Predictive Handling of Asynchronous Concept Drifts in Distributed Environments

Hock Hee Ang, Vivekanand Gopalkrishnan, Indrė Žliobaitė, Mykola Pechenizkiy, Steven C.H. Hoi

Abstract—In a distributed computing environment, peers collaboratively learn to classify concepts of interest from each other. When external changes happen and their concepts drift, the peers should adapt to avoid increase in misclassification errors. The problem of adaptation becomes more difficult when the changes are asynchronous, i.e., when peers experience drifts at different times. We address this problem by developing an ensemble approach, `PINE`, that combines reactive adaptation via drift detection, and proactive handling of upcoming changes via early warning and adaptation across the peers. With empirical study on simulated and real world datasets, we show that `PINE` handles asynchronous concept drifts better and faster than current state-of-the-art approaches, which have been designed to work in less challenging environments. In addition, `PINE` is parameter insensitive and incurs less communication cost while achieving better accuracy.

Index Terms—Classification, Distributed Systems, Concept Drift.

1 Introduction

Distributed classification [1], [2], [3], [4] is the setting where nodes/peers collaboratively learn from each other in order to improve the classification accuracy on their respective data. This learning task is challenging specially where there is a massive number of arbitrarily connected and dynamic peers [5]. Hence an ideal distributed classification scheme should typically be: anytime (able to produce an answer at any time), autonomous (non-blocking), decentralized (no single point of failure) highly scalable (able to handle enormous number of peers), tolerant of peer failures (prevent failures from being catastrophic) and privacy preserving (hide private data of the peers).

Besides the above mentioned challenges, another critical challenge for distributed classification is *concept drift* [2], which is an important problem in many data mining and machine learning applications. Concept drift refers to situations when the data distribution changes over time unexpectedly in unforeseen ways. Given an occurrence of concept drift, classifiers will experience a loss in accuracy from the time the concept changes until the time the classifiers adapt to the new data distribution (if they adapt).

Attention to such learning scenarios has been rapidly increasing in the last few years [6], [7], [8], [9], [10]. Concepts drift in a number of dynamic environments, such as data streams, distributed systems, and affect many applications, such as network intrusion detection, spam categorization, fraud detection, epidemiological, climate or demographic data, marketing and web analytics, financial analysis and many more. It is crucial to address the concept drift problem in distributed classification, as the inability to adapt swiftly to the drifted concept often results in significant loss of classification accuracy.

Although concept drift has been actively studied in typical centralized settings, the phenomenon exhibits fundamental differences in distributed environments. While typical scenarios only model concept drift from a single source of data, in distributed networks, each peer can be viewed as an independent data source. In order for peers to benefit from knowledge sharing, we assume that data streams of a subset of peers follow the same unknown probability distributions where the concept may change from time to time. However, we cannot assume that the concepts always change instantaneously for the entire set of peers. Data is not i.i.d. across the peers at a given point in time, but data across a subset of peers will be identically distributed if we look over very long period of time.

Examples of such distributed environments include disease diagnosis during epidemic outbreaks where the epidemic originates from one country and spreads to another with some delay, or weather prediction across different regions where a cyclone brings

- *H.H. Ang and S.C.H. Hoi are with the School of Computer Engineering, Nanyang Technological University, Singapore.*
Email: {angh0024,chhoi}@ntu.edu.sg
- *V. Gopalkrishnan is with Deloitte Analytics Institute Asia, Singapore.*
E-mail: vivek@deloitte.com
- *I. Žliobaitė is with Smart Technology Research Centre, School of Design, Engineering and Computing, Bournemouth University, Poole, Dorset, UK.*
E-mail: izliobaite@bournemouth.ac.uk
- *M. Pechenizkiy is with Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands.*
Email: m.pechenizkiy@tue.nl

Manuscript received XXX; revised XXX; accepted XXX; published online XXX.
Recommended for acceptance by XXX.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-XXXXXXXXX.
Digital Object Identifier no. XXXXXXXXXX.

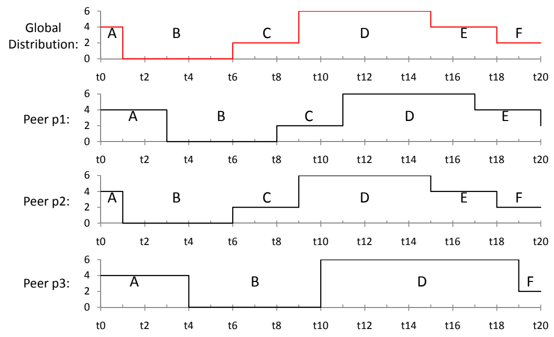


Fig. 1. The global distribution of data and local distributions at peers. The horizontal axis represents time, the vertical axis represents data mean. A, B, C, D and E represent different concepts (environments).

weather changes while moving from region to region with a certain speed and direction. In these examples, the concept drifts from peers in some geographical location to other peers in other locations. Although global changes happen and need to be handled unexpectedly, at the local peer level adapting to changes can be made more effective if the first peers to 'suffer' can share their knowledge with other peers in a controlled manner. Fig. 1 illustrates the case of *asynchronous concept drift*, where peers encounter the same concepts, but with different delays. We can see that peer p_2 experiences concept drift identical to the global distribution whereas peer p_1 experiences the same concept drifts two time steps later. Peer p_3 always experiences concept drifts later than peer p_2, but faster or slower than peer p_1 and misses some concepts. It is obvious that some (temporal) association exists between peers p_1 and p_2. The drifts that happen at individual peers will be referred to as *local drifts* and the drift that happens at the system level will be referred as *global drift*. Hence, mining and exploiting such associations to improve both the detection of and adaptation to the concept drifts is an essential part in handling concept drifts in a distributed environment.

An ideal classification algorithm that deals with the problem of concept drift in distributed networks should possess the aforementioned desirable properties for learning in a distributed setting, and also be able to adapt swiftly to the changes in concepts, without adversely affecting the peers. Although there has been much work on distributed classification, most do not address the problem of concept drift [1], [3]. Others assume the concept drift scenario of a centralized setting (i.e., all peers are affected in the same manner at the same time) [2] or are slow to adapt, inaccurate in adaptation and requires a large number of user inputs [4].

Most of the existing approaches that handle concept drift [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16] are based on a centralized setting and cannot be easily adapted for a distributed environment. While the existing ensemble-based solution seems to be a viable option, they are not designed to be efficient for the demanding environment of distributed networks. Also, they do not consider the problem of asynchronous concept drifts that may occur in a distributed environment and thus do not fully exploit the environmental information to improve concept drift detection and adaptation.

To address the above challenges, we present a novel concept drift adaptation framework, `PINE`, for performing distributed classification in distributed environments. The key features of our framework are:

- it integrates both reactive and proactive adaptation techniques, which predict the occurrences of concept drifts and future concepts from historical trends observed in the distributed network;
- the proposed approach is parameter insensitive;
- adaptation to concept drift is achieved by both drift detection and balancing of the reactive and proactive adaptive classifiers.

Within our framework we design a novel distributed classification approach, which handles concept drift in the distributed environment. We empirically demonstrate that `PINE` outperforms the existing solutions on both synthetic and real world datasets. Our study makes fundamental contributions to distributed classification in changing environments by relaxing the unrealistic assumption of synchronous drifts in distributed environments, as well as contributes a principal extension of handling concept drift in machine learning under concept drift by addressing the distributed classification scenario.

The remainder of this paper is organized as follows. Section 2 introduces the formal setting of the problem. In Section 3 we present our framework for learning with drifting concepts in distributed networks. Section 4 reports the results of empirical evaluation of the proposed framework. Section 5 discusses related work and Section 6 concludes the study.

2 Problem Statement

In this section we introduce the formal setting of classification in a distributed environment.

Consider a typical classification problem, where the task is to produce a mapping from an input vector $\mathbf{x} \in \mathcal{X}$ in d-dimensional space to its class label $y \in \mathcal{Y}$, where $\mathcal{Y}$ denotes the class label space; e.g., $\mathcal{Y} = \{+1, -1\}$ for binary classification. The mapping is then applied for classifying unseen data that is assumed to be drawn from the same underlying distribution. A pair of an input vector and its corresponding class label make an instance $(\mathbf{x}, y)$. A set of instances make a dataset. Let $\mathbf{D} = [\mathbf{x}_1, \ldots, \mathbf{x}_\ell]^{\mathrm{T}} = [(x_{i,1}, \ldots, x_{i,d})_{i=1}^{\ell}]^{\mathrm{T}}$ be a set of input data, and let $\mathbf{y} = [y_1, \ldots, y_\ell]^{\mathrm{T}}$ be a set of corresponding class labels, where ℓ denotes the total number of training data instances, d denotes the dimensionality of the input data points.

This classification problem extends to a distributed setting as follows. Suppose there are N peers in the distributed network, each peer p is a partition $\mathbf{D}_p$ of $\mathbf{D}$ where $\ell = \sum_p \ell_p$. The goal of distributed classification is to collaboratively learn a global prediction function $f : \mathcal{X} \to \mathcal{Y}$ from the training data of all peers, which maps a d-dimensional vector $\mathbf{x} = (x_1, \ldots, x_d)^T \in \mathcal{X}$ to a corresponding class label $y \in \mathcal{Y}$ of a target concept.

In the typical (stationary) classification scenario each training instance $(\mathbf{x}, y)$ is drawn from some unknown but i.i.d. distribution $P(\mathbf{x}, y)$. In a non-stationary environment the target concept is expected to change over time. *Concept drift* is defined as the change of the underlying unknown probability distribution, i.e., $P_k(\mathbf{x}, y) \neq P_j(\mathbf{x}, y)$, which has occurred from time t_k to $t_j (t_k < t_j)$. As a result, the global mapping $f : \mathcal{X} \to \mathcal{Y}$ that has been learned at time t_k may be no longer accurate at time t_j, where $t_j > t_k$. In a changing environment the training data of a peer p can be represented as $\mathbf{D}_p = \{D_p^1, \ldots, D_p^j\}$, where t_j is the current time, and D_p^j is drawn from some unknown probability distribution $P_j(\mathbf{x}, y)$. In this setting given an optimal prediction function f_k^* for $P_k(\mathbf{x}, y)$, it is no longer optimal for $P_j(\mathbf{x}, y)$, i.e. we get $Err(f_k^*, \mathbf{D}_p^k) < Err(f_k^*, \mathbf{D}_p^j)$, where $Err(f, \mathbf{D}) = (\sum_{\mathbf{x}_i \in \mathbf{D}, \mathbf{y}_i \neq f(\mathbf{x}_i)} 1)/|\mathbf{D}|$ is the error rate of f on $\mathbf{D}$.

Hence the ultimate goal of distributed classification in a changing environment is to minimize the error ε for all peers over time

$$\varepsilon = \sum_{p,t} Err(f_p, \mathbf{D}_p^t) \tag{1}$$

while satisfying the constraints of the distributed environments, where f_p is the most up-to-date classification model (potentially consisting of a combination of multiple classifiers) of peer p at the time $\mathbf{D}_p^t$ is to be classified.

3 PINE FRAMEWORK

In this section, we propose a novel framework called *Predictive and parameter INsensitive Ensemble* (PINE) for handling asynchronous drifting concepts in distributed networks. We start with a brief overview of the proposed framework followed by detailed discussions on each component of our framework.

3.1 Overview of PINE

The sequence diagram of our framework from one peer p perspective is presented in Fig. 2. The peer p in our system monitors its own data stream $\mathbf{D}_p$, and maintains its own ensemble of reactive predictive models denoted as $\mathbf{REM}_p$ and an ensemble of proactive predictive models denoted as $\mathbf{PEM}_p$. The reactive models are the classifiers that represent peer p's current data distribution, and the proactive models

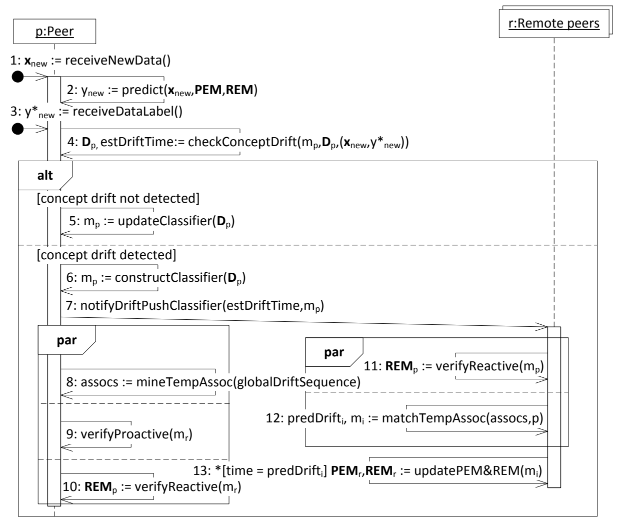

p:Peer
r:Remote peers
1: x_new := receiveNewData()
2: y_new := predict(x_new,PEM,REM)
3: y*_new := receiveDataLabel()
4: D_p, estDriftTime:= checkConceptDrift(m_p,D_p,(x_new,y*_new))
alt
[concept drift not detected]
5: m_p := updateClassifier(D_p)
[concept drift detected]
6: m_p := constructClassifier(D_p)
7: notifyDriftPushClassifier(estDriftTime,m_p)
par
8: assocs := mineTempAssoc(globalDriftSequence)
9: verifyProactive(m_r)
10: REM_p := verifyReactive(m_r)
par
11: REM_p := verifyReactive(m_p)
12: predDrift_i, m_i := matchTempAssoc(assocs,p)
13: *[time = predDrift_i] PEM_r,REM_r := updatePEM&REM(m_i)

Fig. 2. The proposed framework as a sequence diagram.

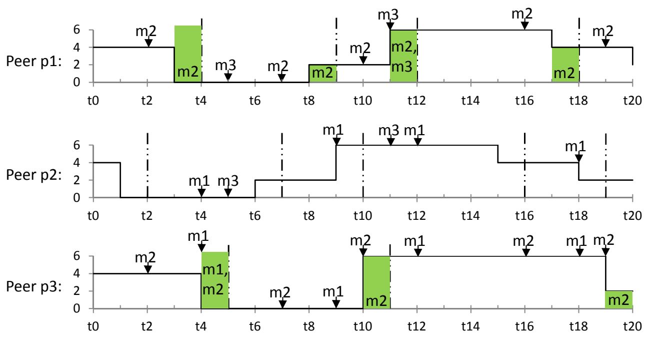


Fig. 3. An illustration of optimal *proactive* concept drifts for two peers. The shaded areas indicate non detected concept drift periods and the labels m_i indicate the optimal proactive models, where i is the peer number from Fig. 1.

are the classifiers that may represent peer p's future data distribution which is different from the current distribution. Whenever a new unseen data instance $\mathbf{x}_{new}$ arrives, peer p consults both $\mathbf{REM}_p$ and $\mathbf{PEM}_p$ and from a combination of their outputs predicts the class label $\hat{y}_{new}$ (cf. Section 3.7).

The idea and concepts of the proactive and reactive models is illustrated in Fig. 3 and 4 respectively, each of which follows the example shown in Fig. 1. In Fig. 3 and 4, we illustrate the change in concepts of peers (change in y axis value), the concept drift detection and adaptation (vertical dotted line) and propagation of the newly trained models to other remote peers (labelled arrows). For instance, peer p_1's concept drifted at time t_3 (drop in y axis value), was detected and adapted at time t_4 (vertical dotted line).

In addition, Fig. 3 shows the optimal model that should be used before a peer's concept drift is detected and Fig. 4 shows the optimal model(s) that should be used after a peer's concept drift is detected,

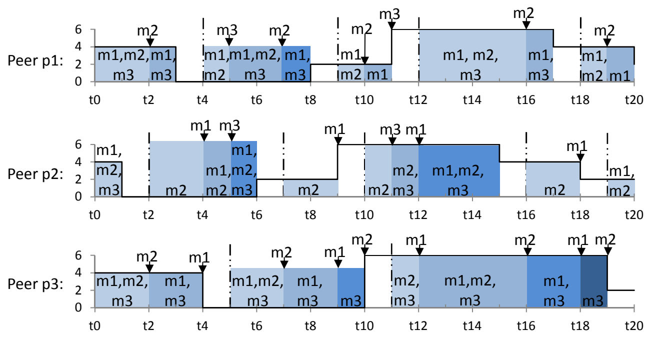


Fig. 4. An illustration of optimal *reactive* concept drifts for two peers. The shaded areas and their labels indicate optimal reactive models.

which is indicated in both figures by the labels in the shaded areas. The example for proactive model presented in Fig. 3 is as follows. Peer p_1's concept has drifted at time t_3 and was detected at time t_4. Hence, between time t_3 and t_4, peer p_1's current model is not suitable for predicting its own data since the model is made outdated by the concept drift. Similarly, peer p_3 is also not suitable. As such, from time t_3 to t_4, peer p_2's model should be the most accurate (optimal) for predicting p_1's unlabelled data.

Similarly, the example for reactive model presented in Fig. 3 is as follows. Peer p_1 concept drift was detected in time t_4 and hence, from time t_4 to t_5, the best models for predicting peer p_1 unlabelled data are the current models of peers p_1 and p_2. At time t_5, peer p_3 detected its concept drift and updated its model which it propagated to peer p_1. Hence, the optimal models for p_1's unlabelled data are those of peers p_1, p_2 and p_3. However, at time t_7, peer p_2 adapted to a new concept which is different from that of peer p_1. Hence, peer p_2's current model at time t_7 is no longer suitable for peer p_1 and the optimal models for p_1's unlabelled data are models of p_1 and p_3.

Therefore, the aim of our framework is to ensure timely supply of the optimal models for each peer either locally or 'borrowed' from other peers (combined of Fig. 3 and 4).

A walk-through of the various components in our framework illustrated in Fig. 2 is as follows.

Methods 1 & 2. First, whenever an unlabelled data point $\mathbf{x}_{new}$ is received ($receiveNewData$) by peer p, its label y_{new} is predicted using the models in $\mathbf{PEM}_p$ and $\mathbf{REM}_p$.

Methods 3 to 7. Once the true label $y*_{new}$ of $\mathbf{x}_{new}$ arrives ($receiveDataLabel$), peer p then checks if has have been a concept drift for the received training examples ($checkConceptDrift$). If the concept has not drifted, p then updates its training dataset D_p that represents the current concept and updates the local classifier m_p ($updateClassifier$). Otherwise, if the concept has drifted, peer p then updates D_p by removing data of the old concept and moving them to $\hat{D}_p$ which represents the old concept. A new local classifier m_p is then built from D_p ($constructClassifier$). Next, peer p notifies all other peers r_i of the time its concept drifted and sends the new local classifier m_p to them ($notifyDriftPushClassifier$). The details of training local classifiers and handling concept drift are presented in Section 3.2.

Method 8. To exploit the proactive models from other peers, peer p mines the temporal association rules from the sequence of concept drift occurrences ($mineTempAssoc$), which are used for concept drift prediction. These rules consist of a sequence of peers whose concept drift occurrences were followed by the local concept drift at peer p (cf. Section 3.3).

Methods 9 & 10. Then peer p tests all remote classifiers m_{r_i} using $\mathbf{D}_p$ and $\hat{\mathbf{D}}_p$ to verify if there are accurate proactive classifiers ($verifyProactive$). If accurate, the remote classifiers are flagged as potential proactive models (cf. Section 3.4). In addition, it also verifies if the remote classifiers m_{r_i} are reactive classifiers ($verifyReactive$) and adds to $\mathbf{REM}_p$ if accurate (cf. Section 3.5). This marks the end of all tasks required to be performed after the local concept drift occurrence.

Methods 11 to 13. Next, whenever a remote peer r_i receives a notification of concept drift from peer p and its new classifier m_p, r_i then tests m_p using $\mathbf{D}_{r_i}$ to verify if it is a reactive classifier ($verifyReactive$) and adds a copy to its $\mathbf{REM}_{r_i}$ if accurate. In addition, peer p's time of concept drift is recorded and checked against the existing temporal association rules to predict if local concept drift is likely to occur in peer r_i ($matchTempAssoc$). When a rule is fully matched, peers listed in the rule sequence are marked as potential proactive classifiers and time-stamped at time concept drift of r_i is predicted (cf. Section 3.6).

At the time point when a local concept drift at peer p is predicted by a temporal association rule ($updatePEM\&REM$), remote classifiers m_{r_i} of peers listed in the rule are then added to $\mathbf{PEM}_p$ if they are valid proactive classifiers and removed from $\mathbf{REM}_p$ (cf. Section 3.6).

3.2 (Re-)Training Local Classifiers

This section presents details of how local classifiers are (re-)trained following arrival of every true label depending on whether a concept drift has been detected (**Methods 5 & 6**). The detection of concept drift will be discussed in the next section.

In principle, any classification algorithm can be employed at peer p. However, since we consider distributed network settings, we select an algorithm that has two desired properties – low time complexity to deal with the possible large amount of data and low model propagation cost since the models need to be propagated to other peers. Hence, in this study, we recommend and use the state-of-the-art linear SVM classifier.

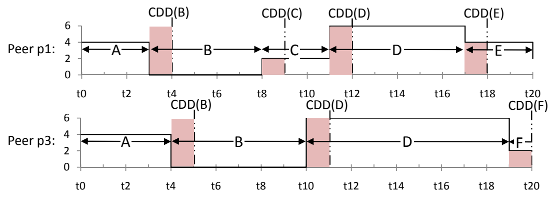


Fig. 5. An example lag in drift detection. $CDD(B)$ represents the detection of concept drift of concept B. Shaded area indicates the transition period where the concept drift has changed but has not yet been detected.

As new labelled instances arrive, the performance of a local classifier m_p is monitored for detecting concept drift. If no change is detected, then the classifier is incrementally updated with new labelled data. Otherwise, if a change is detected, the change detector points out the time when drift occurred. In such a case a new classifier is trained with the data accumulated after the occurrence of the drift. By doing so, we can ensure that the prediction on new data will not be affected by the old concept, which will lower the prediction accuracy. This is a common practise among ensemble based solutions which deal with concept drifts [6], [10], [13], [17], [18]. This classifier is sent to other remote peers that are likely to experience this drift in the (near) future.

Locally learnt base classifier at peer p can become part of REM_{ri} at any remote peer r_i. In order to keep the communication costs low, we allow p to propagate its updated classifier only when the sample size used for inducing the local base classifier increases in geometric progression as compared to the previous sample size, so that the improvements are substantial.

3.2.1 Detecting Concept Drift

In this section we give the details of how concept drift detection is organized in our framework.

A change detector is a function that monitors the historical data and signals the occurrence of concept drift. A detector needs to see a number of instances that represent a new concept before it can detect the drift. The period from the time when the actual drift occurred to the time when it was detected is called the *detection lag*. A good change detector would detect drifts accurately with as few false positives and as small detection lag as possible. Fig. 5 illustrates the setting of concept drift detection in our distributed classification example. We can see the detection lags shaded in grey. We need a change detector to estimate the time of actual concept drift occurrence, which is the arrival time of the oldest data point in the new concept. This allows us to timely deploy the proactive models whenever concept changes as illustrated in Fig. 3.

In our framework, we detect concept drift (**Method 4**) for each peer using a modified version of the ADWIN algorithm [19], which has sound theoretical justifications and uses only a few parameters. Given a sequence of data points, ADWIN splits the sequence into all possible pairs of sequential windows and compares the means of data within the windows. When the absolute differences of the means are significant, the oldest element is dropped and the comparison repeats until no significant difference is found.

More formally, suppose μ_1 and μ_2 are the means of the two sub-sequences as a result of a split. Then the criterion for signaling a change is $|\mu_1 - \mu_2| > \epsilon_{cut}$, where

$$\epsilon_{cut} = \sqrt{\frac{1}{2\mu}\log\frac{4n}{\delta}}, \quad \mu = \frac{1}{\frac{1}{n_1}+\frac{1}{n_2}}, \tag{2}$$

here n is the length of the full sequence, while n_1 and n_2 are lengths of the sub-sequences respectively. Note that $n = n_1 + n_2$, and $\delta \in (0,1)$ is a hyper-parameter of the model.

Our modifications to ADWIN and motivation behind these modifications are as follows. First, in the original ADWIN comparisons are done over the input data (attribute by attribute), while we run the detection over a stream of *accuracies* represented as ones for correct and zeros for wrong predictions). We do this modification, as classification accuracy is our main concern and evaluation criteria, besides, in some cases concept drifts may occur without any change in the distribution of the attributes (the real concept drift).

Second, we consider concept drift to occur when the mean of the first window is *significantly larger* than that of the second window, i.e., $\mu_1 - \mu_2 > \epsilon_{cut}$.

Third, to speed up the detection, instead of dropping a single element from a window, we drop *all elements* in the previous window.

Note that even though a concept drift is detected and old data may seem not necessary anymore, in our framework every peer p maintains two sets of data $\mathbf{D}_p^{t-1}$ and $\mathbf{D}_p^t$ corresponding to the previous and the current concept. This is necessary for the proactive model verification (cf. Section 3.4). In this work, the size of the two datasets $\mathbf{D}_p^{t-1}$ and $\mathbf{D}_p^t$ are unbounded as we did not find it necessary to limit their sizes. The average size of $\mathbf{D}_p^{t-1}$ and $\mathbf{D}_p^t$ are less than 2000 over all the experiments that we conducted, which is not an unreasonable size to maintain. However, bounding of the dataset size should only have a negligible effect, and if the bounding window is large enough to represent the concept (e.g., 1000 to 10000), we do not expect any adverse effects in bounding the window size.

3.3 Mining Temporal Associations

This section describes how the temporal associations of concept drifts among peers are mined in our framework (**Method 8**).

The temporal associations of the concept drift occurrences among peers can allow us to proactively predict occurrence of drifts in peers. Hence minimize the lag between the drift and adaptation of the models. In order to obtain updated temporal associations, the temporal association rules are mined whenever a local concept drift is detected and the old rules mined in the previous concept drift are then forgotten and replaced with the new rules mined.

With reference to the time notation in this paper, an important point to note here is that while PINE requires all peers to share a global time, no global synchronization is required among peers as they can simply make use of their local system clocks with periodic synchronization to the Coordinated Universal Time (UTC) to achieve the intended purpose (with the correct offsets). Unless the time gap between concept drift occurrences are very close, otherwise some differences between the local clocks and UTC is acceptable and peers only need to synchronize with UTC once in a while.

We need to find not only temporal associations but also the *expected delays* between the occurrences of concept drift at different peers (that can be seen also as the propagation time of concept drift from one peer to another). We mine this knowledge from a sequence of global concept drift occurrence events. The setting for this mining is formally defined as follows [20].

Given a set of peers E, a concept drift event is a pair (A, t) where $A \in E$ corresponds to a peer whose concept has drifted at time t_i. A concept drift event sequence $\mathbf{s}$ is a triple (s, T_s, T_e) where $s = \langle (A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n) \rangle$ is an ordered list of concept drift events such that $t_i \leq t_{i+1}, \forall i \in \{1, \ldots, n\}$ and T_s and T_e are the starting and ending time where $T_s \leq t_i \leq T_e, \forall i \in \{1, \ldots, n\}$. Hence, from Fig. 1, we can derive the follow sequence $\mathbf{s}_{global} = (s_{global}, t1, t20)$ where $s_{global} = \langle (p_2, t1)$, $(p_1, t3)$, $(p_3, t4)$, $(p_2, t6)$, $(p_1, t8)$, $(p_2, t9)$, $(p_3, t10)$, $(p_1, t11)$, $(p_2, t15)$, $(p_1, t17)$, $(p_2, t18)$, $(p_3, t19)$, $(p_1, t20) \rangle$.

Our problem is closely related to the frequent episode mining problem [20], [21], [22]. Hence, from here on, we shall use the terms *episodes* and *temporal association rules* interchangeably. In general, an episode can be considered as a partially ordered collection of events occurring together [20]. Existing frequent episode mining algorithms aim to find all possible frequent episodes from the sequence. Our scenario is a bit different, as our peers are only interested in concept drift events leading to their own concept drifts. Hence, we simplify the problem by splitting the sequence into sub-sequences using the local concept drift event as the delimiter, i.e., setting $T_s + 1$ and T_e as time of two consecutive local drifts events. Table 1 shows the resultant sub-sequences by segmenting s_{global} where p_1 is the local peer, e.g., the set of sub-sequences is $\{(ss1, t1, t3)$, $(ss2, t4, t8)$, $(ss3, t9, t11)$, $(ss4, t12, t17)$, $(ss5, t18, t20)\}$.

In this setting we can simply apply frequent sequence mining algorithms [23] to find these frequent

TABLE 1
Peer $p1$'s segmented sequences of the global sequence of concept drift occurrences.

Tid	Segmented sequences
$ss1$	$(p_2, t1)$
$ss2$	$(p_3, t4), (p_2, t6)$
$ss3$	$(p_2, t9), (p_3, t10)$
$ss4$	$(p_2, t15)$
$ss5$	$(p_2, t18), (p_3, t19)$

episodes α denoted as a pair $(V, \leq, g)$ where V is a set of nodes, $\leq$ is a partial order on V and $g : V \rightarrow E$ is a mapping associating each node with a peer. The interpretation of an episode is that the events in $g(V)$ have to occur in the order described by $\leq$. Formally, an episode $\alpha = (V, \leq)$ occurs in an event sequence $\mathbf{s} = (s, T_s, T_e)$, denoted as $\alpha \in \mathbf{s}$, if there exists an injective mapping $h : V \rightarrow \{1, \ldots, n\}$ from nodes of α to events of $\mathbf{s}$ such that $g(i) = A_{h(i)}$ for all $i\ inV$, and for all $i, j \in V$ with $i \neq j$ and $i \leq j$ we have $t_{h(i)} < t_{h(j)}$. For ease of reading, we present the episodes in our text as $e = A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_n$ where $A_i \rightarrow A_j$ denotes concept drift event A_i occurs before A_j, e.g., ss2 in Table 1 can be represented as $p_3 \rightarrow p_2$.

When computing the *delay* from the occurrence of an episode until the local concept drift event, we are only interested in computing delay from the smallest sub-sequence with the earliest ending time which the given episode occurs in (*earliest occurring sub-sequence*). This is because the episodes are matched in a state transition manner. Hence, given an episode α and a sequence $\mathbf{s} = (s, T_s, T_e)$, *earliest occurring sequence* is denoted as $\mathbf{s}_{earliest} = (s_{earliest}, T_s, t_e)$ where $t_e = \min_{t \leq T_e} \{\mathbf{s} = (s, T_s, t) | \alpha \in \mathbf{s}\}$. Therefore, the *delay* is computed as $t_l - t_e$ where t_l is the time the local concept drift event occurred.

For example given a subsequence $s = \langle (p_2, t1), (p_3, t2), \quad (p_2, t4), (p_3, t5), \quad (p_1, t6)) \rangle$ and a frequent episode $e0 = p_2 \rightarrow p_3$ of peer p_1, $\langle (p_2, t1), (p_3, t2) \rangle$ is the first occurrence of $e0$ and the delay of an episode $delay(e0)$ is computed from the last event of the first occurrence of the episode to the local concept drift event, e.g., $delay(e0) = t6 - t2$.

In this setting, based on the definition of serial episode [22], all the mined frequent episodes will have hundred percent confidence, which is not useful at all. Hence, we deviate from the standard definition to define confidence as the same value as support σ since it gives the probability a given episode leading to a local concept drift.

In addition, we propose a new measure, delay deviation d_{dev} which is the standard deviation of the expected delay leading to the local concept drifts given a matched frequent episode, indicating the confidence we have on the expected delay. A lower d_{dev} implies

TABLE 2
Frequent episodes leading to concept drifts of peer p_1. For simplicity, the consequent p_1 is removed.

Rid	Freq. episodes	Delays	σ	d_{med}	d_{dev}
$e1$	p_2	2,2,2,2	1	2	0
$e2$	p_3	4,1,1	0.6	1	1.732
$e3$	$p_2 \rightarrow p_3$	1,1	0.4	1	0
$e3$	$p_3 \rightarrow p_2$	2	0.2	2	0

a higher confidence since the expected delay covers a smaller range and hence is more accurate.

For estimating the delay until the local concept drift event, we define the variable *expected delay* d_{med} computed as the median of the delays. Median is chosen to reduce the effects of outliers. Table 2 provides an illustration of the frequent episode, support, delays, expected delay d_{med} and delay deviation d_{dev}, with reference to Table 1.

Note that the frequent sequence mining does not compute the support, expected delay d_{med} and delay deviation d_{dev}. Hence, post processing is required. We use the fast counting technique presented in [24] to match the episodes and perform the statistics computations.

Finally, the filtering of the episodes, to reduce the number of episodes mined and prevent mining of irrelevant episodes, are as follows. Two input parameters, viz., min support σ_{min} and max delay deviation $d_{max-dev}$, are specified. The min support σ_{min} is used as the first cut filter when all episodes with support σ lower than σ_{min} are discarded. Then the max delay deviation $d_{max-dev}$ is used where rules with d_{dev} larger than $d_{max-dev}$ are discarded. These (unfiltered) frequent episodes, expected delay d_{med} and delay deviation d_{dev} are then used for the temporal association rule matching (cf. Section 3.6).

3.4 The Ensemble of Proactive Models (PEM)

This section provides the details of how the ensemble of proactive models **PEM** is managed and maintained (**Method 9**).

The classifier m_{r_i} of a remote peer r_i is considered as a proactive classifier if it is learnt at r_i after a concept drift has occurred at that peer and we expect that this drift will propagate in a near future to p. Whenever peer p detects a local concept drift, the following condition is checked for the classifier m_{r_i} of every remote peer r_i:

$$Err(m_{r_i}, \mathbf{D}_p^{t-1}) - Err(m_{r_i}, \mathbf{D}_p^{t}) > \epsilon_{cut} \qquad (3)$$

Since a significant difference have been found by the local classifier, then a proactive classifier of a remote peer r_i should also be able to do so and obtain a significantly higher error on the old concept compared to the new concept, thus satisfy (3). If m_{r_i} satisfies (3), peer r is flagged as a proactive peer and the accuracy $1 - Err(m_{r_i}, \mathbf{D}_p^t)$ of m_{r_i} is stored and used later if it is used as a proactive classifier.

3.5 The Ensemble of Reactive Models (REM)

In this section we detail how the ensemble of reactive models **REM** is managed and maintained (**Methods 10 & 11**).

The ensemble of reactive models for each peer p consists of one locally learnt model (i.e. from data observed by peer p) and zero or more models which were learnt remotely at other peers and validated on recent data at p.

Every peer p stores the latest classifier m_{r_i} of every remote peers r_i. Whenever a new classifier m_{r_i} arrives, the following conditions (two of which are based on `ADWIN`'s measure) are checked:

$$\begin{aligned} Err(m_{r_i}, \mathbf{D}_p^t) &< 0.5 && (4)\\ Err(m_{r_i}, \mathbf{D}_p^t) - Err(m_p, \mathbf{D}_p^t) &\leq \epsilon_{cut} && (5)\\ Err(m_{r_i}, \mathbf{D}_p^t) - Err(m_{r_i}, \mathbf{D}_p^{t-1}) &\leq \epsilon_{cut} && (6)\end{aligned}$$

where n_1 and n_2 of (2) are the size of the respective datasets. If all three conditions are satisfied, then m_{r_i} is a reactive model and it is added to the reactive ensemble and the preceding model removed, i.e., $\mathbf{REM}_p \leftarrow m_{r_i}$ and $\mathbf{REM}_p = \mathbf{REM}_p \setminus \hat{m}_{r_i}$, where $\hat{m}_{r_i}$ is the predecessor of m_{r_i}.

In addition to the criteria of adding a classifier to an adaptive ensemble when its error rate goes below a certain threshold, e.g. 0.5 in (4), we employ two more conditions specific to our distributed settings, which we expect to improve the accuracy of the ensemble. In (5), we verify that the error rate of m_{r_i} is not significantly higher than the local classifier m_p. By doing so, we are trying to ensure that m_{r_i} is built on data from the same concept as m_p. In (6), we verify that the error rate of m_{r_i} on the previous concept is not significantly higher than the current concept of peer p since this might indicate that m_{r_i} is built from previous concept data and its error rate might increase in the future.

Note that whenever local concept drift (LCD) is identified, every remote model m_{r_i} is verified and added to REM if it passes the validation. Recall Fig. 4, which provides an illustration the situations when the reactive models need to be used.

3.6 Balancing Reactive and Proactive Ensembles

This section specifies the details of how the Reactive and the Proactive ensembles interact with each other and details the role of the temporal association rules in this process (**Methods 12 & 13**).

Whenever the local peer is notified of a peer's concept drift, the previously mined temporal association rules will be checked. A match of peer's concept drift will transit the rule to the next state. Note that the initial state of all rules is the first item. For instance, given a temporal association rule $e4 = p_4 \rightarrow p_2$, its initial state will be p_4, meaning that it will wait for the occurrence of peer p_4's concept drift. Given that now p_4 concept drift has occurred, then the state will be transited to p_2.

When the entire rule is matched, all peers identified in the rule will be made into proactive candidates until d_{med} of the rule has passed (from the point the last peer's has concept drifted). Note that if a new rule with lower d_{dev} is matched, the d_{med} of all peers in the new rule are replaced with the new d_{med} as it has a higher confidence hence taking precedence.

Whenever the delay of a proactive candidate r_i has passed, its current classifier will be added to the proactive ensemble, i.e., $\mathbf{PEM} \leftarrow m_{r_i}$, and removed from the reactive ensemble, i.e., $\mathbf{REM} = \mathbf{REM} \setminus m_{r_i}$.

At any point in time for a given peer we need to balance between the reactive and proactive predictions depending on the expectations of concept drift. We use the process of adding models to **PEM** and removing models from **REM** as our mechanism for adaptively balancing the weights between reactive and proactive prediction. As more rules are matched, that indicates that more peers have experienced concept drift and it becomes more evident that the local concept of a given peer is likely to drift. As such, we need to increase the importance of the proactive prediction which is achieved by increasing in the number of models in **PEM** or reducing the size of **REM**. We expect this simple approach to be more accurate than assigning weights for the two ensembles. Finally, note that whenever local concept drift (LCD) is identified, all models in **PEM** are removed.

3.7 Prediction by Reactive and Proactive Ensemble Models

All the previous section of the PINE framework presentation detailed the process of online training and adaptation of the system. This final section of the framework describes how to get a prediction from a trained PINE for an unseen data point (**Method 2**).

When peer p receives a new unseen data $\mathbf{x}_{new}$, its class label y_{new} is predicted using all models from the reactive and proactive ensembles, i.e., $\forall i, m_i \in \mathbf{REM}_p \cup \mathbf{PEM}_p$, using a weighted voting approach. The weights of the models in $\mathbf{REM}_p$ and $\mathbf{PEM}_p$ are computed by different ways. In particular, the weights of model in $\mathbf{REM}_p$ are computed based on their error on the training data $\mathbf{D}_p$ of the current concept, i.e., $w_i = 1 - Err(m_i, \mathbf{D}_i), m_i \in \mathbf{REM}_p$. In contrast, for the models $m_i \in \mathbf{PEM}_p$, their weights are obtained during the proactive classifier verification or in another words it is estimated from the performance of the previous classifier on current concept, e.g., $w_i = 1 - Err(\hat{m}_i, \mathbf{D}_p)$, where $\hat{m}_i$ denotes the predecessor of m_i. Finally, the class label is predicted by the following function: $y_{new} = \text{sign}\big(\sum_i (w_i m_i(\mathbf{x}_{new}))\big)$.

We described a general overview and detailed design choices of our framework for distributed classification under asynchronous concept drifts. The framework makes use of associations between drifts in different peers happening over time to speed up the adaptation by predicting the occurrence of drifts. In the next section we empirically evaluate the performance of PINE.

4 Experimental Evaluation

We conduct extensive experiments to evaluate the performance of our proposed approach. We compare the performance of PINE with existing approaches that can handle drifting concepts in distributed network classification.

The main goals of our experimental study include:

- presenting the motivation and potential of proactive handling of concept drift in distributed settings, i.e. demonstrating that the changes in different peers can be related, and that corresponding temporal relations between peers (w.r.t. occurrence of drifts) can be mined effectively;
- demonstrating that we can estimate the propagation time of concept drift from one peer to another accurately enough such that this estimate can be used for proactive handling of concept drift;
- showing that the proposed PINE approach, improves the accuracy of RePCoDE – the state-of-the-art approach for handling concept drift in P2P (distributed) networks;
- demonstrating that PINE is insensitive to parameters;
- demonstrating that PINE has lower communication cost than existing approaches and acceptable computational costs.

4.1 Datasets

For our experiments we use both synthetic and real world dataset.

Synthetic data. We employ the *moving hyperplane* data generator [14] that acts as a benchmark for evaluating classifiers adapting to concept drift. With the hyperplane we simulate both the (incremental) gradual and sudden drifts scenarios. We choose this data model since it has several important properties for analyzing concept drift. First, it does not change the prior probabilities of classes; second, the drift is easy to quantify by the angle of rotation; third, the hyperplane (or plane in 2D) is easy to interpret.

Our moving hyperplane generator is as follows. A hyperplane in d-dimensional space is expressed by the equation $\sum_{i=1}^{d} w_i x_i = w_0$, where w_i is the weight of the attribute x_i. Data instances satisfying $\sum_{i=1}^{d} w_i x_i \geq w_0$ are labeled as positive and negative otherwise. Concept drifts are introduced by changing the weights of the attributes and the direction of change. The generated data consists of 8 attributes of which 2 are drifting. The probability of noise, which means assigning a wrong class label, is set at 0.05. The weight change per attribute is set to 0.5 and 5, the probability of direction change is set to 0.1 and 1 and concepts are set to drift every 10 and 500 instances, randomized by adding a small random value, for gradual and sudden drift scenarios respectively. Peers' initial concepts are the same and are set to drift to the next concept with different delays, randomly assigned in the range of $[0, 1000)$. We conducted 10 independent runs with every run consisting of 100 peers. Results were obtained by averaging over all peers and runs. The total number of instances each peer receives for one run are 17,000 and 19,000 for the gradual and sudden drift scenarios respectively.

Real data. To verify if peers share correlated concepts (and concept drifts), we conducted experiments on *real world weather* data from SOD[1], which consists of daily observations recorded in different meteorological stations. The task is to predict whether it is going to rain from 10 weather observations, viz., mean temperature, mean dew point, mean sea level pressure, mean station pressure, mean visibility, mean wind speed, maximum sustained wind speed, maximum wind gust, maximum temperature and minimum temperature. We selected 500 worldwide stations each representing a peer and observations from year 2001 to 2010 consisting of 3038 days of observations that exist in all selected stations.

4.2 Experimental setup

We compare our proposed `PINE` approach with three other approaches, first two of which can be considered as two baselines and the third one as the main existing competitor: 1) *Local* – uses only the latest local adaptive model (based on our modified `ADWIN`) of a peer to perform prediction. It represents existing centralized solutions that does not share any information. 2) *All* – uses the latest adaptive model of all peers to perform prediction. It represents both ensemble approaches and approaches that assume a global drifting distribution. 3) RePCoDE – an existing approach for handling concept drift in P2P networks and has already been shown to perform better than fixed window based approaches on the moving hyperplane dataset [4].

The δ value of `ADWIN` for all adaptive detection approaches is set as 0.25. The default *min support* σ and *max delay deviation* d_{dev} is set at 0.25 and 100 respectively. Parameters of RePCoDE are set as follows: window size 100, proactive and reactive ratio 0.5, number of nearest neighbor voters 10% of peers, number of future voters 10% of peers, propagation delay 4, concept drift threshold 0.05 and number of sequences to match 4.

We test the approaches in a prequential manner [25], i.e., the incoming data are first used for testing and then for updating classifiers. Results are computed as the moving average (100 data points) of averaged prequential error rate of all peers.

For implementation, all algorithms were implemented in C++. We adopt the LIBLINEAR [26] package for linear SVM as the base classifier for all approaches. Linear SVM is chosen due to its high efficiency and scalability for learning and model propagation in distributed networks.

4.3 Performance of the compared approaches

In this section we compare the four approaches on accuracy, communication and computational cost.

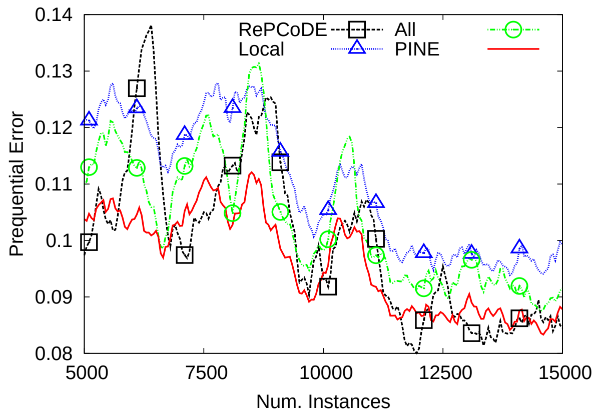
4.3.1 *Accuracy*

In Fig. 6 the accuracies of the approaches are compared. The increase in error rates indicates that concepts in peers start to drift and the decrease indicates that the new concepts have been learnt. For the gradual hyperplane dataset (Fig. 6(a)), we observe that the prequential error of all approaches decreases as time passes, especially for RePCoDE and `PINE` which become much better than Local and All, and PINE achieves the lowest error among all. This is because both RePCoDE and `PINE` learn from historical data and hence can reduce error as time passes. We also find that for all the concept drifts, i.e., when the error starts to rise, `PINE` can detect and adapt to the concept drift faster and thus achieves lower peaks in error throughout.

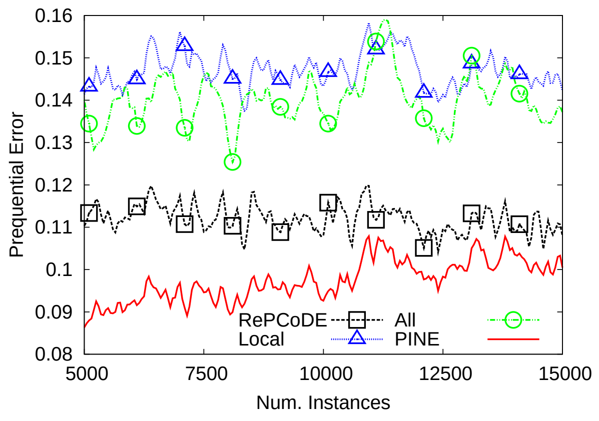
For the sudden hyperplane dataset (Fig. 6(b)), we can see that `PINE` consistently achieves lower prequential error than all other approaches followed by RePCoDE, then All and finally Local. This is most likely due to the fact that in this dataset, delay among the occurrences of peers' concept drift are fixed and the drifts are sudden. Hence `PINE` can consistently detect concept drifts and accurately learn the temporal association between peers' concept drift occurrences.

For the weather dataset (Fig. 6(c)), we observe that concept drifts consistently occurs around April every year, which corresponds to spring and autumn in the northern and southern hemisphere respectively. This could indicate that the patterns of rainfall change every year, instead of quarterly when seasons change. Also we found that initially `PINE` is comparable to RePCoDE, but as time passes, its prequential error becomes lower than that of RePCoDE. While not shown in the figures, note that `PINE` has higher prequential error in all datasets initially. However, the prequential error quickly reduces to become lower

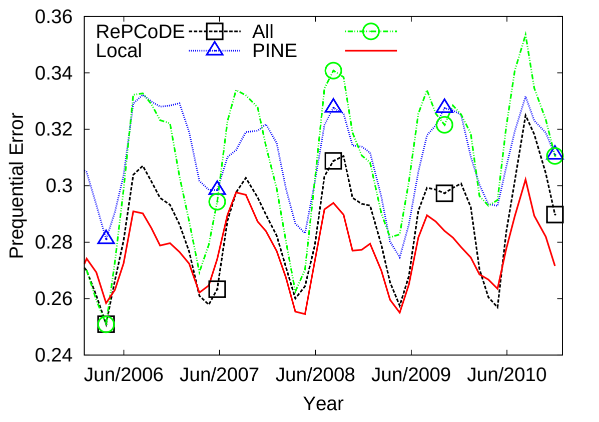
1. http://www.ncdc.noaa.gov/cgi-bin/res40.pl?page=gsod.html



(a) Gradual Hyperplane



(b) Sudden Hyperplane



(c) Weather

Fig. 6. Moving average of prequential error rate over 100 data points.

than all approaches over time. This is because PINE learns associations from the history of concept drift occurrences and hence several occurrences are needed to learn the pattern of the drifts before it can perform well.

Given that RePCoDE performs better than Local and All in most datasets, we can see that prediction of the future concepts does indeed reduce prequential errors even though RePCoDE uses the fixed window drift detection which is in general slower in concept drift detection. Moreover, PINE achieving the lowest prequential error demonstrates that in addition to predicting future concepts, predicting when concept drifts occurs can further reduce the prequential error.

TABLE 3
Average communication cost per peer in terms of kilobyte (models propagated).

Approach	Gradual	Sudden	Weather
RePCoDE	634.80 (72.87)	820.99 (94.24)	654.30 (12.61)
PINE	293.51 (33.69)	609.13 (69.92)	616.73 (11.88)

TABLE 4
Average time taken (in msecs).

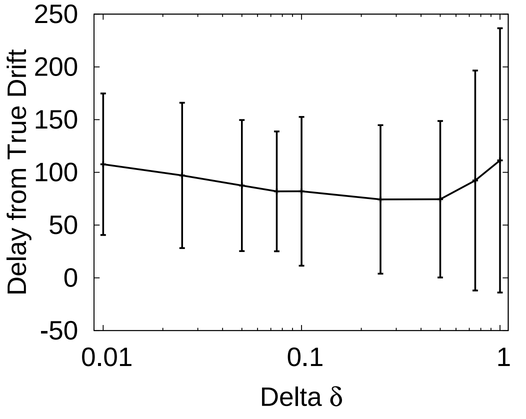
Data	RePCoDE	Local	All	PINE
Model Construction				
Gradual	0.227	2.813	2.813	2.813
Sudden	0.252	1.311	1.311	1.311
Weather	131.3	509.2	509.2	509.2
Pre-prediction				
Gradual	0.370	-	-	37.32
Sudden	0.394	-	-	178.0
Weather	11.48	-	-	71.56
Prediction				
Gradual	0.003	0.0003	0.018	0.009
Sudden	0.004	0.0004	0.019	0.010
Weather	0.010	0.0004	0.093	0.034

4.3.2 *Communication and computational costs*

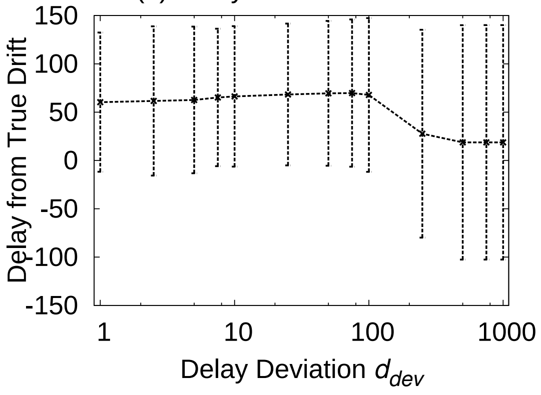
Tables 3 and 4 present communication and computational costs of the compared approaches.

The main communication cost incurred by the approach are the sending of models which is presented in Table 3. Note that Local does not incur any communication cost and the additional cost incurred by PINE compared to *All* is the cost needed to notify the time of drift. Hence, we only present the results of RePCoDE and PINE. Table 3 shows that PINE incurs less communication cost compared to RePCoDE. The difference is most likely due to the adaptive detection and propagation of models compared to the fixed window drift detection and fixed delay model propagation.

Table 4 presents the average time taken by each peer to perform various tasks. Model construction refers to the time taken to build a classifier. Pre-prediction refers to the computations required before the prediction, i.e., indexing, model retrieval and sequence similarity matching for RePCoDE and temporal association mining, temporal association rule matching and model verifications for PINE. Prediction refers to the time taken to predict on one unlabeled data point. Even though PINE incurs the most computational cost compared to other approaches, it is still acceptable as



(a) Delay in detection



(b) Delay in prediction

Fig. 9. Effects of delta δ (ADWIN) and delay deviation d_{dev} on delay from true drift.

the absolute time taken is very low, i.e., all tasks can be computed in less than one second.

4.4 PINE sensitivity to the parameters

To demonstrate that the error rate of PINE is not much affected by the choice of parameters, we tested different parameter values and present the results in Fig. 7 and 8. In addition, we show in Fig. 9 how delta δ and delay deviation d_{dev} affect the delay in detection and prediction respectively on the sudden hyperplane dataset, which is only possible as ground truth of the concept drifts is known.

Fig. 7 shows the prequential error of varying delta (ADWIN) from 0.01 to 1. For the gradual hyperplane dataset (Fig. 7(a)), observe that as delta increases, the prequential error decreases. This is expected as unlike the sudden hyperplane dataset where there is a gap in the concept drift occurrences, the concept of the gradual hyperplane dataset is constantly drifting and hence decreasing delta provides no benefits at all whereas increasing delta can speed up the drift detection thereby reducing the prequential errors.

For the sudden hyperplane dataset (Fig. 7(b)), the prequential error initially drops when delta increases and then raises when it is greater than 0.25. This is expected as a lower delta value makes the drift detection more stringent but can also cause late detection of concept drift resulting in higher prequential error when it is too stringent. On the other hand, while a larger delta value relaxes the drift detection criteria and speeds up the detection, it can also introduce more false positives resulting in higher prequential error.

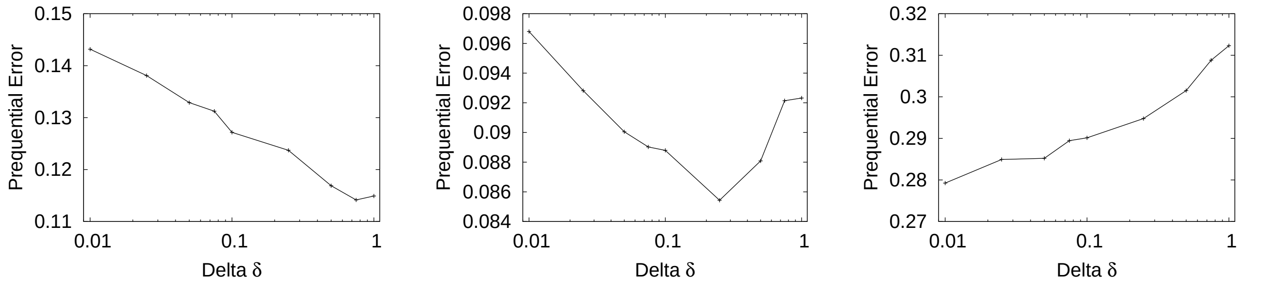
For the weather dataset (Fig. 7(c)), observe that as delta increases, prequential error also increases. Since we do not have any ground truth about its concept drifts, we can only draw conclusion from the results. It would seem that the concept of the weather dataset fluctuates a lot and hence requires a very small delta to detect an *actual* concept drift. Otherwise, it would be very easy to trigger false positives and adversely affect prediction accuracy. Further examination of the experiments does shows that the weather dataset triggers a larger number of concept drift detection as compared to the hyperplane datasets.

Note that the maximum difference in maximum and minimum prequential error for all the dataset is at most 3% over the entire range of delta values from 0.01 to 1. This indicates that PINE is not sensitive to the value of delta.

Fig. 9(a) illustrates the delay after the true concept drift has occurred until it is being detected. Observe that a large delta value increases the delay and its deviation for detecting concept drift. Similar to the prequential error, we found that the delay only varies a little given different delta values. Nevertheless, a good delta value should be between 0.1 and 0.5 observed from Fig. 7.

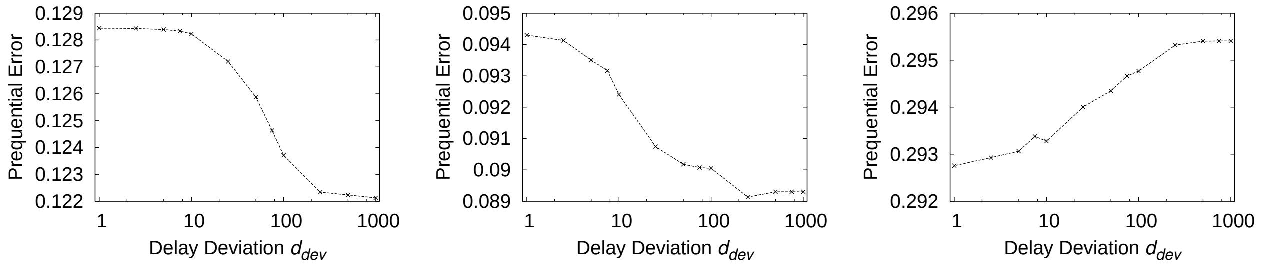
Fig. 8 shows the prequential error of varying the max delay deviation d_{dev} from the range 1 to 1000. Observe from Fig. 8(a) and 8(b) that as d_{dev} increases, the prequential error decreases. It is intuitive that increasing the d_{dev} allows more rules with higher delay deviations to be mined as demonstrated in Fig. 9(b) which should result in higher prequential error. However, as PINE replaces the predicted time of concept drift of a matched rule with that of another rule with lower d_{dev}, it can effectively prevent the increase in error when the d_{dev} is set too high.

However, Fig. 8(c) shows the opposite result, whereby as d_{dev} increases, the prequential error also increases. One possible explanation is that a large number of false positive concept drifts are trigger and this causes the rules to be mined and matched wrongly. Hence resulting in the increase in prequential error as d_{dev} increases. Therefore, it may be beneficial to set the value of d_{dev} only after delta is set and to adjust it accordingly. However, it is recommended to set a large d_{dev} value in most cases, with the drawback being an increase in time required to mine the rules and perform rule matching. In addition, note that the maximum difference between the maximum and minimum prequential error for d_{dev} values ranging



(a) Gradual Hyperplane (b) Sudden Hyperplane (c) Weather

Fig. 7. Effects of delta δ (ADWIN) on error rate.



(a) Gradual Hyperplane (b) Sudden Hyperplane (c) Weather

Fig. 8. Effects of delay deviation d_{dev} on error rate.

from 1 to 1000 is only 0.7%. This indicates that PINE is not sensitive to the value of d_{dev}.

Although the results of varying support are not presented, we still provide a discussion on its results. The prequential error obtained by varying support the range 0.01 to 1 is stagnant except until 0.75 where it drops slightly and increases by 0.5% upon reaching 1. The increase in support filters out temporal association rules that are not significant but when set too high it also removes useful rules, hence resulting in the increase in error when support is set to 1 as no rules are mined. Hence, it is easy to derive that any low value of support should be sufficient to maintain low prequential error, but a trade off would be an increase in time required to mine the rules and perform rule matching.

From the spectrum of variations analyzed, we can conclude that PINE does not rely on parameters heavily. It is exploitation of the links between peers that makes PINE perform better than the other approaches.

5 Related Work

Existing works on classification in non stationary distributed environments studies only the simplified problem settings. Bhaduri *et al.* [2] propose a decision tree for P2P networks that can adapt to a sudden change happening in all peers simultaneously. Such scenario, as our weather data experiments suggest, is too simple to be realistic. Chen *et al.* [27] explore web mining from multiple distributed data streams, where each distributed site learns a local Bayesian Network (BN) model, but it needs to send a subset of the relevant observations to a centralized site, which is often not available in many distributed environments. In our settings observations must not be transmitted over the network.

A number concept drift handling techniques have been developed in the last decade, we overview only the most relevant studies. A number of techniques employ only a single model, where the previous model (or a part of it) is regularly replaced with a new one [2], [8], [14]. Single model approaches are not suitable for distributed environments due to the nature of their centralized settings. In contrast, ensemble techniques keep many individual models [6], [10], [13], [17], [18], the system adapts via combination rules or model selection. Typically only incremental updates are done, the ensemble is not replaced at every time step or as soon as a change happens. Distributed network forms a natural ensemble, where each peer can contribute to one or more individual models.

An individual model update can be organized in an evolving manner or using detectors. The evolving techniques do not care whether a change has happened or not. In contrast, detectors aim to detect changes and then drop the old data. Evolving ensembles are among the most popular techniques for handling concept drift [6], [9], [10], [13], [15], [17]. Evolving techniques do not update their individual models, but they can discard and replace them. In distributed settings each local model represents a physical peer, thus we cannot throw away local models and build new ones. Thus evolving ensembles are not of direct relevance to our work. Hence, in our settings it is natural to use detectors for managing training sets at the local model level.

Approaches with detectors monitor and describe data over time, adaptive actions are taken based on warning signals. Changes can be detected while monitoring the incoming data distribution [28], [29] or performance indicators, for instance, the streaming error [30]. Change detection is a widely researched field on its own [31], in addition, a number of works are dedicated to change detection in concept drift context [11], [30], [32], [33], [34], [35]. These approaches handle concept drift using only a single centralized model, whereas distributed setting requires a distributed approach. Nevertheless, change detection techniques are important for efficient detection of the concept changes. Hence, we integrate ADWIN change detection [34] into our propose framework.

All the above approaches are *reactive*, time needs to pass before a change is detected and they start to adapt. Several *proactive* solutions exist that aim to predict a change before it happens [4], [36]. RePro [36] incorporates change predictions into a reactive ensemble. The prediction using a Markov chain is purely based on historical data, it does not explore relations between local models. In our present study we do not forecast changes, as by definition they happen unexpectedly. Instead we speed up the detection process, using early warnings, issued by other peers in the network.

RePCoDE [4] performs sequence matching to find peers with similar historical change patterns and use this information to manage the weights of the local models in an ensemble over time. The approach has several limitations. It detects concept drift using fixed windows which is often slower compared to adaptive techniques [19]. In addition, RePCoDE fails to fully exploit the association knowledge of the peers; it performs only pairwise sequence matching which can result in less accurate results as compared to mining from multiple associations. Finally, the performance of RePCoDE is sensitive to optimizing a number of parameters.

Classifier ensembles that are designed for handling reoccurring contexts [37], [38], [39] resemble PINE as they maintain local models, that are explicitly responsible for particular regions of the instance space. However, they do not adapt online and they do not use mechanisms to update the local models, as relevant for distributed problem.

To sum up, the existing solutions are unsuitable or insufficient for deploying in distributed environments, as they are either based on centralized settings or ignore the relationships between peers.

6 CONCLUSION

In this paper we proposed a novel framework PINE for handling asynchronous concept drift in classification in distributed networks. Our framework is an ensemble learning approach that combines reactive adaptation via drift detection, and proactive handling of upcoming changes via early warning and adaptation across the peers. Extensive experiments on synthetic and real world datasets illustrate that PINE adapts well to both gradual and sudden concept drifts and outperforms the state-of-the-art approaches.

REFERENCES

[1] P. Luo, H. Xiong, K. Lü, and Z. Shi, "Distributed classification in peer-to-peer networks," in *KDD*, 2007, pp. 968–976.

[2] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta, "Distributed decision-tree induction in peer-to-peer systems," *Statistical Analysis and Data Mining*, vol. 1, no. 2, pp. 85–103, 2008.

[3] H. H. Ang, V. Gopalkrishnan, W. K. Ng, and S. C. H. Hoi, "Communication-efficient classification in P2P networks," in *ECML/PKDD (1)*, 2009, pp. 83–98.

[4] ——, "On classifying drifting concepts in p2p networks," in *ECML/PKDD (1)*, 2010, pp. 24–39.

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

[6] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavalda, "New ensemble methods for evolving data streams," in *KDD*, 2009, pp. 139–148.

[7] S. Bach and M. Maloof, "A bayesian approach to concept drift," in *NIPS*, 2010, pp. 127–135.

[8] E. Ikonomovska, J. Gama, and S. Dzeroski, "Learning model trees from evolving data streams," *Data Mining and Knowledge Discovery*, vol. 23, pp. 128–168, 2011.

[9] M. Masud, Q. Chen, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham, "Addressing concept-evolution in concept-drifting data streams," in *ICDM*, 2010.

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

[11] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.

[12] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weighting," *Intelligent Data Analysis*, vol. 8, pp. 281–300, 2004.

[13] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007.

[14] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *KDD*, 2001, pp. 97–106.

[15] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *KDD*, 2003, pp. 226–235.

[16] I. Zliobaite, "Learning under concept drift: an overview," Vilnius University, Tech. Rep., 2009.

[17] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *KDD*, 2001, pp. 377–382.

[18] S. H. Bach and M. A. Maloof, "Paired learners for concept drift," in *ICDM*, 2008, pp. 23–32.

[19] A. Bifet and R. Gavalda, "Adaptive learning from evolving data streams," in *IDA*, 2009, pp. 249–260.

[20] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 259–289, 1997.

[21] S. K. Harms and J. S. Deogun, "Sequential association rule mining with time lags," *Journal of Intelligent Information Systems*, vol. 22, no. 1, pp. 7–22, 2004.

[22] T.-Y. Lee, E. T. Wang, and A. L. P. Chen, "Mining serial episode rules with time lags over multiple data streams," in *DaWaK*, 2008, pp. 227–240.

[23] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Machine Learning*, vol. 42, no. 1/2, pp. 31–60, 2001.

[24] S. Laxman, P. S. Sastry, and K. P. Unnikrishnan, "A fast algorithm for finding frequent episodes in event streams," in *KDD*, 2007, pp. 410–419.

[25] J. Gama, R. Sebastiao, and P. P. Rodrigues, "Issues in evaluation of stream learning algorithms," in *KDD*, ser. KDD '09. ACM, 2009, pp. 329–338.

[26] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear svm," in *ICML*, 2008, pp. 408–415.

[27] R. Chen, K. Sivakumar, and H. Kargupta, "Distributed web mining using bayesian networks from multiple data streams," in *ICDM*, 2001, pp. 75–82.

[28] X. Song, M. Wu, C. Jermaine, and S. Ranka, "Statistical change detection for multidimensional data," in *KDD*, 2007, pp. 667–676.

[29] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *VLDB*, 2004, pp. 180–191.

[30] J. Gama, P. Medas, G. Castillo, and P. P. Rodrigues, "Learning with drift detection," in *SBIA*, 2004, pp. 286–295.

[31] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes - Theory and Application*. Prentice-Hall, Inc., 1993.

[32] M. Leeuwen and A. Siebes, "Streamkrimp: Detecting change in data streams," in *ECML/PKDD*, 2008, pp. 672–687.

[33] K. Nishida and K. Yamauchi, "Detecting concept drift using statistical testing," in *Discovery Science*, 2007, pp. 264–269.

[34] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *SDM*, 2007.

[35] R. Klinkenberg and I. Renz, "Adaptive information filtering: Learning in the presence of concept drifts," in *Learning for Text Categorization*, 1998, pp. 33–40.

[36] Y. Yang, X. Wu, and X. Zhu, "Mining in anticipation for concept change: Proactive-reactive prediction in data streams," *Data Mining and Knowledge Discovery*, vol. 13, no. 3, pp. 261–289, 2006.

[37] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Tracking recurring contexts using ensemble classifiers: an application to email filtering," *Knowledge and Information Systems*, vol. 22, pp. 371–391, 2010.

[38] J. B. Gomes, E. Menasalvas, and P. A. C. Sousa, "Tracking recurrent concepts using context," in *RSCTC*, 2010, pp. 168–177.

[39] I. Zliobaite, J. Bakker, and M. Pechenizkiy, "Towards context aware food sales prediction," in *ICDM Workshops*, 2009, pp. 94–99.

Hock Hee Ang is a PhD student in the School of Computer Engineering at Nanyang Technological University in Singapore. His research interests include distributed data mining and ensemble learning.

Vivekanand Gopalkrishnan is the director of research for the Deloitte Analytics Institute. He has nearly 20 years of experience in research, teaching, consulting on the development of practical solutions to real-world issues using analytics.

Vivek has advised clients on strategies for driving data-driven insights, and specialises in architecting innovative solutions that mine insights even when the data is not well-behaved. His research covers data mining, machine learning and data warehousing, and he has published over 50 papers in these fields.

Vivek continues to actively serve the academic and research communities. He is on the editorial boards and review committees of leading research journals, and on the program committees of top international data mining and information management conferences. As a passionate educator, he continues to guide university academic programmes and research councils in analytics.

Indrė Žliobaitė is a Lecturer in Computational Intelligence at Bournemouth University UK. She received her PhD from Vilnius University, Lithuania in 2010. I. Žliobaitė has six years of experience in credit analysis in banking industry. Her research interests concentrate around online data mining, including learning from evolving streaming data, change detection, adaptive and context-aware learning, predictive analytics applications. Recently she has co-chaired workshops at ECMLPKDD 2010 and ICDM 2011, co-organized tutorials at CBMS 2010 and PAKDD 2011 on adaptive learning. She is a Research Task Leader within the INFER.eu project that is developing evolving and robust predictive systems. For further information see http://zliobaite.googlepages.com.

I. Žliobaitė's research has received partial funding from the European Commission within the Marie Curie Industry and Academia Partnerships & Pathways (IAPP) programme under grant agreement no 251617.

Mykola Pechenizkiy is Assistant Professor at the Department of Computer Science, Eindhoven University of Technology, the Netherlands. He received his PhD in Computer Science and Information Systems from the University of Jyvaskyla, Finland in 2005. He has broad expertise and research interests in data mining and data-driven intelligence, and its application to various (adaptive) information systems serving industry, commerce, medicine and education. He has coauthored over 70 publications and has been organizing several workshops (HaCDAIS@ECML/PKDD2010, LEMEDS@AIME2011), conferences (IEEE CBMS 2012, EDM 2011, IEEE CBMS 2008, BNAIC 2009) and tutorials (at ECML/PKDD 2012, PAKDD 2011, IEEE CBMS 2010, ECML/PKDD 2010) in these areas. Recently, he has co-edited the Handbook of Educational Data Mining and served as a guest editor of the special issues in SIGKDD Explorations, Elsevier's DKE and AIIM, and Springer's Evolving Systems journals. Currently, he takes a leading role in NWO HaCDAIS, STW CAPA, EIT ICT Labs Stress@Work and NL Agency CoDaK projects information on which can be found at http://www.win.tue.nl/ mpechen/

Steven C. H. Hoi is an Assistant Professor of the School of Computer Engineering at Nanyang Technological University, Singapore. He received his Bachelor degree from Tsinghua University, P.R. China, in 2002, and his Ph.D degree in computer science and engineering from The Chinese University of Hong Kong, in 2006. His research interests are machine learning and data mining and their applications to multimedia information retrieval (image and video retrieval), social media and web mining, and computational finance, etc. He has published over 100 referred papers in top conferences and journals in related areas. He has served as general co-chair for ACM SIGMM Workshops on Social Media (WSM'09, WSM'10, WSM'11), program co-chair for the fourth Asian Conference on Machine Learning (ACML'12), book editor for "Social Media Modeling and Computing", guest editor for ACM Transactions on Intelligent Systems and Technology (ACM TIST), technical PC member for many international conferences, and external reviewer for many top journals and worldwide funding agencies, including NSF in US and RGC in Hong Kong. He is a member of IEEE and ACM.