance after their models get out of date. However, if we consider only the performance of the online nodes, then the performance remains essentially the same even for the highest proportion of offline nodes. This is because nodes are online long enough (compared to the speed of convergence) so that most models in the system are able to perform long random walks without getting stuck on a node that is going offline before reaching convergence.

In the case of extremely short short session lengths, it does not matter whether we examine online or offline nodes. In scenarios with a high proportion of offline nodes convergence will slow down anyway, because most random walks will be able to make only one or two steps before getting stuck on a node that is going offline, which results in an overall slowdown of the convergence of all the models. Note however, that this scenario is highly unrealistic.

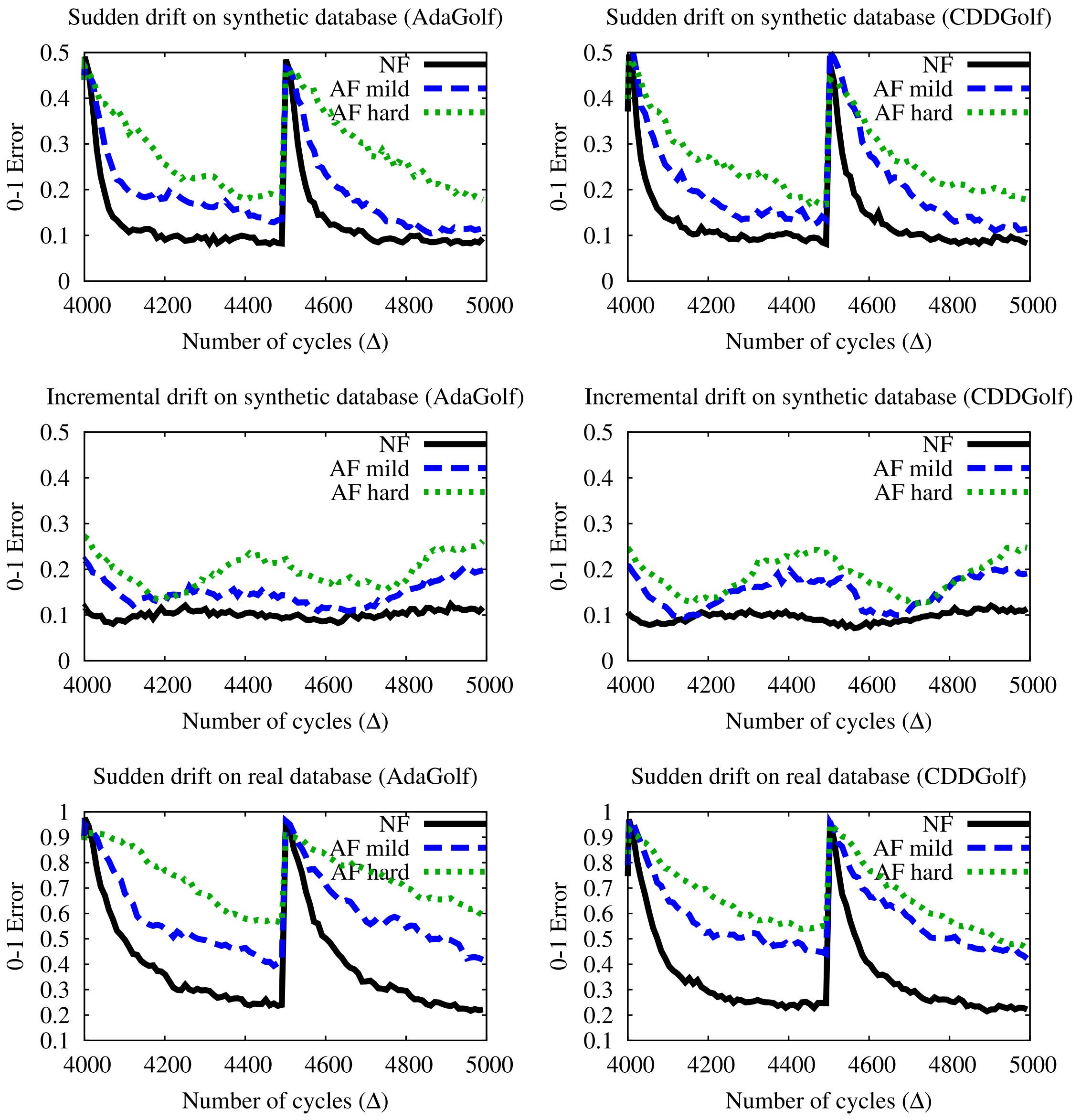

Figure 6: Prediction performance under failure.

Another interesting aspect of churn is its effect on the age distribution of models. The case without churn is discussed theoretically in Section 5. Figure 9 shows the histogram of the model age distribution in the case of churn, averaged over a set of 10,000 snapshots during the simulation, that were taken in every 10th cycle during a simulation of 100,000 cycles. Note that model age is defined by the number of samples a given model has been trained on (as opposed to the number of cycles elapsed after its initialization). We can observe, that the age distribution remains very similar to the case without churn. As before, this is due to the fact that churn events are rare so they do not interfere with the ageing of models much. However, in Figure 10 we can see that if session lengths are extremely short, then the distribution gets biased towards the younger models. Note that the histograms that consider all the nodes (including those that are offline) are very similar (not shown).

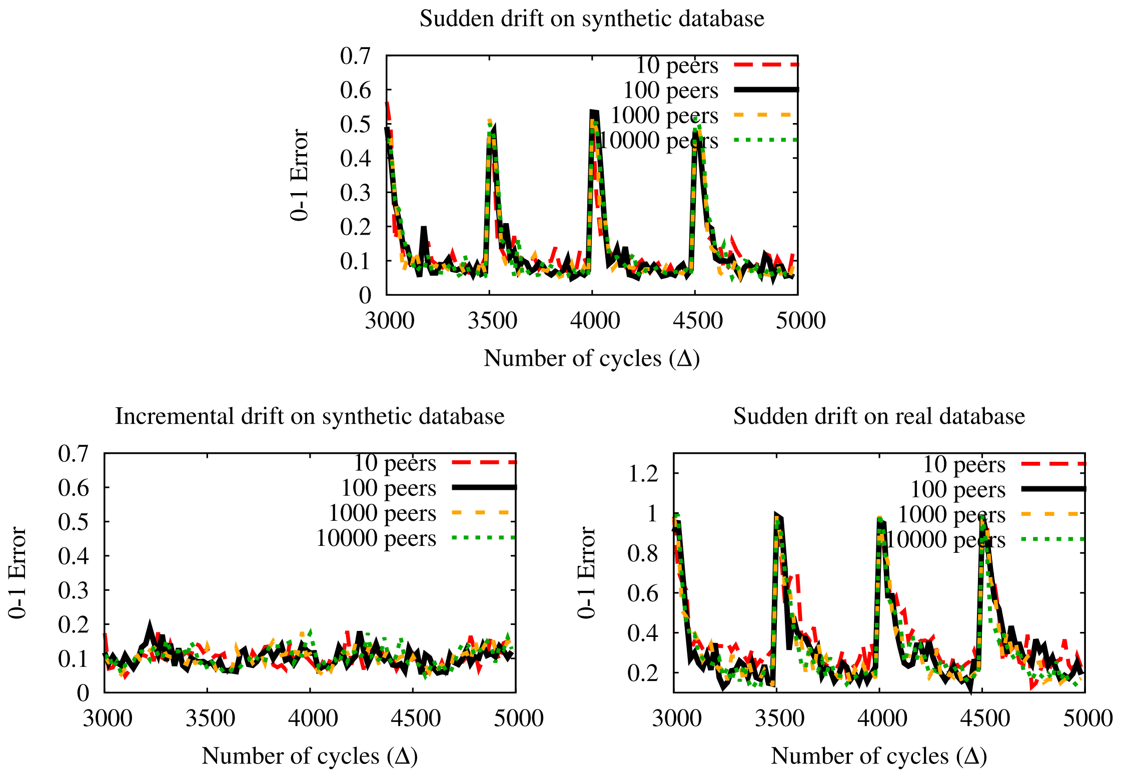

Figure 7: The effect of network size (with sampling rate $1/\Delta$).

8 Conclusion

In this paper we proposed adaptive versions of our GoLF framework: AdaGoLF and CDDGoLF. In the case of AdaGoLF the 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. CDDGoLF also restarts some of the models, but this decision is not blind as in the case of AdaGoLF, but instead it is based on the performance history of the model in question.

We performed a thorough experimental study in which we compared AdaGoLF and CDDGoLF 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, our solutions clearly outperform all the baseline solutions, approximating the "God's Eye view" model, that represents the best possible performance.

One surprising observation is that, although completely blind, AdaGoLF has practically the same performance as CDDGoLF in the scenarios and databases we examined. This suggests that it is an attractive option if we need to issue no explicit alarms in the case of drift. CDDGoLF detects drift explicitly, but its implementation is slightly more complicated, and it could in principle be sensitive to certain patterns of drift since its drift detection heuristic depends on the drift pattern.

We also indicated, that our algorithms 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.

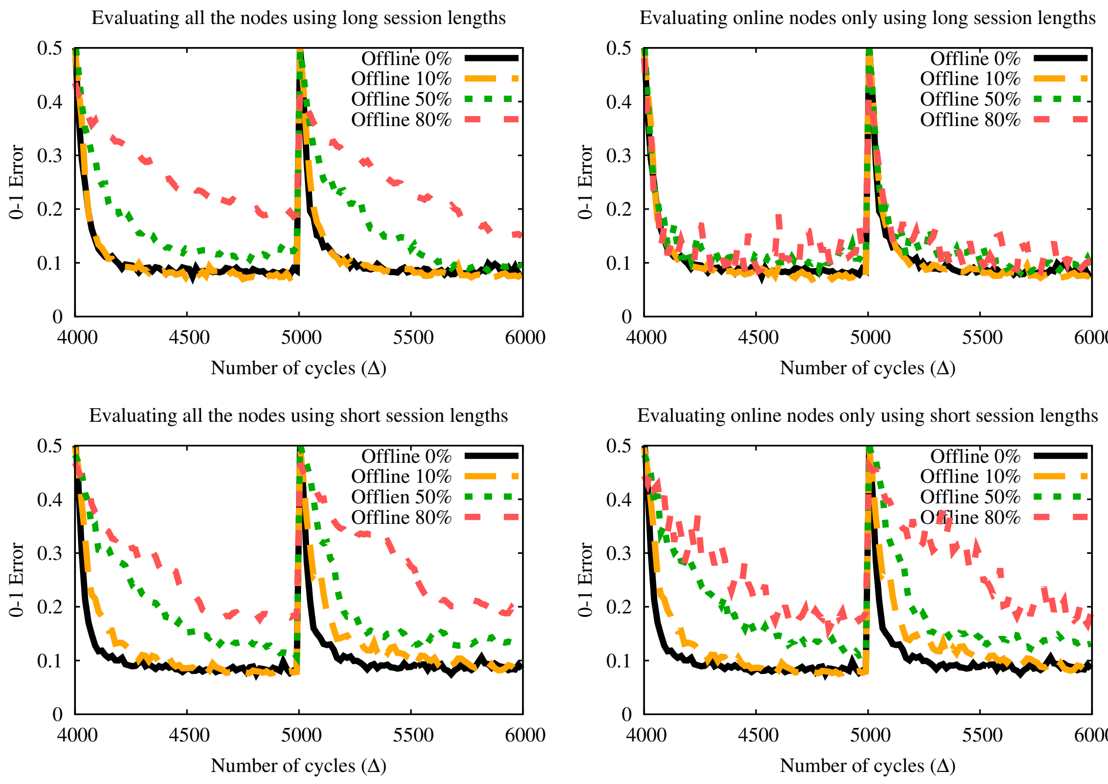

Figure 8: The effect of the churn.

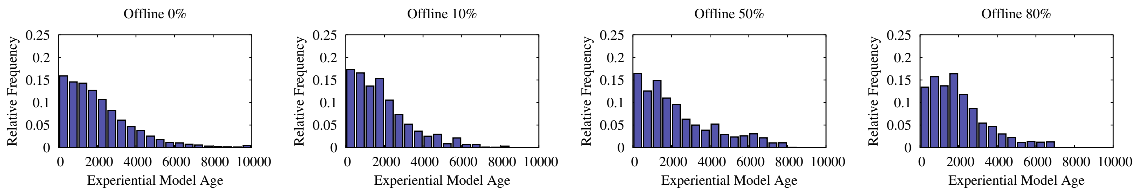

Figure 9: Model age histograms over online nodes for long session lengths.

References

[1] Filelist, `http://www.filelist.org` (2005).

[2] Abdelzaher, T., Anokwa, Y., Boda, P., Burke, J., Estrin, D., Guibas, L., Kansal, A., Madden, S., and Reich, J., Mobiscopes for human spaces, *Pervasive Computing, IEEE* **6** (2007) 20–29.

[3] Ang, H., Gopalkrishnan, V., Hoi, S., and Ng, W., Cascade RSVM in peer-to-peer networks, in *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, eds. Daelemans, W., Goethals, B., and Morik, K., *Lecture Notes in Computer Science*, Vol. 5211 (Springer, 2008), pp. 55–70, doi:10.1007/978-3-540-87479-9_22.

[4] Ang, H., Gopalkrishnan, V., Ng, W., and Hoi, S., Communication-efficient classification in P2P networks, in *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, eds. Buntine, W., Grobelnik, M., Mladenic, D., and Shawe-Ta ylor, J., *Lecture Notes in Computer Science*, Vol. 5781 (Springer, 2009), pp. 83–98, doi:10.1007/978-3-642-04180-8_23.

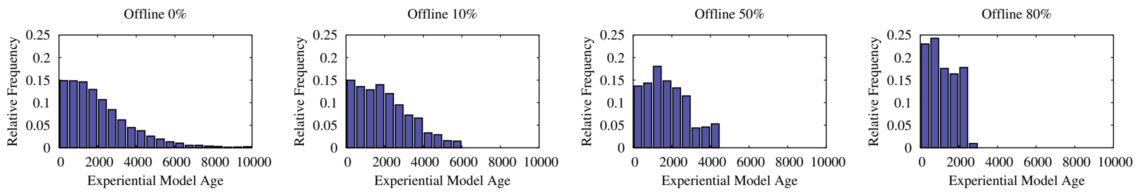

Figure 10: Model age histograms over online nodes for short session lengths.

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

[6] Bache, K. and Lichman, M., UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`.

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

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

[9] Bekkerman, R., Bilenko, M., and Langford, J. (eds.), *Scaling up Machine Learning: Parallel and Distributed Approaches* (Cambridge University Press, 2011).

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

[11] Buchegger, S., Schiöberg, D., Vu, L.-H., and Datta, A., PeerSoN: P2P social networking: early experiences and insights, in *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems (SNS'09)* (ACM, New York, NY, USA, 2009), pp. 46–52, doi:10.1145/1578002.1578010.

[12] Craig, T. and Ludloff, M. E., *Privacy and Big Data* (O'Reilly Media, 2011).

[13] Datta, S., Bhaduri, K., Giannella, C., Wolff, R., and Kargupta, H., Distributed data mining in peer-to-peer networks, *IEEE Internet Computing* **10** (2006) 18–26.

[14] Dean, J. and Ghemawat, S., Mapreduce: simplified data processing on large clusters, *Commun. ACM* **51** (2008) 107–113.

[15] Delany, S. J., Cunningham, P., Tsymbal, A., and Coyle, L., A case-based technique for tracking concept drift in spam filtering, *Know.-Based Syst.* **18** (2005) 187–195.

[16] Dries, A. and Rückert, U., Adaptive concept drift detection, *Stat. Anal. Data Min.* **2** (2009) 311–327.

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

[18] Giannotti, F., Pedreschi, D., Pentland, A., Lukowicz, P., Kossmann, D., Crowley, J., and Helbing, D., A planetary nervous system for social mining and collective awareness, *The European Physical Journal Special Topics* **214** (2012) 49–75.

[19] Grimmett, G. and Stirzaker, D., *Probability and Random Processes*, Texts from Oxford University Press (Oxford University Press, 2001).

[20] Hegedűs, I., Lehel, N., and Róbert, O., Detecting concept drift in fully distributed environments, in *2012 IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics*, SISY'12 (IEEE, 2012), ISBN 978-1-4673-4748-8, pp. 183–188.

[21] Hegedűs, I., Róbert, B.-F., Róbert, O., Márk, J., and Balázs, K., Peer-to-peer multi-class boosting, in *Euro-Par 2012 Parallel Processing*, eds. Kaklamanis, C., Papatheodorou, T., and Spirakis, P., *Lecture Notes in Computer Science*, Vol. 7484 (Springer Berlin / Heidelberg, 2012), ISBN 978-3-642-32819-0, pp. 389–400, doi:10.1007/978-3-642-32820-6_39.

[22] Hegedűs, I., Róbert, O., and Márk, J., Gossip-based learning under drifting concepts in fully distributed networks, in *2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems*, SASO'12 (IEEE, 2012), ISBN 978-0-7695-4851-7, pp. 79–88, doi:10.1109/SASO.2012.13, `http://dx.doi.org/10.1109/SASO.2012.13`.

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

[24] Hulten, G., Spencer, L., and Domingos, P., Mining time-changing data streams, in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01 (ACM, New York, NY, USA, 2001), pp. 97–106, doi:10.1145/502512.502529.

[25] Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., and van Steen, M., Gossip-based peer sampling, *ACM Transactions on Computer Systems* **25** (2007) 8.

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

[27] Lane, N., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., and Campbell, A., A survey of mobile phone sensing, *Communications Magazine, IEEE* **48** (2010) 140–150.

[28] Lane, T. and Brodley, C. E., Approaches to online learning and concept drift for user identification in computer security, in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, eds. Agrawal, R., Stolorz, P. E., and Piatetsky-Shapiro, G. (AAAI Press, 1998), ISBN 1-57735-070-7, pp. 259–263, `http://www.aaai.org/Library/KDD/1998/kdd98-045.php`.

[29] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M., Distributed graphlab: A framework for machine learning and data mining in the cloud, *Proceedings of the VLDB Endowment* **5** (2012) 716–727.

[30] Luo, P., Xiong, H., Lü, K., and Shi, Z., Distributed classification in peer-to-peer networks, in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'07)* (ACM, New York, NY, USA, 2007), pp. 968–976, doi:10.1145/1281192.1281296.

[31] Minku, L., White, A., and Yao, X., The impact of diversity on online ensemble learning in the presence of concept drift, *Knowledge and Data Engineering, IEEE Transactions on* **22** (2010) 730 –742.

[32] Mitchell, T. M., *Machine Learning*, 2nd edn. (McGraw-Hill, New York, 1997).

[33] Montresor, A. and Jelasity, M., Peersim: A scalable P2P simulator, in *Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P 2009)* (IEEE, Seattle, Washington, USA, 2009), pp. 99–100, doi:10.1109/P2P.2009.5284506, extended abstract.

[34] Ormándi, R., Hegedűs, I., and Jelasity, M., Asynchronous peer-to-peer data mining with stochastic gradient descent, in *Euro-Par 2011*, *LNCS*, Vol. 6852 (Springer, 2011), pp. 528–540, doi:10.1007/978-3-642-23400-2_49.

[35] Ormándi, R., Hegedűs, I., and Jelasity, M., Gossip learning with linear models on fully distributed data, *Concurrency and Computation: Practice and Experience* **25** (2013) 556–571.

[36] Pentland, A. S., Society's nervous system: Building effective government, energy, and public health systems, *Computer* **45** (2012) 31–38.

[37] Pouwelse, J. A., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D. H. J., Reinders, M., van Steen, M. R., and Sips, H. J., TRIBLER: a social-based peer-to-peer system, *Concurrency and Computation: Practice and Experience* **20** (2008) 127–138.

[38] Rodenhausen, T., Anjorin, M., Domínguez García, R., and Rensing, C., Context determines content: an approach to resource recommendation in folksonomies, in *Proceedings of the 4th ACM RecSys workshop on Recommender systems and the social web*, RSWeb '12 (ACM, New York, NY, USA, 2012), ISBN 978-1-4503-1638-5, pp. 17–24, doi:10.1145/2365934.2365938, `http://doi.acm.org/10.1145/2365934.2365938`.

[39] Siersdorfer, S. and Sizov, S., Automatic document organization in a P2P environment, in *Advances in Information Retrieval*, eds. Lalmas, M., MacFarlane, A., Rüger, S., Tombros, A., Tsikrika, T., and Yavlinsky, A., *Lecture Notes in Computer Science*, Vol. 3936 (Springer, 2006), pp. 265–276.

[40] Street, W. N. and Kim, Y., A streaming ensemble algorithm (sea) for large-scale classification, in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01 (ACM, New York, NY, USA, 2001), pp. 377–382, doi:10.1145/502512.502568.

[41] Tölgyesi, N. and Jelasity, M., Adaptive peer sampling with newscast, in *Euro-Par 2009*, *LNCS*, Vol. 5704 (Springer, 2009), pp. 523–534, doi:10.1007/978-3-642-03869-3_50.

[42] Vreeken, J., van Leeuwen, M., and Siebes, A., Characterising the difference, in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07 (ACM, New York, NY, USA, 2007), pp. 765–774, doi:10.1145/1281192.1281274.

[43] Wang, H., Fan, W., Yu, P. S., and Han, J., Mining concept-drifting data streams using ensemble classifiers, in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03 (ACM, New York, NY, USA, 2003), pp. 226–235, doi:10.1145/956750.956778.

[44] Webb, G. I., Pazzani, M. J., and Billsus, D., Machine learning for user modeling, *User Modeling and User-Adapted Interaction* **11** (2001) 19–29.

[45] Wei, G., Zhang, T., Wu, S., and Zou, L., An ensemble classifier method for classifying data streams with recurrent concept drift, in *Proceedings of the 4th International Conference on Awareness Science and Technology (iCAST)* (2012), pp. 3–9, doi:10.1109/iCAwST.2012.6469580.

[46] Zliobaite, I., Learning under concept drift: an overview, Technical Report 1010.4784, arxiv.org (2010), `http://arxiv.org/abs/1010.4784`.