Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

HDSM: A distributed data mining approach to classifying vertically distributed data streams[☆]

Benjamin Denham [*], Russel Pears, M. Asif Naeem

School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland, New Zealand

ARTICLE INFO

Article history:
Received 21 December 2018
Received in revised form 8 October 2019
Accepted 9 October 2019
Available online 14 October 2019

Keywords:
Distributed data stream mining
Vertically-distributed data
Online classification

ABSTRACT

The rise in the Internet of Things (IoT) and other sensor networks has created many vertically-distributed and high-velocity data streams that require specialized algorithms for true distributed data mining. This paper proposes a novel Hierarchical Distributed Stream Miner (HDSM) that learns relationships between the features of separate data streams with minimal data transmission to central locations. Experimental evaluation demonstrates significant improvements in classification accuracy over previously proposed distributed stream-mining approaches while minimizing data transmission and computational costs. HDSM's potential for dynamically trading off accuracy with computational resource costs is also demonstrated.

1. Introduction

The ever-growing number of "big data" sources presents many opportunities for applying data mining to discover patterns across large and diverse collections of data. However, larger datasets also present many challenges that require the development of new, specialized data mining methodologies in order to perform effective analysis.

Typical data mining algorithms assume the entire dataset under analysis is in one physical location, but this is not always the case. Some datasets are so large they must be distributed for efficient processing, while others are inherently distributed geographically across autonomous sites. Even if data-centralization is feasible, it would still be wasteful of network resources to unnecessarily transmit data that can be mined locally. Research into distributed data mining (DDM) has sought to address these issues by devising methods of combining distributed "local" data mining models into "global" models while minimizing data transmission cost.

Typical data mining algorithms also require a representative sample of a dataset to produce a model that describes the patterns in the data. However, the underlying patterns in data streams may drift over time (so-called "concept drift"), requiring continual adaptation of models. Online learning algorithms have been developed to perform data mining tasks on continuous streams of data, where each record can only be processed once and the model must be able to adapt to changing patterns in the data. While both distributed data mining and data stream mining have been extensively researched, there has been less work on approaches to mining distributed data streams, and it is considered an open research problem.

Given the proliferation of quality sensors at relatively low cost, a resurgence in distributed mining, particularly in mining heterogeneous data sources, is evident. Such applications that can benefit from distributed data stream mining techniques include analyzing data streams originating from many different mobile devices [1], addressing environmental issues by analyzing heterogeneous streams of traffic, wind, and weather data [2], topic modeling of resource requests for adaptive prefetching in distributed networks [3], and monitoring networks of streaming sensors in manufacturing plants for quality assurance [4]. This research has spawned a new set of approaches that feature distributed architectures, just as HDSM does. Some of these approaches include the autonomous construction of models at local sites, but they often require excessive communication of each record to a central site in order to perform classification [5–7]. In contrast, our proposed HDSM approach minimizes the data synchronization required for each record and hence is able to minimize communication overhead, thus making it more suitable for mining distributed heterogeneous streams of data.

This paper proposes a Hierarchical Distributed Stream Miner (HDSM) produced by extending the distributed data mining approach of Park et al. [8] for use in a distributed data stream mining context. The contributions we make in this research are:

[☆] No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.knosys.2019.105114.

[*] Corresponding author.
E-mail addresses: ben@denham.nz (B. Denham), russel.pears@aut.ac.nz (R. Pears), mnaeem@aut.ac.nz (M.A. Naeem).

https://doi.org/10.1016/j.knosys.2019.105114
0950-7051/© 2019 Elsevier B.V. All rights reserved.

- A novel architecture that supports autonomous creation of local classification models at each source site while supporting stream mining.
- We augment local classification capability by automatically building classifiers that learn and adapt to changes in relationships between data fragments dispersed at different source sites.
- We define a data transmission protocol that minimizes data transmission volume while providing superior accuracy to previously proposed distributed mining approaches.
- We demonstrate the architecture's potential to dynamically trade off accuracy with computational resource time.

The rest of the paper is structured as follows. Section 2 discusses previous work in the area of distributed data mining. Section 3 presents the architecture of HDSM for distributed data stream mining. The experimental results that demonstrate the performance and benefits of HDSM are presented next in Section 4. We conclude the paper in Section 5, which reviews some of the insights gained from the research and provides pointers for future research.

2. Related work

The age of big data has brought with it a need for machine learning techniques that can be applied not only to larger static datasets, but also to high-velocity and ever-changing data streams. Such applications are often motivated by requirements for real-time analysis and decision making, or single-pass analysis of streams that are too large to store for later analysis. Furthermore, stream mining methods must be able to account for changes caused by concept drift. Algorithms have been developed to tackle many different classes of machine learning problems in data stream environments, including utility and erasable pattern mining [9,10], regression [11], and classification. Data stream classification is the focus of this paper, and as such we make use of the state-of-the-art Adaptive Random Forest stream classifier [12] and the Hoeffding Tree [13] that is used as the base classifier for its ensemble. Ensembles of classifiers are a powerful technique used in recent stream mining advances [14–16], and they also play an important role in the HDSM architecture presented in this paper. However, some streams involve such large volumes of data that they cannot be processed by a single machine, and instead require algorithms that utilize concurrent processing techniques to process more records in less time.

Big data stream environments that necessitate concurrent data processing generally arise from one of two situations: either there is a single, high-throughput stream of data, or there is a large number of streams that together result in a large volume of data. The former class of problems involving a single stream can be addressed with parallelized data mining techniques, which partition data for parallel processing across multiple cores or machines. However, these approaches often depend on the assumption that data originates in a single stream, and are therefore not always appropriate for distributed data stream mining. For example, the approach of Tennant et al. [17] for classification based on nearest-neighbors and micro-clustering evenly partitions records across multiple sites for model training, but unlabeled records must be broadcast to all sites for classification. Furthermore, while the Vertical Hoeffding Tree (VHT) proposed by Kourtellis et al. [5] parallelizes tree construction by distributing the values for different data features to different sites, the centralized classifier it produces still requires that all features of each record to be classified are available at a single location. Even some approaches designed for distributed environments suffer from this problem because they combine models trained at different sites on local streams to produce centralized classifiers [6,18]. These approaches are appropriate for the use cases of producing a model of relationships within distributed data [18] or distributing already centralized data for parallel learning [5], but not for performing online classification with distributed streams. Because they can result in an unacceptable level of data transmission for distributed online classification, such approaches are not discussed further in this paper.

Online distributed classification techniques assume that the target dataset is distributed across different physical *sites*, and that a global model of the data must be produced and used to classify records with as little data transmission from the individual sites as possible. According to Park et al. [8], data is typically distributed in one of two ways: Either each site describes different data records according to the same set of features (known as horizontal or homogeneous distribution), or each site describes the same records, but according to a different set of features at each site (known as vertical or heterogeneous distribution). With vertical distribution, each site essentially has a different view of the same set of records, and it is generally assumed that all sites share at least one "unique key" feature that can be used to join vertical fragments of records from different sites. Recent approaches to vertically distributed data mining have focused on addressing the heterogeneous nature of such data sources [19,20], and preserving privacy by controlling how data is shared between sites [21]. However, these approaches do not prioritize the reduction of data communication costs, which is the priority of our HDSM architecture. While approaches for performing various data mining tasks in a vertically distributed environment have been developed (such as frequent itemset [22] and association rule [21] mining), vertically distributed classification is the focus of this paper.

Liu et al. [23] present a method for vertically distributed, online, and semi-supervised classification, but their method is specific to a support vector machine classifier. Similarly, Kholod et al. [24] describe a method for parallelizing a Naive Bayes classifier on either horizontally or vertically distributed data. A more general approach to vertically distributed data mining is to apply ensemble learning techniques to aggregate many local classifications, which can be produced by any kind of classifier. Distributed ensemble methods can be characterized as having a two-layer architecture, where local classifiers are applied in the first layer, and local classifications are aggregated in the second. Because ensemble learning only requires the centralization of local classifications and not data features, it can also be used for data stream classification. The key difference between various ensemble learning techniques is in how they aggregate the local classifications.

Several types of ensemble learning approaches have been used to aggregate results from local classifiers in the distributed (though not necessarily streaming) context. Aggregation can be based on local classification confidence, such as selecting the classification with the maximum confidence [8], selecting a classification based on other order statistics [25], or treating local classification confidences as probabilities that can be used to approximate a global posterior probability [26]. An approach used by Skillicorn and McConnell [27] is to select the classification that receives the majority vote of local sites, optionally weighting votes by test accuracy (or potentially, the recently observed accuracy in a stream mining context). Alternatively, a "stacked" classifier can be trained to predict the true classification based on local classifications. Parker et al. [28] use a hierarchy of stacked classifiers to classify vertically distributed data streams with the ability to learn new classes as they appear over time. Yi et al. [2] also present a weighted-merge aggregation method, though this is designed for use with deep neural networks. However, not all methods of ensemble aggregation are suitable for fully online stream learning. For example, those used by Recamonde-Mendoza and Bazzan [29] require batches of records to be ranked according to local confidence before applying social choice functions to merge sets of local rankings.

More recent approaches to classifying vertically distributed, non-streaming data either produce a "centralized classifier" [7, 30–32] or are privacy-preserving methods that require multiple rounds of communication between sites [33]. Therefore, these methods are not suitable for a data stream environment. The most relevant approaches for comparison are ensembles of local classifiers with aggregation based on maximum confidence [8], voting [27], and stacking [28]. The method of Park et al. [8] that centralizes a small subset of the data is also suitable for comparison, as it forms the basis for the approach of HDSM (as described in the following section).

2.1. Foundational distributed data mining approach

The previous work of most relevance to this paper is the work of Park et al. [8], which forms the foundation for the architecture of HDSM, presented in the next section.

In their approach, each local or "primary" site constructs a classifier from its data and identifies a subset of "trouble records" that were classified with a confidence value below a certain "confidence-threshold". If a particular record appears in the trouble records of all primary sites, then all fragments of the record are transmitted to a central "trouble site" to be used to train an additional classifier. The trouble site's classifier captures relationships (hereafter referred to as cross-terms) that exist within the data: patterns that can only be discovered by viewing the data from the combined perspective of two or more sites. To classify new records, the models at each primary site and the trouble site are applied, and the classification results are transmitted to a central "aggregator". The classification result that achieved the highest confidence is then selected as the final classification. Experimentation showed that as the confidence-threshold was increased to allow more data to be sent to the trouble site, the overall classification accuracy also increased as a result.

There are some limitations of the approach of Park et al. that are addressed in the HDSM architecture proposed by this paper. Firstly, having only a single, central trouble site to collect records from all primary sites may not be necessary if cross-terms exist between the features of only a subset of primary sites. While the problem of data stream mining was not specifically addressed, the approach is generally amenable for use in classifying data streams. However, it is not equipped to control data transmission volume in a dynamic stream environment, as is demonstrated in the sections below.

3. Proposed distributed stream mining architecture

The following section describes the proposed HDSM architecture for performing classification on multiple, vertically-distributed data streams without joining all record fragments at a central location. HDSM is based on the previously described approach of Park et al. but contains three main extensions to make it applicable to a distributed data stream mining context:

1. The single trouble site is replaced with a flexible hierarchy of many trouble sites that process different combinations of feature subsets belonging to records from primary sites or other trouble sites.
2. The static confidence-threshold for selecting trouble records is replaced with a mechanism for dynamically regulating the volume of data received by each trouble site.

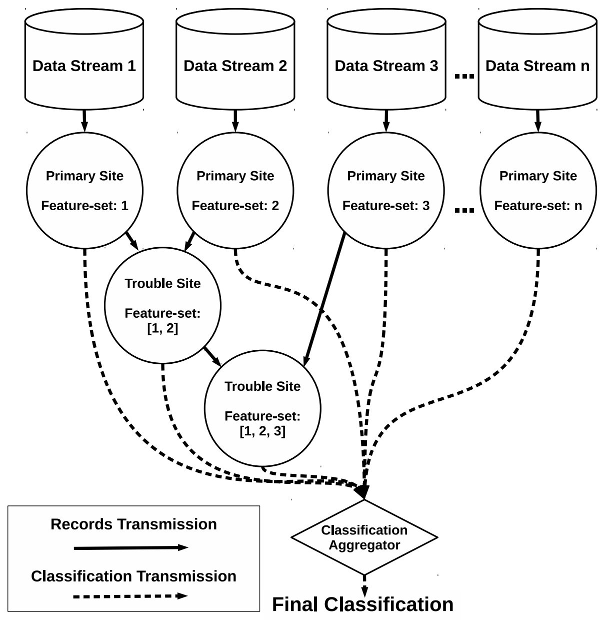
Data Stream 1
Data Stream 2
Data Stream 3
...
Data Stream n
Primary Site Feature-set: 1
Primary Site Feature-set: 2
Primary Site Feature-set: 3
Primary Site Feature-set: n
Trouble Site Feature-set: [1, 2]
Trouble Site Feature-set: [1, 2, 3]
Classification Aggregator
Final Classification
Records Transmission
Classification Transmission

Fig. 1. Physical view of HDSM.

3. An algorithm is described for learning a suitable hierarchy of trouble sites from the incoming data streams, and adapting the hierarchy in the case of concept drift.

Each of these extensions is described in detail throughout the rest of this section.

3.1. Trouble site hierarchies

A key limitation of Park et al.'s approach is that there is only one trouble site. This may result in a large amount of unnecessary data transmission when cross-terms only exist between the features across some subset of primary sites. Therefore, HDSM allows for multiple trouble sites, each of which may receive trouble records from only a subset of two or more primary or trouble sites. A trouble site that considers only a subset of features may need to escalate to a higher-level trouble site that considers more features. Fig. 1 presents a physical view of the HDSM architecture, which is composed of multiple primary sites and trouble sites, and a single classification aggregator. The behaviors of each component are described in Algorithm 1.

Each primary site receives input from a different heterogeneous data stream. The feature set of each stream may contain a different number of features, and there could be overlap between feature sets. It is assumed that each feature set will contain a common "key" feature that is not used for classification, but allows record fragments and classifications to be merged at trouble sites and the classification aggregator respectively. Because each primary site only needs to process data from a single stream, it is logical for each primary site to be physically located at the source of its data stream. Procedure `processRecord` shows how each record that arrives at a primary site will be classified by a local stream classifier. It is possible to use any stream classification algorithm at each site, or even different algorithms at different sites, as long it is possible to produce a confidence value on a scale consistent with all other sites so that the aggregator can determine the classification with the highest confidence.

Each trouble site in the network is connected to two or more "source sites" (either primary sites or other trouble sites) that forward trouble records to the trouble site, as shown in Fig. 1.

```
1  Procedure processRecord(site, record):
2      Classify record at site to get the classification-result;
3      aggregateClassification(site, record-key,
       classification-result, t-site-forwarding);
4      foreach connected t-site do
5          if classification-result confidence < threshold from
           site to t-site then
6              forwardTroubleRecord(t-site, record);
7          end
8      end
9  Procedure forwardTroubleRecord(t-site,
   record-fragment):
10     if t-site has received a record-fragment from all source
       sites then
11         processRecord(t-site, merged-record);
12     else
13         buffer record-fragment at t-site;
14     end
15 Procedure aggregateClassification(site, record-key,
   classification-result, t-site-forwarding):
16     log classification-result as received from site for
       record-key;
17     foreach t-site in t-site-forwarding do
18         if site forwarded record-key to t-site AND record-key
           has not been logged as unexpected from t-site then
19             log that a result for record-key may be expected
               from t-site;
20         else
21             log that a result for record-key is unexpected
               from t-site;
22         end
23     end
24     if all p-sites and expected t-sites have provided a
       classification-result then
25         aggregate all classifications for record-key into a
           final-classification;
26     end
```

Algorithm 1: Procedures for primary, trouble, and aggregator site behavior.

A trouble site can be said to be of a certain "order", which is one higher than the order of any of its source sites (where primary sites are of order 0). Procedure `forwardTroubleRecord` shows that if every source site for a given trouble site "agrees" a particular record is "trouble", then that trouble site will apply its own stream classifier to a merged-record containing all of the features provided by its source sites. If any source site does not forward its fragment of a record as trouble, then the trouble site does not need to process that record, as the classification confidence was high enough at that source site to serve as a classification for the record without additional processing. To prevent a trouble site from waiting indefinitely for a record fragment that will never be sent, a fixed-size buffer may be used to store incoming record fragments. If all of the fragments arrive for a particular record, then it can be removed from the buffer and processed. However, if some fragments of a record never arrive, then it will eventually be removed from the end of the buffer to make space for new incoming record fragments. Furthermore, if the key values for each record are known to arrive at primary sites in a sequential order (e.g. when the key is a timestamp), then a trouble site may choose to stop waiting for the fragments of records from a particular source site if it receives a record from it with a later sequential key.

Every site (primary or trouble) within the network must forward its classification result (containing the predicted class label and classification confidence) for each record it processes to the "classification aggregator", which provides the final classification for all records. The simplest approach, used by Park et al. is to select the class label predicted with the highest confidence. Procedure `aggregateClassification` shows how the aggregator keeps track of which trouble sites a classification result may be "expected" from (because at least one of its source sites declared that it was forwarding the record to that trouble site) and which trouble sites a classification result is definitely "unexpected" from (because at least one of its source sites declared that it was not forwarding the record to that trouble site). In this way, the aggregator is able to produce a final classification as soon as all primary and trouble sites that will process the record have provided their classification results.

It is assumed that each site will eventually receive the true classification label of each record it processes, so that it may train its stream classifier with them. A trouble site will only be trained on a particular record if there is agreement between its source sites; it cannot be trained if it does not have the combination of features sent from its source sites.

As a concrete example of the architecture presented above, consider a scenario where we wish to perform image recognition/classification based on the video streams from two cameras. Each camera captures the portion of the image defined by its own field of view. Thus, depending on camera orientation relative to the image (i.e record) the image captured by one camera could be different from that of the other camera. We will consider an HDSM architecture with a primary site for each camera and a single trouble site that is connected to both primary sites. Each primary site can train and apply a classifier on the image frames streaming in from its camera, without the need to transmit any image data. If a primary site is not confident in its classification of a particular frame, then it will forward that frame to the trouble site. For most frames, it is assumed that the image from at least one of the cameras will contain enough information to perform classification confidently. However, if neither primary site is confident in its classification, then the frame will also be classified at the trouble site using the combined image data provided by both cameras. All class labels assigned to each record by primary and trouble sites are transmitted to a classification aggregator site, which selects the classification with the highest confidence as the final classification of the system.

3.1.1. Demonstration of transmission protocol completeness

The transmission protocol described in Algorithm 1 ensures the aggregator has all of the information it requires to produce a classification for every record. To see the completeness of the transmission protocol, consider a hierarchy containing two primary sites (p_1 and p_2) that feed trouble records to a single trouble site (t). The protocol will be applied to classify a record r with key k_r divided into two fragments r_1 and r_2 arriving at sites p_1 and p_2 respectively. Classifications of record r from sites p_1, p_2, and t will be represented as c_1, c_2, and c_t respectively. The completeness of transmission to the aggregator is demonstrated below under the four possible cases of primary site confidence for a particular record.

The first possible case of primary site confidence is that both p_1 and p_2 classify the record with confidence values exceeding their local confidence thresholds (i.e. both sites are confident). In this case, each primary site will send its classification (c_1 or c_2 respectively) to the aggregator along with the notice that it is not forwarding the record to the trouble site t (`t-site-forwarding` $= \{t : 0\}$ in the call to `aggregateClassification` on line 3 of Algorithm 1). Therefore, when the aggregator is processing each primary site classification, the condition on line 18 will evaluate as false for trouble site t, and the record will be logged as unexpected from t (line 21). This will cause the condition on line 24 to evaluate as true when processing the second classification received from a primary-site (as both `p-sites` will have provided a classification, and no `t-sites` will be expected to provide a classification), and the final classification will be produced. Algorithm 2 demonstrates this case by outlining the high-level flow of the procedure calls involved (indented blocks contain procedure calls triggered by the behavior of the procedure above).

```
processRecord ($p_1$, $r_1$)
    → aggregateClassification ($p_1$, $k_r$, $c_1$, {$t$ : 0})
processRecord ($p_2$, $r_2$)
    → aggregateClassification ($p_2$, $k_r$, $c_2$, {$t$ : 0})
```
Algorithm 2: Procedure calls for case when both p_1 and p_2 are confident.

In the second case, primary site p_2 confidently classifies the record, while p_1 is not confident in its classification (i.e. for p_1, `t-site-forwarding` $= \{t : 1\}$ on line 3 of Algorithm 1 and the condition on line 5 evaluates as true for trouble site t). Because t will only receive a record fragment from one of its source sites (r_1 from p_1), t will still not process the record (as the condition on line 10 will not evaluate as true). However, the primary sites will now differ in their decision to forward the record, resulting in the condition on line 18 for trouble site t evaluating as true for p_1 and false for p_2. Because logging that a classification is unexpected from t on line 21 will override logging that a classification *may* be expected from t on line 19, a classification will not be expected from t regardless of the order in which the classifications are received from p_1 and p_2. The aggregator site can then produce the final classification for the record as soon as messages arrive from p_1 and p_2 without waiting for a message from t. Algorithm 3 shows the flow of procedure calls for this case.

```
processRecord ($p_1$, $r_1$)
    → aggregateClassification ($p_1$, $k_r$, $c_1$, {$t$ : 1})
      forwardTroubleRecord ($t$, $r_1$)
processRecord ($p_2$, $r_2$)
    → aggregateClassification ($p_2$, $k_r$, $c_2$, {$t$ : 0})
```
Algorithm 3: Procedure calls for case when p_1 is not confident and p_2 is confident.

The third possible case of primary site confidence is for primary site p_1 to be confident in its classification while p_2 is not. This case is identical to the second case, except that the roles of p_1 and p_2 are reversed, and therefore does not require further analysis.

In the fourth and final case, both primary sites p_1 and p_2 are not confident in their classification of the record. Because both primary sites forward their fragments to trouble site t, t will process the record and transmit its classification (c_t) to the aggregator (the condition on line 10 of Algorithm 1 evaluates as true for t). The aggregator will know to wait for the classification from site t because the condition on line 18 will evaluate as true for classifications from both primary sites, resulting in t being added to the list of `expected t-sites`. Therefore, the condition on line 24 will only evaluate as true once classifications have been received from both `p-sites` and `t-site` t, resulting in a final classification that takes into account all produced site classifications. This is true as long as the (size-limited) buffer at t is large enough to not drop a record fragment from one source site before there has been adequate time to receive fragments from the other source site. Algorithm 4 shows the flow of procedure calls for the primary sites and trouble site in this final case.

```
processRecord ($p_1$, $r_1$)
    → aggregateClassification ($p_1$, $k_r$, $c_1$, {$t$ : 1})
      forwardTroubleRecord ($t$, $r_1$)
processRecord ($p_2$, $r_2$)
    → aggregateClassification ($p_2$, $k_r$, $c_2$, {$t$ : 1})
      forwardTroubleRecord ($t$, $r_2$)
       /* $t$ has received fragments from all source
          sites, and therefore processes $r$          */
       → processRecord ($t$, ($r_1 + r_2$))
          → aggregateClassification ($t$, $k_r$, $c_t$, {})
```
Algorithm 4: Procedure calls for case when both p_1 and p_2 are not confident.

The properties demonstrated in this section are also generalizable to trouble site hierarchy configurations involving more primary sites and multiple layers of trouble sites. It is straightforward to see that the aggregator will receive enough information to know when to expect classifications for trouble sites with more than two source sites. Furthermore, the forwarding behavior of trouble sites acting as source sites for higher-order trouble sites will be the same as that of primary sites.

3.2. Protocol for regulating data transmission

Park et al. used a static confidence-threshold to determine which records to forward as trouble records, which may result in vastly different volumes of data being forwarded depending on how confident a site's classifier is. This is acceptable when dealing with non-streaming data, as the static threshold can be selected after evaluating the distribution of confidence values achieved at each primary site. However, in a data stream mining context, the confidence of site classifiers may change over time, which could lead to overloading trouble sites with more data than they are capable of receiving. HDSM addresses this problem: each source site forwards a fixed proportion of its records below a "quantile-threshold", which is determined by the volume of data the trouble site can process. By finding the rank of a confidence value within a sliding window of recent local classification confidences, a site can determine whether that confidence value falls below the quantile-threshold.

A number of different factors must be taken into account when determining the quantile-threshold. Firstly, the combined volume of data forwarded from all source sites should be approximately equal to the volume of data that can physically be processed by the trouble site, irrespective of the number of source sites or the size of the feature set each provides. Secondly, the proportion of records forwarded from a particular trouble site should not be affected by the level of "agreement" at that site (i.e. the proportion of received record fragments that were actually processed). Finally, the number of trouble records sent by each source site should be approximately equal, even though record fragments from different source sites may be of different sizes. Because a trouble record is only processed at a trouble site if fragments are received from all source sites, it is not worthwhile sending more trouble records from one source site than from another.

Before an expression for the quantile-threshold can be defined, a method for quantifying the agreement at a particular trouble site must be established, which should represent the probability that all source sites agree any received record is "trouble". To measure the agreement, a trouble site must maintain a sliding "agreement-window" with entries for each unique record received by the site. Each entry records the number of source sites that forwarded their fragment of that unique record. Consider a scenario involving three records (A, B and C) and four source sites sending record fragments to a single trouble site. The first three source sites send fragments of records A and B in sequence, while the fourth sends fragments of records A and C in sequence. This transmission will result in three window entries: [4, 3, 1] (record A was forwarded by all 4 source sites, B by 3 source sites, and C by only a single source site). Out of the record fragments sent from each source site, there was agreement on only one record (A) which accounts for 4 matches out of a total transmission of 8 fragments, and so the level of agreement is 0.5, as illustrated by Eq. (1).

$$a_t = s\left(\frac{\sum_{i=1}^{s} I(w_i = s)}{\sum_{i=1}^{s} w_i}\right) \tag{1}$$

Eq. (1) defines the level of agreement a_t at a trouble site t; s is the number of source sites that feed data to the trouble site; $I(k)$ is the identity function that returns 1 if k is true and 0 otherwise, and w_i is the ith entry in the agreement-window. Therefore, $I(w_i = s)$ equals 1 if there is agreement for the ith entry, and 0 otherwise.

Applying Eq. (1) to the example above results in: $a_t = 4 \times \frac{1}{8} = 0.5$.

To quantify the diminishing number of records transmitted between successive trouble sites (as more features are processed, fewer records can be processed), the concept of a reduction coefficient needs to be defined. This reduction coefficient represents the probability that a site will both receive and process any given record. Eq. (4) defines an expression for computing the reduction coefficient (r_s) at site s. For primary sites, which process all records entering the system, the r_s coefficient is always equal to one. For a trouble site, r_s is the product of the trouble site's agreement (a_s) with a scaling factor (k_s) that regulates the maximum number of records that can be processed at a given trouble site. The scaling factor takes into account the processing capacity of a trouble site; a scaling factor of k_s indicates that the trouble site is capable of processing k_s times the combined volume of data available at its feeder source sites. The scaling factor is expressed in terms of a user-configured trouble factor (t_s), which allows the user to control the volume of data that will be processed at a trouble site independently of the sizes of its source sites' feature sets. A typical trouble factor would be the maximum feature set size taken over the set of primary sites, which indicates that the trouble site is capable of receiving the same volume of data as any primary site.

$$l_s = \sum_{i=1}^{m_s} l_s^i \tag{2}$$

$$k_s = \frac{t_s}{l_s} \tag{3}$$

$$r_s = \begin{cases} 1 \text{ if s is a primary site} \\ a_s \times \min(1, k_s) \text{ otherwise} \end{cases} \tag{4}$$

where l_s is the size of the feature set at site s, m_s is the number of features processed at site s, and l_s^i denotes the size of feature i at site s.

Having defined agreement and the reduction coefficient, we can now use Eq. (5) to compute the quantile threshold ($q_{s\Rightarrow d}$) that determines the fraction of records that will be forwarded from a given source site s to a destination trouble site d.

$$q_{s\Rightarrow d} = \min\left(1, \frac{k_d}{r_s}\right) \tag{5}$$

Eq. (5) shows that the quantile threshold is essentially the inverse of the reduction factor achieved at the source site s with the product of the scaling factor that can be tolerated at the destination site d.

Note that when the source site s happens to be a trouble site, the quantile threshold becomes inversely proportional to the agreement at s (through the definition of r_s). This means that a trouble site with low agreement may send nearly all of the records it processes to a subsequent trouble site. Because of this and the fact that a trouble site with low agreement will make use of far less data than it receives; trouble sites with low agreement should be avoided in practice.

This quantile-based approach to controlling data transmission ensures that a trouble site receives a proportion of the dataset that is within its processing capacity, and no more. This control is exercised by the trouble factor parameter, and solves the issue of flooding a trouble site with too many records when the confidences of its source sites' classifiers drop.

Lemma 1 shows that the quantile threshold guarantees that a trouble site will receive an equal proportion of records (r_a, r_b) from each of its source sites a and b respectively. Any imbalance in the number of records sent from source sites would not lead to an increase in agreement but would incur unnecessary transmission overhead, and so the quantile threshold needs to ensure this balance in order to maintain transmission efficiency.

Lemma 1. *For all source sites (S_d) of a trouble site d, the proportion of trouble records transmitted by each source site is equal to that transmitted by every other source site, that is, we have: $r_a \times q_{a\Rightarrow d} = r_b \times q_{b\Rightarrow d} \forall a, b \in S_d$, provided there is a sufficient number of records available at each source site. (i.e. the quantile threshold q does not need to be limited to its maximum value of 1).*
The proportion of all records sent by source site a to its trouble site is given by:

$$\begin{aligned} r_a \times q_{a\Rightarrow d} &= r_a \times \frac{k_d}{r_a} \quad \textit{by def. of } q_{a\Rightarrow d} \\ &= k_d \end{aligned} \tag{6}$$

Similarly, the proportion of all records sent by source site b to the trouble site is given by:

$$\begin{aligned} r_b \times q_{b\Rightarrow d} &= r_b \times \frac{k_d}{r_b} \quad \textit{by def. of } q_{b\Rightarrow d} \\ &= k_d \end{aligned} \tag{7}$$

Furthermore, Lemma 2 demonstrates that the trouble factor (t_d) and the size of the feature set at a trouble site (l_d) control the maximum number of records that will be processed by the trouble site. This control is needed as the system can then ensure that each trouble site is not overloaded with more trouble records than it can handle.

Lemma 2. *The proportion of records r_d processed by the trouble site will not exceed the ratio $k_d = \frac{t_d}{l_d}$ of the trouble factor (t_d) to the size of the feature set (l_d) at the trouble site d.*

$$\begin{aligned} r_d &= a_d \times \min(1, k_d) && \textit{by def. of } r_d \\ &\le a_d \times k_d && \textit{as } a_d \ge 0 \\ &\le k_d && \textit{as } a_d \le 1 \end{aligned} \tag{8}$$

3.3. Learning trouble site hierarchies

By supporting multiple trouble sites, HDSM allows many possible trouble site hierarchies for a given set of primary sites. Some trouble site hierarchies may be partially or entirely determined by physical limitations, such as introducing trouble sites for groups of physically neighboring primary sites. In other cases, where there is no practical basis for a particular configuration, a hierarchy must be selected that achieves the best classification accuracy

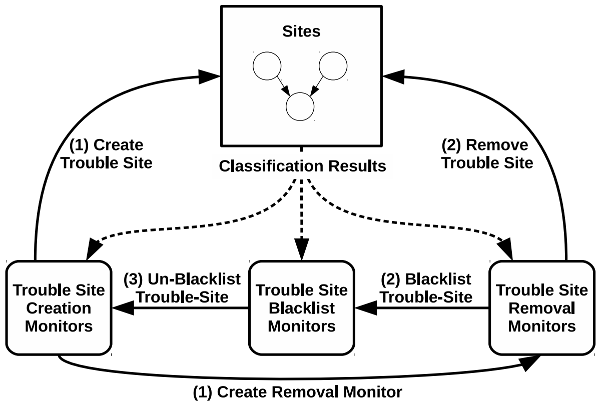
Sites
(1) Create Trouble Site
(2) Remove Trouble Site
Classification Results
Trouble Site Creation Monitors
(3) Un-Blacklist Trouble-Site
Trouble Site Blacklist Monitors
(2) Blacklist Trouble-Site
Trouble Site Removal Monitors
(1) Create Removal Monitor

Fig. 2. Continuous process for creating, removing, and un-blacklisting trouble sites.

gains with the least data transmission. The optimal hierarchy will contain trouble sites that capture cross-terms between the features of different primary sites. Furthermore, as the underlying patterns and cross-terms in the data streams change over time, trouble sites may need to be added and removed from the hierarchy to optimize performance. The following section presents an online approach to learning and adapting the trouble site hierarchy.

To begin learning the trouble site hierarchy, the system can be initialized with a minimal structure consisting solely of the primary sites and required classification aggregator, with no trouble sites. Fig. 2 demonstrates three sets of "monitors" that will then add or remove trouble sites based on the classification results of the system. Firstly, the "trouble site creation monitors" watch for possible sets of source sites that achieve agreement above a threshold for a sufficient period of time, which indicates a new trouble site should be created. Step 1 shows the creation of a trouble site along with a "trouble site removal monitor" to evaluate both how often the trouble site is used for the final classification and how accurate it is when used. The trouble site will be removed (step 2) if its utilization or accuracy drops below a threshold for a sufficient period of time, and it will also be added to a "blacklist" of trouble sites that should not be recreated until they are "un-blacklisted". The "trouble site blacklist monitors" then watch for concept drifts in the data streams that indicate a trouble site should be un-blacklisted (step 3), which will allow a creation-monitor to recreate them if and when appropriate. To make these evaluations, each type of monitor must receive for each record: the final classification result of the system; the identifier of the site that provided the final classification; the individual classifications and confidences of each site; and finally, a representation of the current structure of the hierarchy. The remainder of this section describes the behavior of each type of monitor in more detail.

A creation monitor will always exist for each combination of existing sites that could be used as the source sites for a new trouble site, provided that trouble site is not currently blacklisted. In the current implementation of HDSM, the set of potential source site combinations is limited to prevent an explosion of monitors when there are many potential trouble sites. Specifically, potential source sites must not already be acting as a source site to a different trouble site, which reduces redundant transmission of the same features to multiple trouble sites. Additionally, only pairs of source sites are allowed, as agreement will generally drop with a greater number of source sites, resulting in more fruitless data transmission. Each creation-monitor uses the same window-based agreement monitoring method described in the previous section, but only considers records that were classified incorrectly by the entire HDSM classifier, as there is no need to improve the classifications of records that are already classified correctly. If the level of agreement in the window rises above a threshold (T_a) for a configured "threshold-time-period" (a number of records), then the trouble site is created.

A trouble site must be removed when it ceases to provide a meaningful benefit to the accuracy of the overall classifier, typically when there is some concept drift in the data streams. It is possible that a trouble site never provides a meaningful benefit, as creation-monitors must decide whether to create trouble sites based on limited information that may not reflect the actual benefit of the trouble site to the accuracy of the system. It is also important that a trouble site being used as a source site for another trouble site is not monitored for removal, as the subsequent trouble site may be providing a benefit even if the intermediary trouble site is not. A removal-monitor will use sliding windows to monitor both how often the trouble site is used as the final classification of the overall classifier, and the accuracy it achieves when it is. If either the utilization or accuracy drop below their respective thresholds (T_u and T_{acc}) for a configured "threshold-time-period" (a number of records), then the trouble site is removed and blacklisted.

For each blacklisted trouble site, a blacklist-monitor will be created to watch for drift in the accuracy of the classifier at each source site. If drift is detected at any source site, then a cross-term may have appeared in the dataset that the trouble site could capture. As a result of this, the trouble site may now meet the criteria for creation once again. The trouble site is therefore removed from the blacklist, which allows a new creation-monitor to be created for it. For experimentation in this paper, ADWIN [34] was used for drift detection in blacklist-monitors.

3.3.1. Monitor thresholds

The creation and removal monitors use four thresholds to determine when to trigger their respective actions. Two of these must be configured through user-defined parameters: both monitor types' threshold-time-period (set to a sufficient period of time to judge the permanence of a pattern) and the removal monitor's utilization-threshold (set based on how frequently a site must be used for it to justify its real-world resource cost). Conversely, the removal-monitor's accuracy-threshold and creation-monitor's agreement-threshold can be set dynamically based on stream behavior, as shown in Eqs. (12) and (13) and described below.

$$\tilde{a} = \prod_{i=1}^{s} p_i \tag{9}$$

$$\Theta = 1 - \tilde{a}^{\theta} \tag{10}$$

$$\varepsilon = \sqrt{\frac{\ln(1/\delta)}{2n}} \cdot R \tag{11}$$

$$T_{acc} = \min\left(1, acc_{o-1} + \varepsilon\right) \tag{12}$$

$$T_a = \min\left(1, \tilde{a} + \Theta\varepsilon\right) \tag{13}$$

A trouble site's accuracy-threshold (T_{acc}) must require a significant improvement over the accuracy that would be achieved without that site. Specifically, accuracy should be compared with the accuracy that could be achieved by aggregating the classifications of only lower-order sites (acc_{o-1}), which prevents simultaneously monitored sites of the same order being removed for mutually redundant accuracy improvement. Therefore, the accuracy threshold is set to this lower-order accuracy plus an additional term that demands a significant improvement (ε).

Similarly, the agreement-threshold (T_a) must require a significant improvement over the level of agreement that would be expected by chance ($\tilde{a}$). $\tilde{a}$ can be calculated as the expected proportion of overlap between the trouble record sets transmitted by source sites (s) if trouble records were selected randomly, which is the product of the proportion of records (p) each source site transmits. As with the accuracy-threshold, the required improvement is represented by the ε term, but it is smoothed by an additional term (Θ). This smoothing term diminishes the required improvement as expected agreement approaches 100%, where exceeding the expected agreement is less likely. The degree of smoothing is determined by a manually set smoothing-factor (θ).

For both dynamically set thresholds, the term to require significant improvement (ε) is provided by the Hoeffding bound [35]. Application of the Hoeffding bound is reasonable, as the measured accuracy and agreement values can be considered sums of bounded independent random variables (the binary accuracy and agreement values for each observed record in the accuracy and agreement windows, divided by the size of the monitor window). When computing the Hoeffding bound, the range (R) is 1 for both accuracy and agreement (which vary from 0%–100%). The number of observations n is the current size of the window monitoring accuracy or agreement plus the threshold-time-period, as this represents the size of the entire sample the improvement will be evaluated over. The δ parameter of the Hoeffding bound determines the confidence level $(1-\delta)$ for ϵ, and hence δ is set to a small value, such as 10^{-3}. The influence of the Hoeffding bound is also restricted to never increase either threshold above the maximum possible accuracy and agreement value of 100%, as shown in Eqs. (12) and (13).

3.4. Complexity analysis

In this section we evaluate the computational complexity of procedures at the different sites within the HDSM architecture as compared to the complexities of state-of-the-art distributed ensemble methods and mining a centralized data stream. As HDSM and the state-of-the-art methods are distributed architectures that can exploit parallelism and concurrent execution, the complexity analysis will focus on achievable system throughput. Furthermore, as our analysis considers the architectures of these methods, it will be independent of the underlying stream classification algorithm used. While most data stream classification algorithms can be expected to process each record independently and only once such that the complexity will scale linearly with the number of records in the data stream, different algorithms will scale differently with the size of a record's feature set. We therefore represent the complexity of processing a single record as $O(m^{\alpha})$, with the value of α determining the complexity class with respect to the number of features m. Such a representation covers the complexity classes of sub-linear, linear, and super-linear classification functions, and allows the complexity of processing each record in a centralized data stream to be represented as $O(m^{\alpha})$.

3.4.1. Distributed ensemble complexities

The state-of-the-art distributed ensembles that will be experimentally compared to HDSM in Section 4 can all be characterized as having a two-layer architecture that determines their complexities. The first layer is comprised of local classifiers at primary sites that run in parallel to classify each record based on local feature sets. Therefore, the complexity of this layer is determined by the maximum number of features that needs to be processed at any primary site (m_p): $O({m_p}^{\alpha})$. The second layer takes the classification outputs from the first layer and applies an aggregation operation to them. For aggregation based on maximum confidence or voting, the operations can be performed in constant-time, but a stacked classifier's complexity will be determined by the number of primary sites ($|P|$) that provide classifications to the stacked classifier: $O(|P|^{\alpha})$. As both layers can execute concurrently on adjacent records in the stream, the overall complexity is that of the layer with the greatest complexity (i.e. the layer that is throttling throughput of records through the system). Therefore, distributed ensembles with aggregation based on maximum confidence or voting have complexity $O({m_p}^{\alpha})$, as do stacking-based ensembles when $m_p \geq |P|$ and which otherwise have complexity $O(|P|^{\alpha})$.

3.4.2. HDSM complexity

While HDSM is similar to the distributed ensembles in that it has a fixed layer of primary sites and a final aggregator site, it also has a dynamic hierarchy of trouble sites that depend on the outputs of primary sites and potentially other trouble sites. As there is a large degree of potential parallelism among primary and trouble sites, the throughput of HDSM is therefore determined by the complexity along the critical path: the chain of dependent sites with the greatest combined complexity that must be executed in series. Furthermore, as serially-dependent sites can execute concurrently on adjacent records in the stream, the overall complexity is determined by the maximum complexity of any single site along the critical path. To simplify the complexity evaluation, we consider a worst-case scenario in which the layers of sites are executed serially, i.e. all sites of the same order must finish processing a record before any higher order sites can begin processing that record, as illustrated in Fig. 3. As sites within the same layer can execute in parallel, and layers can execute concurrently on adjacent records in the stream, the overall complexity of HDSM is determined by the individual site with the greatest complexity (which must always be equal to or worse than the site with the greatest complexity on the critical path).

As we consider HDSM with a maximum confidence aggregator site, the constant-time complexity of the `maximum` operation over the classification confidence values received for each record will be exceeded by the complexity at primary and trouble sites. For each record processed at a primary or trouble site x, the underlying classification algorithm must be applied to the feature set available at the site ($O({m_x}^{\alpha})$) and the quantile of the latest confidence value within the window of recent classification confidences must be determined (amortized $O(1)$ with the application of the IQ agent algorithm that estimates arbitrary quantiles over a data stream [36]). As other operations in Procedures `process-Record` and `forwardTroubleRecord` of Algorithm 1 can also be performed in constant-time, the overall average complexity for processing a record at a site x is $O({m_x}^{\alpha})$. Furthermore, as HDSM's transmission protocols limit the proportion of records processed at a site x to r_x, the complexity can be amortized to: $r_x O({m_x}^{\alpha})$. Based on the definition for r_x given in Eq. (4), this can be simplified to $O({m_p}^{\alpha})$ for primary sites, and $O(\frac{t}{m_d}{m_d}^{\alpha})$ for any trouble site d (by assuming a fixed trouble factor of t for all trouble sites, a worst-case agreement of 100%, and that the number of features determines the size of the feature set ($l_d = m_d$), such that $r_d \leq k_d = \frac{t}{l_d} = \frac{t}{m_d}$). A demonstration of these site complexities is also provided in Fig. 3.

Exactly which primary or trouble site has the maximum complexity is dependent on the complexity of the underlying classification algorithm, which in turn is dictated by the value of α. Given an environment where primary and trouble sites have equal processing capacity, the trouble factor t is equal to the largest number of features at a primary site m_p. Additionally, as a trouble site will process a combination of feature sets from primary sites, we can re-express m_d as sm_p, where $s > 1$. These assertions allow us to re-express the complexity at a trouble site as $\frac{m_p}{sm_p}O((sm_p)^{\alpha}) = \frac{1}{s}O((sm_p)^{\alpha})$, which will be less than the

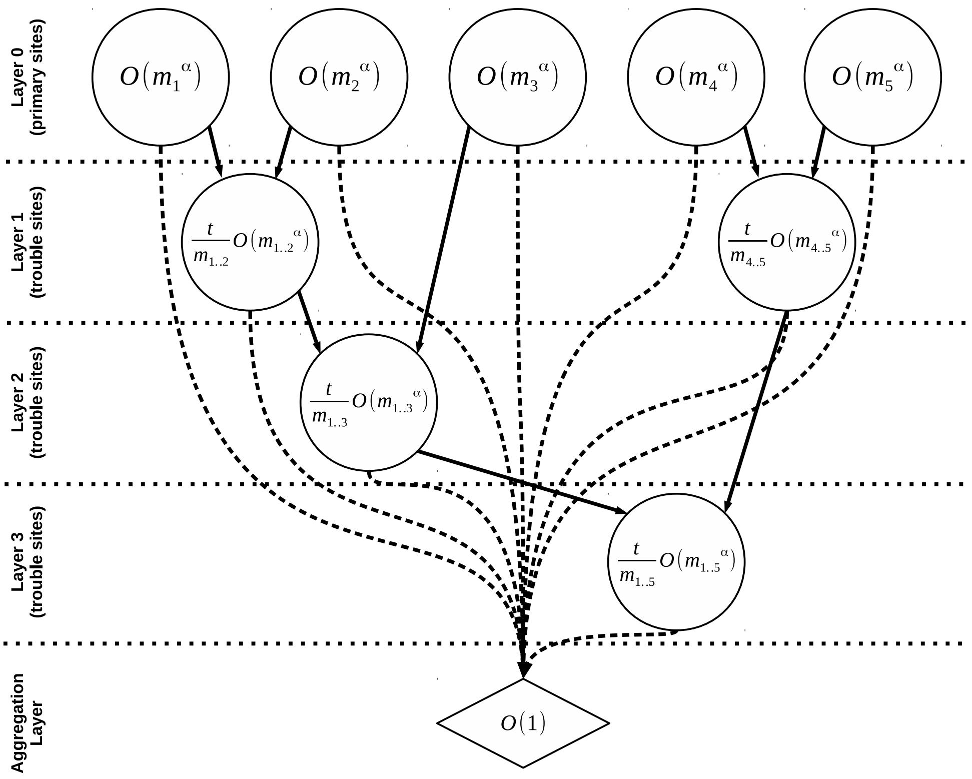

Fig. 3. Demonstration of HDSM site complexity and layering. The number of features from a range of primary sites $i..j$ is represented as $m_{i..j}$.

primary site complexity of $O(m_p{}^{\alpha})$ when the underlying classification algorithm is linear or sub-linear with respect to the number of features (i.e. when $\alpha \leq 1$), and greater than the primary site complexity when classification is super-linear with respect to the number of features (i.e. when $\alpha > 1$). Furthermore, when $\alpha > 1$, trouble site complexity will grow with the number of features (m_d), resulting in a worst-case complexity of $\frac{t}{m}O(m^{\alpha})$ when a trouble site processes the full feature set m.

3.4.3. Complexity comparison

When $\alpha \leq 1$, HDSM's complexity of $O(m_p^{\alpha})$ will outperform the complexity of a centralized classifier ($O(m^{\alpha})$), as a primary site must have fewer features than the full dataset ($m_p \leq m$). Furthermore, HDSM's complexity will be less than or equal to those of distributed ensembles: $O(m_p{}^{\alpha})$ and $O(|P|^{\alpha})$ (for stacked classifiers when $m_p < |P|$).

When $\alpha > 1$, HDSM's complexity of $\frac{t}{m}O(m^{\alpha})$ will still outperform the complexity of a centralized classifier because of the reduction in the number of records processed at the highest-order trouble site ($\frac{t}{m} \ll 1$). Also, because HDSM's complexity is still independent of the number of sites, it is able to vertically scale with more efficiency than a stacking-based distributed ensemble, which scales with the number of primary sites ($|P|$). However, HDSM's complexity will exceed that of the other distributed ensembles as well stacking when $m_p \geq |P|$. This is due to the fact that these ensembles perform minimal or no modeling of combined feature sets, which results in substantially better performance when $\alpha > 1$. However, this also means they are unable to learn from cross-terms between feature sets. It should also be emphasized that it is rare for a classification algorithm to scale super-linearly with the number of features ($\alpha > 1$). Many typical classification algorithms scale linearly with the number of features, such as Naive Bayes, Bayesian Networks, and Neural Networks (E.g. doubling the number of input features doubles the number of links to the first hidden layer).

As Park's approach can be considered a special case of HDSM in computational terms (having a single trouble site that uses all primary sites as source sites), it has an equivalent computational complexity to HDSM.

3.4.4. HDSM monitor complexity

HDSM's creation, removal, and blacklist monitors must also perform update operations for each record. However, these monitors do not impact the critical path (as they can be executed after classification) and they are less computationally complex than primary and trouble sites. Creation monitors only perform constant-time operations to check confidence values and update their agreement windows, as do removal monitors to update and check their accuracy and usage windows. The complexity of each blacklist-monitor update depends on the drift detection method used, which is amortized $O(1)$ for the ADWIN method used in this paper [34]. Additionally, while removal-monitors require a single update for every record, creation and blacklist monitors require an update for both of the classifications received from the two source sites,[1] but only for the proportion of records received by the monitor x (k_x, which cannot represent a proportion of records greater than 1). This results in an amortized complexity of $k_xO(1)$ for creation and blacklist monitors, and $O(1)$ for removal-monitors.

3.5. Alternative aggregation methods

Apart from selecting the classification with the highest confidence, it is also possible to use other ensemble learning methods at the aggregator. However, because other methods do not necessarily select a single site's classification as the "winner", they may not be compatible with the utilization monitoring of trouble sites

[1] The implementation of HDSM in this paper uses two source sites for each trouble site.

by removal-monitors. For this reason, aggregation with other methods should be performed in two stages: first using whatever alternative method was selected to aggregate primary site classifications, and then aggregating the produced classification with those from trouble sites using the "maximum-confidence" method. This also means that the first phase of aggregation can be performed while trouble sites are still processing the record.

Two alternative aggregation methods considered in the experimental evaluation below are voting and stacking. Voting is based on the approach of Skillicorn and McConnell [27], where the class with the majority of votes from primary sites is selected, and the confidence is the mean confidence of all primary sites. Stacking is based on the approach of Parker et al. [28]. Primary site classifications are used as the input records to another stream classifier (using the same or a different algorithm to that used at primary sites and trouble sites), which produces a new classification and confidence.

4. Experimental evaluation

HDSM was implemented for experimental evaluation using Clojure and machine-learning algorithms from the MOA data stream mining system [37]. The source-code (including data preparation and experimentation) has been made available in a GitHub repository.[2] The experiments presented below illustrate HDSM's ability to improve classification accuracy with limited transmission of data features from primary sites to trouble sites, including: a demonstration of the behavior of HDSM's dynamic trouble-site hierarchy, a comparison of HDSM's performance with other distributed stream mining approaches on a variety of real-world and synthetic stream datasets, and a demonstration of HDSM's potential to dynamically trade off between accuracy and response time.

All experiments were performed with OpenJDK 1.8.0 on a 64-bit Ubuntu 16.04 installation running on a 4×2.60 GHz Intel Core i5 CPU with 8 GB of memory. As all of the data features within each experiment were of the same data-type, they were treated as being the same size ($l_x^i = 1$), meaning that a trouble factor equal to the number of features at a primary site would represent a trouble site capable of receiving as much data as a primary site. The list below summarizes configuration parameters that were kept constant throughout experimentation (except where noted otherwise):

- Size-limit for all windows at sites and monitors: 1000 latest records
- Threshold time period for creation and removal monitors: 500 records
- Hoeffding bound δ in agreement and accuracy thresholds: 10^{-3}
- Smoothing factor for agreement threshold (θ): 5
- Site utilization threshold for removal-monitors (T_u): 5%

4.1. Performance evaluation metrics

The following section describes the metrics that were used to compare the accuracy, volume of data transmission, and computational resource time for different stream mining methods. The metrics are also summarized in Table 1. In order to evaluate the typical running state of each method, the first 1000 records of each experiment were considered a "warm-up" time and ignored in all metric calculations.

Accuracy was evaluated prequentially: each record was classified and then immediately used for training. Each site that processed a record was only trained with the features that were transmitted to that site.

Table 1
Performance evaluation metrics summary.

Metric	Description
Accuracy	Prequentially evaluated classification accuracy.
TDTV	Total Data Transmission Volume to trouble sites.
MDTV	Maximum Data Transmission Volume to any single trouble site.
MTCP	Mean Resource (CPU) Time on Critical Path to final classification.
MTBC	Mean Resource (CPU) Time Between classification Completions.

Record feature communication is reported in terms of TDTV (Total Data Transmission Volume between all sites) and MDTV (Maximum Data Transmission Volume to any single trouble site, averaged across windows of 100 records). Both transmission metrics are reported as proportions of the total data volume contained in the dataset. TDTV may exceed 100% in cases when record fragments are transmitted through multiple layers of trouble sites. Transmission of site classifications to the aggregator is not reported, as transmission of class labels from primary sites is the same for all evaluated methods while being a much less significant cost in comparison to feature transmission. Because of this, all experiments that did not result in transmission of data features to trouble sites have 0% TDTV and MDTV.

Computational resource time is reported in terms of MTCP (Mean Resource Time on Critical Path) and MTBC (Mean Resource Time Between Completions). MTCP is the cumulative CPU time across the longest-running sequence of classifiers used to process a record, averaged over all records. MTBC is the mean difference between the completion times of successive records, given each classifier can begin processing a record after its source sites have processed it and after the site has finished processing the previous record. While MTCP measures the classification latency (response time) for a single record on average, MTBC measures the throughput rate for stream-processing. Resource time includes the CPU processing time of classifiers at primary and trouble sites and stacked aggregators; maximum confidence aggregation and voting overheads were excluded because these costs are less significant and the underlying operations can be parallelized in practice. To account for JVM warm-up, each set of time-evaluated experiments was performed twice in sequence, and timing was recorded from the second run. Both MTCP and MTBC are reported in nanoseconds.

4.2. Demonstration of dynamic trouble site hierarchy

To demonstrate the behavior of HDSM's dynamic trouble site hierarchy, it was tested with a synthetic dataset that abruptly drifted between different underlying cross-terms. A dataset of 30,000 records was generated with 6 binary features and a binary class. Eq. (14) defines the rule used to generate a cross-term dependent class value from random binary features. The dataset always contained two cross-terms ($m = 2$) between two features ($k = 2$): the first and last 10,000 records had cross-terms between feature pairs [0, 1] and [2, 3], while the middle batch of 10,000 records had cross-terms between pairs [0, 1] and [4, 5]. The features were distributed across 6 primary sites, with each site receiving a single feature, thus representing a high degree of data distribution. HDSM was configured with a trouble factor of 1 so that each trouble site would receive approximately the same

2 https://github.com/ben-denham/hdsm.

Table 2
Effects of trouble site hierarchy on performance with a concept drifting data stream.

Site	Without trouble sites			With trouble sites		
	Accuracy			Accuracy		
	Utilization	Contribution	TDTV	Utilization	Contribution	TDTV
0	20.69%	13.10%	N/A	12.41%	10.80%	N/A
1	37.75%	26.76%	N/A	24.96%	21.78%	N/A
2	16.80%	8.92%	N/A	8.62%	8.04%	N/A
3	13.21%	6.70%	N/A	5.83%	5.03%	N/A
4	7.74%	4.91%	N/A	4.60%	4.30%	N/A
5	3.80%	1.68%	N/A	1.87%	1.64%	N/A
[0, 1]				21.43%	21.35%	16.17%
[3, 2]				14.17%	12.11%	12.10%
[4, 5]				6.11%	4.29%	8.69%
[[3, 2], 5]				0.00%	0.00%	0.33%
Totals	100.00%	62.08%	0.00%	100.00%	89.35%	37.29%

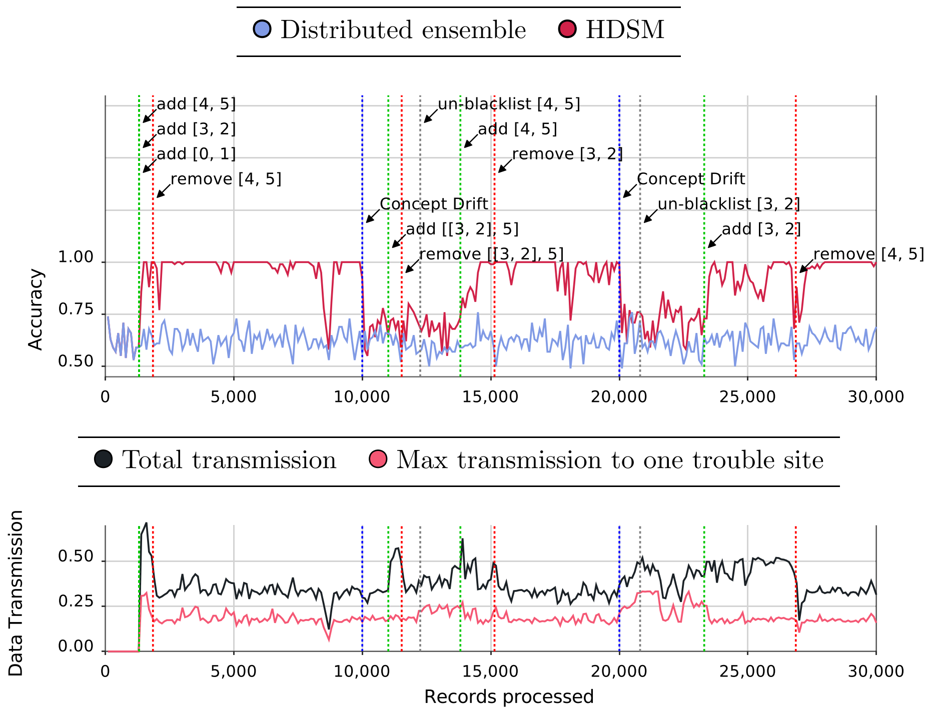

Fig. 4. Timeline of accuracy and data transmission for the synthetic cross-term dataset with concept drift points.

volume of data as any primary site (as there is one feature per site).

$$class = (f_1^1 \wedge f_1^2 \wedge \cdots \wedge f_1^k) \vee \cdots \\ \vee (f_m^1 \wedge f_m^2 \wedge \cdots \wedge f_m^k) \quad (14)$$

where k is the number of features in each cross-term, m is the number of cross terms, and f_j^i is the ith feature involved in the jth cross-term.

HDSM was compared with a distributed ensemble without trouble sites where final classifications were based solely on the maximum-confidence aggregation of primary site classifications. To allow fast adaptation to abrupt concept drift, Hoeffding trees as proposed by Hulten et al. [13] were used as classifiers at primary and trouble sites. While theoretical issues have been raised regarding the use of the Hoeffding bound in decision-tree split-decisions [38,39], it has been noted that experimental results with other methods are not conclusive [40] and that the Hoeffding tree is still considered useful as a basis for new methods as it produces satisfactory classification accuracy [35].

The utilization of each site in the classification process was tracked in Table 2. Likewise, the contribution made by each site to overall accuracy was also monitored, as was the data volume transmitted to each site. Overall, Table 2 shows clearly that HDSM significantly outperformed the distributed ensemble with an increase in accuracy of 27.27% with data transmission equivalent to only 37.29% of the dataset's total data volume. Trouble sites are represented by pairs of source sites, so it can be seen that trouble sites were created to capture the cross-term pairs ([0, 1], [3, 2], and [4, 5]), and the accuracy contributions of those trouble sites were proportional to the number of records that each cross-term was present for. A higher-order trouble site for [[3, 2], 5] was also created, but was removed due to low utilization shortly after its creation. It can also be seen that the primary sites are used for classification less often in the configuration with trouble sites, but their accuracy contribution drops less significantly when compared to their drop in utilization, thus indicating that the trouble sites are successfully processing records that the primary sites found difficult to classify.

Fig. 4 shows how trouble sites were added and removed over the course of the HDSM experiment, and how this affected the volume of data transmitted and the accuracy improvement over the distributed ensemble. Because the system begins with no trouble sites, there is initially no data transmission and no

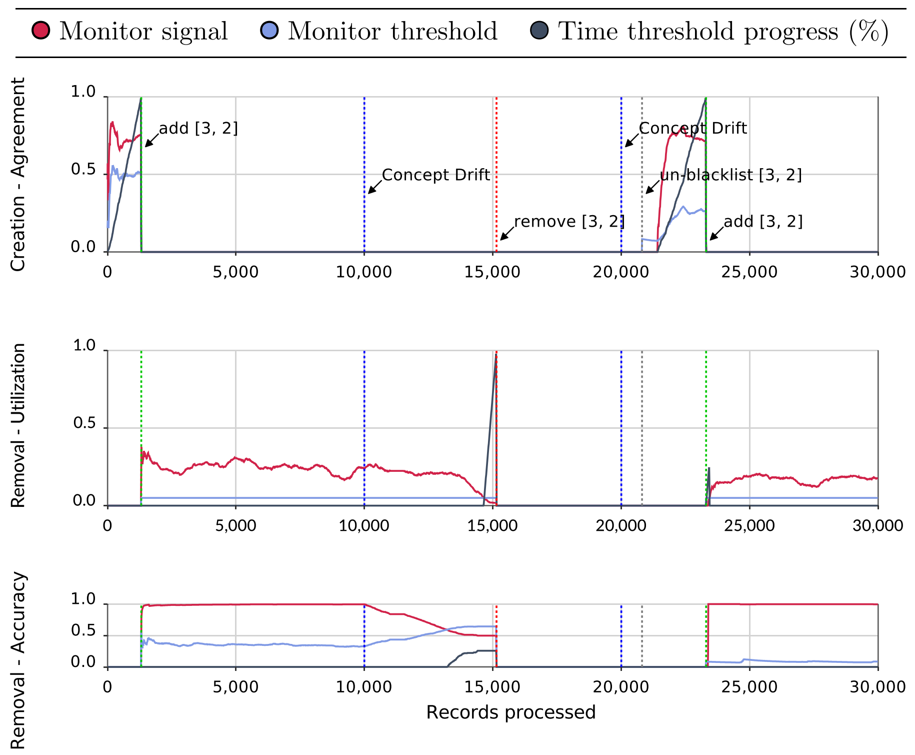

Fig. 5. Timeline of trouble site [3, 2] creation (agreement) and removal (utilization and accuracy) monitors for the synthetic cross-term dataset with concept drift points.

difference in accuracy between HDSM and the distributed ensemble. However, after processing approximately 1300 records, three candidate trouble sites exceed the required agreement-threshold and are therefore added to the system. Trouble sites [0, 1] and [3, 2] capture the cross-terms in the dataset and boost system accuracy at the expense of increased data transmission. The trouble site [4, 5] is quickly removed (and therefore blacklisted) because that cross-term is not yet present, and the site's classifications are not confident enough to be used for the final classification. After the concept drift point at 10,000 records, HDSM's accuracy drops closer to the distributed ensemble's accuracy, and there is renewed activity in altering the trouble site hierarchy. Eventually, the trouble site [4, 5] is un-blacklisted (triggered by the concept drift) and the trouble site is recreated, improving accuracy by capturing that introduced cross-term. The trouble site [3, 2] is removed because its utilization has dropped due to the fact that cross-term is no longer present in the data stream. Finally, after the second drift point at 20,000 records, trouble site [3, 2] is removed and trouble site [4, 5] is re-created, reflecting the shift back to the original cross-terms. Trouble site [0, 1] is never removed because that cross-term is always present and the site is consistently used to improve classification accuracy. While the level of total transmission fluctuates depending on the number of trouble sites, the maximum transmission to any trouble site remains relatively stable near the proportion determined by the trouble factor (in this case, approximately 16.67% of the dataset).

Fig. 5 demonstrates the behavior of the monitors for trouble site [3, 2] over the course of the experiment. When the measured agreement exceeds its threshold, progress against the creation time-threshold increases until it reaches 100%, triggering the creation of the trouble site. Because agreement monitors are only updated based on incorrectly classified records, progress against the time-threshold is not perfectly linear. The creation monitor is inactive while the trouble site exists and while it is still blacklisted after removal. The removal monitors become active after the trouble site is created, and they display a high degree of accuracy and utilization while the cross-term between the two source sites is present in the dataset. After the concept drift at 10,000 records, the accuracy and utilization decrease until they drop below their respective thresholds. The time-threshold for utilization is exceeded first, triggering the removal of the site. Finally, it can be seen that while the utilization threshold for removal remains constant over time, the agreement and accuracy thresholds vary as site communication affects the expected agreement. A short period can also be observed around the 11,000 record point where the monitor values for utilization and accuracy do not change. This is because another trouble site ([[3, 2], 5]) is created that depends on [3, 2], suspending its removal monitors until site [[3, 2], 5] is removed.

4.3. Performance evaluation and comparison

The following experiments compare the performance of HDSM with four previously proposed distributed data stream mining algorithms. Three of these algorithms can be referred to as distributed ensemble methods, as they use local classifiers at primary sites and transmit local classifications to a central site for aggregation. The three methods differ in the rule that is used to aggregate classifications. DMaxConf selects the local classification with the highest confidence, which is the same aggregation rule used by Park et al. [8]. DVote selects the classification with the majority vote, as used by Skillicorn and McConnell [27]. DStack trains a stacked classifier on the local classifications, which is the basis of the approach proposed by Parker et al. [28]. HDSM was also tested using all three of these aggregation rules to aggregate primary site classifications, as described in Section 3.5. The fourth algorithm compared to HDSM is the original approach of Park et al. with a minor modification to make it suitable for streaming data: the single, fixed trouble site that uses all primary sites as source sites was configured with the same dynamic confidence quantile-threshold as HDSM. Without this modification, a static

Table 3
Properties of datasets used for performance evaluation.

Dataset	Features	Features per site	P-sites	Classes	Records	Stream?
EEG	14	1 (site per sensor)	14	2	14,980	Shuffled
GAS	128	8 (site per sensor)	16	6	13,910	Stream
HIG	19	3–4 (see text)	5	2	10,000	Stream
FLT	20	1–3 (see text)	9	2	25,034	Shuffled
OCC	3	1 (site per sensor)	3	2	20,560	Shuffled
SDD	48	4 (site per logical set)	12	11	58,509	Interleaved
WFR	24	4 (adjacent sensors)	6	4	10,913	Stream
RBF-F	10	1	10	5	50,000	Fast incremental drift
RBF-M	10	1	10	5	50,000	Moderate incremental drift
SEA-A	3	1	3	2	100,000	Abrupt drift
SEA-G	3	1	3	2	100,000	Gradual drift

threshold would need to have been set, which would not have adequately controlled the trouble record transmission rate.

A combination of real-world and synthetic stream datasets were used in these experiments. The particular real-world datasets were selected because they represent data streams that could be vertically-distributed in practice and because they exhibit cross-terms between distributed feature-sets. The seven real-world datasets were EEG Eye State (EEG), Gas Sensor Array Drift (GAS), HIGGS (HIG), NASA FLTz (FLT), Occupancy Detection (OCC), Sensorless Drive Diagnosis (SDD), and Wall-Following Robot Navigation (WFR). Additionally, four synthetic stream datasets that had previously been used in evaluating the Adaptive Random Forest classifier [12] were selected to evaluate performance under known concept drift conditions, which were: incremental, abrupt, and gradual drift. These four datasets were based on the Radial Basis Function generator (RBF) and SEA generator [41].

Table 3 lists the properties of these datasets, which represent a range of feature counts, degrees of distribution, and class counts. Some datasets required shuffling for meaningful evaluation, as they consisted of long periods where only a single class value was present. Because the SDD dataset is composed of several streams (one for each class), the streams were interleaved to preserve time-series progression within each stream. Interleaving was performed by continuously making a random selection of which stream to draw the next record from (drawing records in order from each stream), until all records had been drawn from all streams. Only the temperature, humidity, and CO2 features were used from the OCC dataset as the light level was highly predictive on its own and the humidity-ratio was derivable from humidity and temperature. Only the raw features were used for the HIG dataset, of which each jet's features were assigned to a different primary site (3 jets, with 4 features each) and the 3 lepton features were assigned to their own primary site. Furthermore, the original HIG dataset was restricted to the first 10,000 records. Due to the small number of records in the WFR dataset, its record set was repeated once to have an adequate number of records for stream learning. The FLT dataset was constructed from a series of test flights, where each class value represented whether the moving average of the forward velocity variable over a window size of 10 records had been rising or falling. Primary sites were created for the velocity and acceleration features (along each of the lateral and vertical axes), for the reading, rate, and acceleration features (for each of pitch, yaw, and roll), and also for the readings from each of aileron, flap, and rudder features. All real-world datasets were retrieved from the UCI Machine Learning Repository [42], except for FLTz, which was retrieved from NASA's data portal [43]. To the authors knowledge, this is the first assembled set of real-world datasets for evaluating vertically-distributed data stream mining, and could therefore be used as a standard set for future research. The synthetic RBF and SEA datasets were generated with MOA [37]. The RBF generator was configured to generate 5 possible classes with 50 incrementally drifting centroids, and two datasets were generated to simulate fast incremental drift (RBF-F, with a speed-of-change of 0.001) and moderate incremental drift (RBF-M, with a speed-of-change of 0.0001). The SEA generator was configured with 3 concept drift points (at the 25,000, 50,000, and 75,000 record points), and two datasets were generated to simulate abrupt drift (SEA-A, with a drift-width of 1) and gradual drift (SEA-G, with a drift-width of 10,000).

For these experiments, Naive Bayes classifiers were used at primary sites because the algorithm is lightweight enough to run at sensor network edge nodes, and because it almost always produces varying confidence values which distinguish trouble records (unlike Hoeffding trees, which return constant confidence values when there is not enough information to perform splits). Adaptive Random Forest (ARF) classifiers [12] were used at trouble sites and for stacking, as they were found to effectively learn from cross-terms. HDSM was configured with a trouble factor equal to the maximum number of features taken across the set of primary sites. Because low accuracy was observed for Park's approach (due to low agreement, as discussed later), the reported results are for a trouble factor twice that used with HDSM to give it the best chance of achieving a high accuracy (though there is little difference to results achieved with lower trouble-factors). Tables 4 and 5 report the performance of each algorithm on each dataset.

Table 4 gives the classification accuracy (Acc%) and the standard deviation (SD) for each classifier configuration. The standard deviation was computed by dividing the stream into segments of 100 records each and then sampling over these segments. The last segment was not taken into consideration if its size was less than 50 records. Table 4 shows that for 10 out of 11 datasets, variations of HDSM outperform all other classifiers. For reference, the most accurate result (breaking ties by lowest standard deviation) for each dataset is bolded in Tables 4 and 5.

In order to evaluate the statistical significance of the accuracy differences between the different algorithms across the datasets, a Friedman test and an accompanying Nemenyi post-hoc test were performed (as recommended by Demšar [44] for such cases when comparing multiple algorithms over multiple datasets). The null-hypothesis that "all seven algorithms produce equivalent accuracies" was rejected by the Friedman test at the 95% confidence level. The proceeding Nemenyi test produced a critical difference of 2.72, revealing which algorithms were statistically significantly different to each other, as plotted in Fig. 6. Fig. 6 shows that while all HDSM variations were ranked higher than other algorithms, each was significantly different to a different subset. Of particular note is that HDSM with voting was significantly more accurate than all other non-HDSM algorithms other than DStack.

The accuracy study revealed the following trends:

- Park's approach outperformed DMaxConf by more than 1% in only three of the datasets, despite using trouble records to learn cross-term relationships. Its performance was hampered by the use of a single trouble site. A major limitation

Table 4
Accuracy comparison.

Dataset	Distributed ensembles						Park (TF 2×)		HDSM (TF 1×)					
	DMaxConf		DVote		DStack		MaxConf		MaxConf		Voting		Stacking	
	Acc%	(SD)	Acc%	(SD)	Acc%	(SD)	Acc%	(SD)	Acc%	(SD)	Acc%	(SD)	Acc%	(SD)
EEG	48.20	6.82	54.36	4.95	57.72	6.12	48.20	6.82	49.36	6.82	61.04	4.57	**61.65**	5.20
GAS	47.54	29.39	47.28	27.84	91.80	9.32	47.54	29.39	69.47	23.19	78.16	17.41	**91.94**	9.11
HIG	53.18	4.96	53.04	5.33	51.34	5.68	53.40	4.92	53.57	5.08	**53.57**	5.51	51.28	5.23
FLT	67.85	4.49	55.66	4.99	70.82	4.92	67.85	4.49	79.65	5.34	**81.78**	7.02	80.94	5.54
OCC	78.98	4.08	82.04	3.98	81.86	3.96	81.86	3.89	87.24	4.28	**90.18**	4.21	89.36	4.12
SDD	49.72	16.32	55.72	14.02	91.47	5.83	49.72	16.32	86.66	8.53	91.04	6.86	**97.17**	4.29
WFR	56.69	16.15	57.12	14.01	73.88	10.31	56.77	16.10	69.36	12.98	75.75	11.55	**80.77**	8.57
RBF-F	30.06	4.75	29.96	4.64	35.34	7.60	30.06	4.75	34.79	6.08	34.59	5.82	**37.47**	6.44
RBF-M	32.61	5.58	30.36	4.75	**58.32**	6.64	32.61	5.58	44.08	6.24	43.74	5.13	56.02	5.92
SEA-A	75.37	7.66	70.65	7.82	74.65	5.74	79.30	6.69	**82.96**	5.48	79.97	5.76	78.02	6.32
SEA-G	75.42	6.86	70.63	7.03	73.80	5.43	79.29	6.06	79.25	7.28	**79.36**	5.44	76.30	6.35

Table 5
Resource time comparison.

Dataset	Distributed ensembles						Park (TF 2×)		HDSM (TF 1×)					
	DMaxConf		DVote		DStack		MaxConf		MaxConf		Voting		Stacking	
	MTCP	MTBC	MTCP	MTBC	MTCP	MTBC	MTCP	MTBC	MTCP	MTBC	MTCP	MTBC	MTCP	MTBC
EEG	6.59E3	5.78E3	6.73E3	5.95E3	1.17E5	1.10E5	8.88E3	7.80E3	2.46E5	3.52E4	3.47E5	1.30E5	**3.54E5**	**1.43E5**
GAS	1.04E4	9.08E3	1.03E4	9.13E3	1.36E5	1.25E5	1.68E4	1.60E4	2.08E5	3.28E4	2.55E5	8.28E4	**2.17E5**	**1.50E5**
HIG	6.24E3	5.88E3	6.42E3	6.10E3	8.08E4	7.45E4	7.99E3	7.03E3	1.39E5	6.87E4	**1.50E5**	**8.21E4**	1.91E5	9.90E4
FLT	6.31E3	5.78E3	6.31E3	5.83E3	9.76E4	9.13E4	8.26E3	7.52E3	2.29E5	9.34E4	**2.70E5**	**1.16E5**	2.90E5	1.25E5
OCC	5.96E3	5.77E3	5.90E3	5.74E3	1.28E5	1.22E5	1.00E4	6.81E3	7.77E4	6.77E4	**7.46E4**	**6.56E4**	1.81E5	1.40E5
SDD	9.61E3	8.57E3	9.70E3	8.68E3	1.69E5	1.59E5	1.35E4	1.17E4	1.75E5	8.42E4	1.71E5	8.53E4	**2.54E5**	**1.85E5**
WFR	7.45E3	7.04E3	7.51E3	7.09E3	1.32E5	1.25E5	9.57E3	8.65E3	1.61E5	5.68E4	1.49E5	5.66E4	**2.67E5**	**1.74E5**
RBF-F	8.34E3	7.31E3	7.99E3	7.14E3	1.10E5	1.02E5	1.28E4	1.05E4	2.95E5	6.91E4	2.97E5	6.73E4	**3.50E5**	**1.46E5**
RBF-M	8.45E3	7.44E3	8.76E3	7.91E3	**1.47E5**	**1.39E5**	1.25E4	1.05E4	3.16E5	9.51E4	3.18E5	9.04E4	3.31E5	1.65E5
SEA-A	7.17E3	6.83E3	7.33E3	7.05E3	1.68E5	1.61E5	1.21E4	8.69E3	**8.32E4**	**7.21E4**	8.15E4	7.04E4	3.06E5	2.69E5
SEA-G	6.77E3	6.37E3	7.26E3	6.92E3	1.83E5	1.75E5	1.35E4	9.30E3	7.07E4	4.51E4	**9.21E4**	**8.03E4**	3.51E5	3.08E5

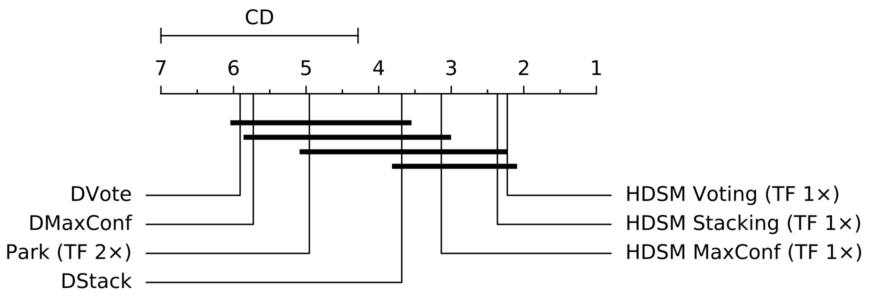

Fig. 6. Critical Difference diagram showing the statistically significant differences in accuracy between the algorithms.

of using a single trouble site is that there is agreement on only a small fraction of the trouble records transmitted by all primary sites. The implication of a low level of agreement is that the trouble site is used to classify very few records, and the training set size is also low, leading to model underfitting and loss of precision. The three datasets where Park's approach did show improvement (OCC, SEA-A, and SEA-G), all have very few primary sites, and therefore suffer less from this problem.

- On the other hand, the distributed ensembles are clearly inadequate and only emerged the winner in one dataset (RBF-M). This underperformance was due to the absence of any mechanism to learn cross-term relationships.
- The use of meta-learning significantly improved the performance of both HDSM and distributed ensemble approaches. Meta-learning methods such as voting and stacking provide opportunities for learning cross-term relationships. The accuracy of the distributed ensemble in particular was boosted significantly by stacking. However, it is interesting to note that a substantial gap in accuracy still exists between HDSM with stacking and DStack for most datasets. This can be explained by the fact that HDSM learns cross-term relationships by training on actual data features while DStack learns them from class labels which do not embody as much information as the original data itself.

Table 5 presents the resource time cost of each classifier, with the most accurate result from Table 4 bolded for ease of reference. DStack is orders of magnitude slower than the other distributed ensembles because the additional ARF stacked classifier is much more computationally expensive than the Naive Bayes classifiers at primary sites. The single trouble site in Park's approach typically has little effect on resource time because the low agreement means few records are actually processed at the trouble site. In terms of MTCP, HDSM is the slowest classifier because of its multiple layers of trouble site ARF classifiers that often process more features than even stacked classifiers (though this point of difference is diminished in cases where there are many primary sites with few features). On the other hand, HDSM is competitive to DStack in terms of the MTBC metric. This is due to two reasons. Firstly, lower order trouble sites begin processing new records before higher order trouble sites have finished processing older

Table 6
HDSM performance as a function of trouble factor.

Dataset	TF	HDSM voting						HDSM stacking					
		Acc%	(SD)	MTCP	MTBC	TDCV	MDCV	Acc%	(SD)	MTCP	MTBC	TDCV	MDCV
EEG	1×	61.04	4.57	3.47E5	1.30E5	58.72	10.55	61.65	5.20	3.54E5	1.43E5	56.22	10.28
	1.5×	64.78	5.23	5.13E5	2.66E5	91.27	13.88	63.65	5.09	4.87E5	2.63E5	84.36	13.94
	2×	66.14	4.58	5.73E5	3.25E5	104.65	14.34	66.99	5.23	5.25E5	2.99E5	95.92	15.03
GAS	1×	78.16	17.41	2.55E5	8.28E4	38.81	9.18	91.94	9.11	2.17E5	1.50E5	12.41	3.26
	1.5×	85.41	13.15	3.00E5	1.24E5	48.94	11.50	93.63	7.96	2.37E5	1.51E5	18.23	4.48
	2×	89.05	10.98	2.83E5	1.60E5	49.09	12.52	94.00	7.43	2.55E5	1.49E5	25.39	5.04
HIG	1×	53.57	5.51	1.50E5	8.21E4	54.94	23.41	51.28	5.23	1.91E5	9.90E4	43.52	21.29
	1.5×	52.76	5.12	2.44E5	1.21E5	77.95	31.23	52.91	5.67	2.95E5	1.62E5	85.10	33.83
	2×	53.11	4.81	3.95E5	2.25E5	125.21	43.24	52.92	5.23	3.62E5	1.25E5	94.03	38.45
FLT	1×	81.78	7.02	2.70E5	1.16E5	64.31	16.66	80.94	5.54	2.90E5	1.25E5	55.80	16.46
	1.5×	89.02	6.04	3.69E5	2.03E5	90.56	22.55	88.80	5.87	3.78E5	2.01E5	91.13	22.26
	2×	89.37	5.91	4.36E5	1.93E5	114.72	27.90	88.78	5.79	4.13E5	1.94E5	93.18	27.35
OCC	1×	90.18	4.21	7.46E4	6.56E4	31.17	30.12	89.36	4.12	1.81E5	1.40E5	31.34	30.29
	1.5×	89.47	4.32	1.25E5	9.56E4	69.67	45.83	90.18	3.97	2.02E5	1.44E5	48.48	45.74
	2×	89.24	6.04	1.70E5	1.24E5	77.34	51.64	87.24	5.73	1.99E5	1.36E5	56.36	42.37
SDD	1×	91.04	6.86	1.71E5	8.53E4	23.26	8.78	97.17	4.29	2.54E5	1.85E5	24.05	8.21
	1.5×	97.94	5.12	2.04E5	1.28E5	37.13	12.29	97.90	3.85	2.51E5	1.86E5	17.12	11.44
	2×	98.61	4.87	2.12E5	1.60E5	29.61	14.85	97.85	4.07	2.71E5	1.85E5	23.51	12.22
WFR	1×	75.75	11.55	1.49E5	5.66E4	51.10	20.30	80.77	8.57	2.67E5	1.74E5	47.69	18.82
	1.5×	86.23	10.82	2.90E5	1.42E5	79.86	28.57	85.32	8.46	3.45E5	1.69E5	72.46	26.92
	2×	90.32	7.69	4.00E5	2.42E5	102.13	32.93	89.07	8.60	4.16E5	2.15E5	93.64	30.04
RBF-F	1×	34.59	5.82	2.97E5	6.73E4	51.78	11.24	37.47	6.44	3.50E5	1.46E5	51.24	11.36
	1.5×	37.05	6.27	4.56E5	1.05E5	87.06	16.53	39.36	5.98	4.77E5	1.44E5	82.59	16.48
	2×	37.56	6.50	5.33E5	1.97E5	116.98	20.66	40.99	6.98	5.53E5	2.23E5	111.52	20.79
RBF-M	1×	43.74	5.13	3.18E5	9.04E4	51.72	11.11	56.02	5.92	3.31E5	1.65E5	42.22	10.92
	1.5×	45.72	5.65	4.80E5	2.17E5	81.71	16.34	54.15	6.43	4.31E5	1.63E5	62.36	16.08
	2×	52.25	5.20	5.63E5	3.63E5	107.61	20.18	55.04	6.93	4.47E5	1.61E5	71.58	18.05
SEA-A	1×	79.97	5.76	8.15E4	7.04E4	33.42	33.07	78.02	6.32	3.06E5	2.69E5	33.15	32.56
	1.5×	86.97	4.21	1.85E5	1.67E5	57.17	49.72	77.59	6.34	3.38E5	2.88E5	27.40	26.25
	2×	79.27	10.96	1.74E5	1.53E5	39.61	34.57	82.58	6.96	3.98E5	3.27E5	40.99	35.14
SEA-G	1×	79.36	5.44	9.21E4	8.03E4	33.66	33.07	76.30	6.35	3.51E5	3.08E5	33.44	30.68
	1.5×	85.78	4.35	2.02E5	1.85E5	54.56	49.69	79.52	6.97	3.78E5	3.18E5	40.18	38.61
	2×	78.73	10.38	1.74E5	1.56E5	36.62	35.00	86.71	4.27	5.15E5	3.89E5	74.24	66.37

records. Secondly, trouble sites of the same order can process different records in parallel as only a subset of trouble sites will be involved in processing any given record.

Table 6 shows the impact of increasing trouble factors with the two most promising variations of HDSM: voting and stacking (results with maximum confidence HDSM demonstrated similar trends, and are excluded for brevity). Trouble factors equal to 1, 1.5, and 2 times the maximum number of features taken across the set of primary sites were used. It can be seen that higher trouble factors generally result in improved accuracy as higher proportions of the less confident set of records are sent to trouble sites for learning and classification. Such records are more likely to be associated with cross-terms and models trained on data record fragments from multiple source sites will thus be more precise than those learned at the primary sites. However, this improved accuracy comes at the expense of increased resource time and data transmission. On the other hand, the effective data transmission cost (captured by the MDTV metric) rarely exceeds the 50% mark. This means that the data transmission cost (as measured by the data volume transmitted across the critical path on the trouble site hierarchy) rarely exceeded more than half of the dataset. We also note that the total data transmission cost as measured by the TDTV metric rarely exceeded the 100% mark. However, this metric does not take into account concurrent streaming of data across different pipelines of the trouble site hierarchy and is thus not a true indicator of transmission cost. Nevertheless it does reflect the bandwidth requirements of the data transmission network.

For a final comparison, Fig. 7 presents the relative ranks in accuracy and resource time of DStack (being the most accurate distributed ensemble) and HDSM voting (having the best trade-off between accuracy and performance for HDSM) averaged across the eleven datasets. While the more accurate HDSM configurations are slower, HDSM voting (TF = 1×) is often able to achieve better accuracy than DStack with better MTBC. Additionally, DStack is further away from the origin point than HDSM voting (TF = 1×) in both charts, showing that it does not represent as good a trade-off between accuracy and resource time. Another crucial point is that HDSM's trouble factor can be varied to achieve an ideal trade-off between accuracy and resource time, whereas the performance of DStack is fixed.

4.4. Suitability for anytime classification

Given that HDSM in one form or another outperformed other approaches, an important question that arises is whether it could be tuned to obtain better performance in terms of the key performance metrics, MTCP and MTBC. One distinguishing characteristic of HDSM is the distributed and layered nature of its classification model. In this context we investigate the distribution of HDSM accuracy over its layers with a view to determining the effectiveness of an early exit in the classification process. We experimented with the FLT, WFR and GAS datasets as well as a synthetic dataset (Synthetic Cross-Term, or SCT), which was generated with known cross-term relationships to assess the effect of layering on accuracy. The GAS dataset was configured with only four features per primary site and a trouble factor equal to the total number of features (the maximum reasonable value), which provided an extreme case with a higher degree of data distribution and transmission to trouble sites. Eq. (15) defines the rule used for generating the SCT dataset with 32 binary features

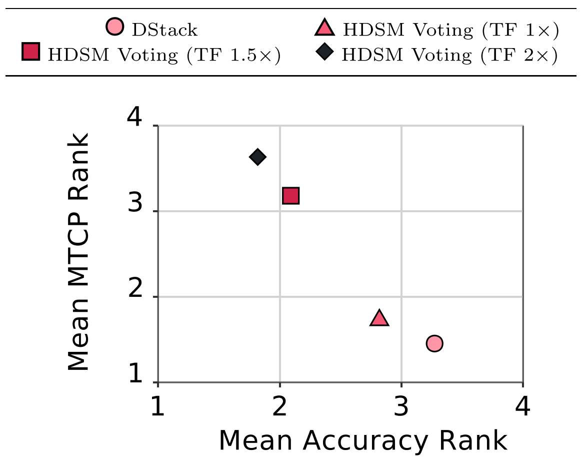

Fig. 7. Accuracy and resource time ranks between HDSM voting and DStack classifiers.

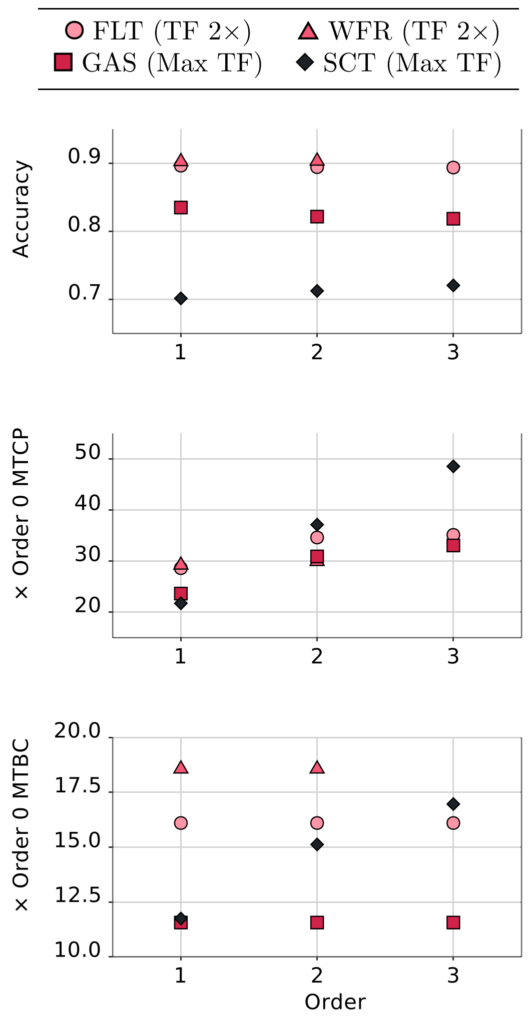

Fig. 8. Effect of anytime classification on HDSM accuracy and resource time.

(distributed sequentially across 8 primary sites) and a binary class. This rule was devised to create complex cross-terms across many primary sites while ensuring a reasonable class balance. The results for other datasets followed the same broad trends, so were omitted in the interests of brevity. Voting was used as the aggregation method for all datasets.

$$\begin{aligned} class = & \big((f_1 \wedge f_5) \vee (f_9 \wedge f_{13}) \vee \\ & (f_{17} \wedge f_{21}) \vee (f_{25} \wedge f_{29}) \big) \ \wedge \\ & \big((f_2 \wedge f_{18}) \vee (f_6 \wedge f_{26}) \vee \\ & (f_{10} \wedge f_{22}) \vee (f_{14} \wedge f_{30}) \big) \end{aligned} \tag{15}$$

Fig. 8 shows how the accuracy and resource time vary with the maximum trouble site order used for classification. MTCP and MTBC are normalized by dividing their values with the corresponding values produced when using the primary sites only (order 0). Order 3 results are not shown for WFR because it never produced sites of this order. The first chart shows that higher-order sites have little or no impact on accuracy; only SCT's accuracy increases because it has relationships across many sites. This suggests that cross-terms generally involved small groups of sites. There is however a significant resource time cost associated with higher-order sites. MTCP increases for all datasets when using higher-order sites, though it tails off for GAS and WFR because few records are transmitted to order 3 sites. On the other hand, MTBC generally does not increase with higher-order sites because few records are transmitted to these sites or because there is parallelism between sites at the same layer. The exception to this is SCT, where many records are transmitted to higher-order sites with little parallelism: all features (for some records) are combined at a single order 3 site. This shows clearly that an early exit has a minimal effect on accuracy whilst substantially boosting the MTCP metric and leaving the MTBC metric either unchanged or improved in value.

The implication is that HDSM can be considered to be an "anytime" classifier in the sense that it can provide a fast and accurate classification from its lower-order trouble sites (typically order 1) without the need for using its entire hierarchy for classification. This short-circuiting process is a direct contributory factor to lowering overheads and reducing the MTCP metric value. This anytime classification property of HDSM makes it even more attractive when compared to approaches such as DStack that require input from every classifier before a final classification outcome can be determined. In effect, HDSM is now able to compete even more favorably to DStack as it maintains its accuracy advantage whilst lowering its computational overheads.

4.5. HDSM parameter sensitivity analysis

In addition to the experiments with different trouble factors, the sensitivity of HDSM to its other parameters was also evaluated. Table 7 presents the results achieved with different parameter values for HDSM with voting on the WFR dataset. While varying each parameter, all other parameters were fixed to their default values (the same values used in the previous

Table 7
Sensitivity analysis of HDSM parameters.

Parameter	Value	Acc%	(SD)	MTCP	MTBC	TDCV	MDCV
Window size-limit	500	75.96	11.43	1.79E5	7.39E4	53.39	20.80
	1000	75.75	11.55	1.81E5	6.85E4	51.10	20.30
	2000	77.19	10.51	1.98E5	7.70E4	51.31	20.82
Threshold time period	100	77.89	10.38	1.94E5	7.46E4	52.69	21.29
	500	75.75	11.55	1.80E5	6.91E4	51.10	20.30
	1000	73.00	14.13	1.54E5	5.64E4	43.64	17.32
Site utilization threshold (T_u)	0.01	75.79	11.51	1.88E5	6.96E4	57.67	20.44
	0.05	75.75	11.55	1.79E5	6.77E4	51.10	20.30
	0.15	69.57	12.58	1.38E5	3.66E4	35.22	17.82
Hoeffding bound δ	1.00E−09	75.76	11.60	1.77E5	6.72E4	50.12	20.15
	1.00E−03	75.75	11.55	1.80E5	6.72E4	51.10	20.30
	1.00E−01	75.76	11.57	1.82E5	6.92E4	51.05	20.34
Smoothing factor (θ)	1	75.75	11.55	1.80E5	6.78E4	51.10	20.30
	5	75.75	11.55	1.77E5	6.61E4	51.10	20.30
	10	75.75	11.55	1.78E5	6.75E4	51.10	20.30

experiments). The trouble factor was fixed to the number of features per primary site.

The results in Table 7 show that the effects of varying these parameters are generally minor. Varying the window size limit has a small impact on accuracy, as this affects exactly which records are selected as trouble records when comparing confidence values to those in the recent window. Decreasing the threshold time period slightly improves accuracy at the cost of extra transmission, as this causes the initial trouble sites to be created slightly sooner. When the site utilization threshold is increased, trouble sites are removed more aggressively when they are infrequently used in classification, resulting in a notable drop in both accuracy and transmission. Finally, the changes in the Hoeffding bound δ and the smoothing factor (θ) demonstrate virtually no impact on the behavior of HDSM, as the δ has a small impact on the thresholds for trouble site creation and removal, and the smoothing factor will only impact the creation of trouble sites with very high proportions of records.

4.6. Limitations and applicability of HDSM

It is worth highlighting the current limitations of HDSM, and the situations where it may not be applicable. It can be seen in Table 8 that for some datasets (such as EEG and RBF), even the most accurate variation of HDSM (of all aggregation method and trouble factor combinations) was not able to achieve accuracy comparable to that of an ARF classifier trained on a copy of the dataset containing all features (as if the data had been centralized, though this is not plausible in real circumstances). The complex relationships in these datasets require a greater degree of data centralization for effective learning. Additionally, if a primary site does not have enough information to reasonably classify the proportion of records that will not be forwarded as trouble, then those records will be classified poorly. The extreme case of this is when a primary site cannot learn any kind of model, and all records are classified with the same, low confidence. In this case, the selection of trouble records based on confidence will be arbitrary. One way to combat these extreme cases is to perform complete centralization of all records at low order trouble sites before selecting subsets of trouble records for higher order trouble sites, which can be achieved by using a high trouble factor (e.g. at least twice the number of features at each primary site). Alternatively, a mechanism could be devised to improve agreement without increasing transmission by selecting trouble records on the basis of both local confidence and the expected confidence at other input sites (produced by monitoring the joint probabilities of feature value combinations among input sites and the confidence values associated with those feature combinations).

Table 8
Comparison of HDSM and centralized accuracy.

Dataset	Best HDSM	Centralized ARF
EEG	66.99	82.07
GAS	94.00	95.95
HIG	53.57	53.57
FLT	89.37	89.74
OCC	90.18	95.65
SDD	98.61	99.52
WFR	90.32	92.82
RBF-F	40.99	68.54
RBF-M	56.02	80.71
SEA-A	86.97	88.61
SEA-G	86.71	87.26

5. Conclusions and future work

In this paper we presented a novel Hierarchical Distributed Stream Miner (HDSM) for mining vertically-distributed data streams. Our experimentation showed that HDSM was able to achieve significant accuracy improvements with minimal data transmission to trouble sites while being competitive on computational resource time with other distributed stream mining approaches. The experiments also demonstrated that HDSM is robust to the presence of concept drift, as it adapts to changing cross-terms that manifest in the streams. Finally, we have shown that HDSM can be used for anytime classification, which improves classification response time with minimal impact on accuracy.

However, there are still many opportunities to improve the accuracy and efficiency of HDSM. Other classification result aggregation methods could boost accuracy, including weighting votes by site accuracy (as measured by sliding windows). Accuracy could also be improved with alternative rules for triggering trouble site creation and removal, such as taking into account the degree to which the agreement-threshold is exceeded when deciding between different candidate trouble sites involving a particular source site. Improved trouble site creation rules could also reduce the inefficient churn that can occur if many trouble sites are created and then removed soon after for lack of utility. To avoid the overhead of re-learning patterns in the case of recurrent concepts, models of removed trouble sites could be retained for later re-use, as in the work of Sakthithasan et al. [45]. Communication efficiency could also be improved in several ways by transmitting data in batches. Furthermore, HDSM is a flexible architecture that could be applied with many other classification algorithms, or even adapted to other machine learning problems (such as numeric regression).

In its current form, HDSM is dependent on the arrival of the true class label for a record within a short period of the record's arrival in order to identify concept drift and adapt the trouble site hierarchy to new concepts, which is a reasonable assumption in various applications. For classifiers that attempt to predict future states or events, subsequent recordings of actual class values can be used to provide online feedback to the classifier. This could be the case when attempting to predict near-future seismic activity from a distributed network of sensors in an earthquake or tsunami early warning system. Alternatively, if a classifier's decisions are used to trigger actions, then it may be possible to measure the subsequent effects of those actions to provide correct class values. As a simple example, a robot attempting to navigate with a binary control (2 class values) based on readings from multiple sensors (similar to the WFR dataset) could measure the remaining distance to its target location in order to determine whether recent control inputs were correct. However, there are also cases when the classifier will only receive infrequent and delayed class labels, such as with the sensorless drive diagnosis (SDD) dataset, where feedback may only be provided from a manual inspection of a drive after it has been classified as defective or has failed.

For applications when class labels are not available within a timeframe reasonable for online supervised learning, it may be possible to use multivariate unsupervised change detection methods [46–48] to infer concept drift directly from predictor variables without reliance on class labels. This could form the basis for un-blacklisting and potentially even decisions to remove trouble sites. However, class labels would still be required in order for the base classifier at each site to learn new concepts, which in turn influences which records are identified as trouble and therefore which trouble sites are created. In some settings, it may be possible to transfer control back to a human when concept drift is detected, allowing the classifier to re-train on the class labels provided by the human's decisions. Consider the scenario of training a self-driving car under controlled conditions. If concept drift is detected (e.g. moving from a sealed to an unsealed road), control could be passed back to a human driver [49], allowing the machine to learn how they respond to the new conditions.

Finally, the challenging issue of privacy preservation is not currently addressed by HDSM. Privacy preserving data mining (PPDM) techniques have been developed to enable knowledge discovery from distributed sources which contain sensitive data that cannot be shared in raw form [50]. It may be possible to apply PPDM principles to data streams so that HDSM may learn from cross-terms without access to raw features.

References

[1] M.H. Rehman, C.S. Liew, T.Y. Wah, M.K. Khan, Towards next-generation heterogeneous mobile data stream mining applications: Opportunities, challenges, and future research directions, J. Netw. Comput. Appl. 79 (2017) 1–24, http://dx.doi.org/10.1016/j.jnca.2016.11.031, URL http://www.sciencedirect.com/science/article/pii/S1084804516302995.

[2] X. Yi, J. Zhang, Z. Wang, T. Li, Y. Zheng, Deep distributed fusion network for air quality prediction, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 965–973.

[3] Q. Huang, C. Huang, J. Huang, H. Fujita, Adaptive resource prefetching with spatial-temporal and topic information for educational cloud storage systems, Knowl.-Based Syst. 181 (2019).

[4] M. Stolpe, Distributed analysis of vertically partitioned sensor measurements under communication constraints, Technical report for Collaborative Research Center SFB 876 Providing Information by Resource-Constrained Data Analysis, 2016, p. 105.

[5] N. Kourtellis, G.D.F. Morales, A. Bifet, A. Murdopo, VHT: Vertical hoeffding tree, in: Big Data (Big Data), 2016 IEEE International Conference on, IEEE, 2016, pp. 915–922.

[6] A.I. Weinberg, M. Last, Interpretable decision-tree induction in a big data parallel framework, Int. J. Appl. Math. Comput. Sci. 27 (4) (2017) 737–748.

[7] A.N. Moghadam, R. Ravanmehr, Multi-agent distributed data mining approach for classifying meteorology data: case study on iran's synoptic weather stations, Int. J. Environ. Sci. Technol. 15 (1) (2018) 149–158.

[8] B.-H. Park, R. Ayyagari, H. Kargupta, A fourier analysis based approach to learning decision trees in a distributed environment, in: Proceedings of the 2001 SIAM International Conference on Data Mining, SIAM, 2001, pp. 1–22.

[9] U. Yun, D. Kim, E. Yoon, H. Fujita, Damped window based high average utility pattern mining over data streams, Knowl.-Based Syst. 144 (2018) 188–205.

[10] U. Yun, G. Lee, E. Yoon, Advanced approach of sliding window based erasable pattern mining with list structure of industrial fields, Inform. Sci. 494 (2019) 37–59.

[11] Z. Liao, Y. Wang, Rival learner algorithm with drift adaptation for online data stream regression, in: Proceedings of the 2018 International Conference on Algorithms, Computing and Artificial Intelligence, ACM, 2018, p. 21.

[12] H.M. Gomes, A. Bifet, J. Read, J.P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, T. Abdessalem, Adaptive random forests for evolving data stream classification, Mach. Learn. 106 (9–10) (2017) 1469–1495.

[13] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 97–106.

[14] B. Krawczyk, L.L. Minku, J. Gama, J. Stefanowski, M. Woźniak, Ensemble learning for data stream analysis: A survey, Inf. Fusion 37 (2017) 132–156.

[15] J.N. Rijn, G. Holmes, B. Pfahringer, J. Vanschoren, The online performance estimation framework: heterogeneous ensemble learning for data streams, Mach. Learn. 107 (1) (2018) 149–176.

[16] J.R.B. Junior, M. do Carmo Nicoletti, An iterative boosting-based ensemble for streaming data classification, Inf. Fusion 45 (2019) 66–78.

[17] M. Tennant, F. Stahl, O. Rana, J.B. Gomes, Scalable real-time classification of data streams with concept drift, Future Gener. Comput. Syst. 75 (2017) 187–199.

[18] R. Chen, K. Sivakumar, H. Kargupta, An approach to online Bayesian learning from multiple data streams, in: Proceedings of Workshop on Mobile and Distributed Data Mining, PKDD, Vol. 1, Citeseer, 2001, pp. 31–45.

[19] A. Akbar, G. Kousiouris, H. Pervaiz, J. Sancho, P. Ta-Shma, F. Carrez, K. Moessner, Real-time probabilistic data fusion for large-scale iot applications, IEEE Access 6 (2018) 10015–10027.

[20] F. Nawaz, N.K. Janjua, O.K. Hussain, Perceptus: Predictive complex event processing and reasoning for iot-enabled supply chain, Knowl.-Based Syst. 180 (2019) 133–146.

[21] N. Domadiya, U.P. Rao, Privacy preserving distributed association rule mining approach on vertically partitioned healthcare data, Procedia Comput. Sci. 148 (2019) 303–312.

[22] C.K. Leung, H. Zhang, J. Souza, W. Lee, Scalable vertical mining for big data analytics of frequent itemsets, in: International Conference on Database and Expert Systems Applications, Springer, 2018, pp. 3–17.

[23] Y. Liu, Z. Xu, C. Li, Distributed online semi-supervised support vector machine, Inform. Sci. 466 (2018) 236–257.

[24] I. Kholod, M. Kuprianov, E. Titkov, A. Shorov, E. Postnikov, I. Mironenko, S. Sokolov, Training normal bayes classifier on distributed data, Procedia Comput. Sci. 150 (2019) 389–396.

[25] K. Tumer, J. Ghosh, Robust Order Statistics Based Ensembles for Distributed Data Mining, Tech. rep., Texas University at Austin, Department of Electrical and Computer Engineering, 2001.

[26] J. Basak, R. Kothari, A classification paradigm for distributed vertically partitioned data, Neural Comput. 16 (7) (2004) 1525–1544.

[27] D.B. Skillicorn, S.M. McConnell, Distributed prediction from vertically partitioned data, J. Parallel Distrib. Comput. 68 (1) (2008) 16–36.

[28] B. Parker, A.M. Mustafa, L. Khan, Novel class detection and feature via a tiered ensemble approach for stream mining, in: Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on, Vol. 1, IEEE, 2012, pp. 1171–1178.

[29] M. Recamonde-Mendoza, A.L. Bazzan, Social choice in distributed classification tasks: Dealing with vertically partitioned data, Inform. Sci. 332 (2016) 56–71.

[30] Y. Li, Z.L. Jiang, L. Yao, X. Wang, S. Yiu, Z. Huang, Outsourced privacy-preserving c4. 5 decision tree algorithm over horizontally and vertically partitioned dataset among multiple parties, Cluster Comput. (2017) 1–13.

[31] M.Z. Omer, H. Gao, N. Mustafa, Privacy-preserving of SVM over vertically partitioned with imputing missing data, Distrib. Parallel Databases 35 (3–4) (2017) 363–382.

[32] T. Li, J. Li, Z. Liu, P. Li, C. Jia, Differentially private naive bayes learning over multiple data sources, Inform. Sci. 444 (2018) 89–104.

[33] F. Khodaparast, M. Sheikhalishahi, H. Haghighi, F. Martinelli, Privacy preserving random decision tree classification over horizontally and vertically partitioned data, in: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), IEEE, 2018, pp. 600–607.

[34] A. Bifet, R. Gavalda, Learning from time-changing data with adaptive windowing, in: Proceedings of the 2007 SIAM International Conference on Data Mining, SIAM, 2007, pp. 443–448.

[35] I.H. Witten, E. Frank, M.A. Hall, C.J. Pal, Moving on : applications and beyond, in: Data Mining : Practical Machine Learning Tools and Techniques., Morgan Kaufmann Publisher, [2016], Cambridge, MA, 2016, pp. 510–511, Ch. 13.

[36] J.M. Chambers, D.A. James, D. Lambert, S. Vander Wiel, et al., Monitoring networked applications with incremental quantile estimation, Statist. Sci. 21 (4) (2006) 463–475.

[37] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive online analysis, J. Mach. Learn. Res. 11 (2010) 1601–1604, URL http://portal.acm.org/citation.cfm?id=1859903.

[38] L. Rutkowski, L. Pietruczuk, P. Duda, M. Jaworski, Decision trees for mining data streams based on the McDiarmid's bound, IEEE Trans. Knowl. Data Eng. 25 (6) (2013) 1272–1279.

[39] P. Matuszyk, G. Krempl, M. Spiliopoulou, Correcting the usage of the hoeffding inequality in stream mining, in: International Symposium on Intelligent Data Analysis, Springer, 2013, pp. 298–309.

[40] E. Ikonomovska, J. Gama, S. Džeroski, Online tree-based ensembles and option trees for regression on evolving data streams, Neurocomputing 150 (2015) 458–470.

[41] W. Street, Y. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 377–382.

[42] D. Dheeru, E. Karra Taniskidou, UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, URL http://archive.ics.uci.edu/ml.

[43] N. Oza, FLTz flight simulator, 2011, URL https://c3.nasa.gov/dashlink/resources/294/.

[44] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (Jan) (2006) 1–30.

[45] S. Sakthithasan, R. Pears, A. Bifet, B. Pfahringer, Use of ensembles of fourier spectra in capturing recurrent concepts in data streams, in: Neural Networks (IJCNN), 2015 International Joint Conference on, IEEE, 2015, pp. 1–8.

[46] G. Boracchi, D. Carrera, C. Cervellera, D. Maccio, QuantTree: Histograms for change detection in multivariate data streams, in: International Conference on Machine Learning, 2018, pp. 638–647.

[47] L.I. Kuncheva, Change detection in streaming multivariate data using likelihood detectors, IEEE Trans. Knowl. Data Eng. 25 (5) (2013) 1175–1180.

[48] X. Song, M. Wu, C. Jermaine, S. Ranka, Statistical change detection for multi-dimensional data, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2007, pp. 667–676.

[49] N. Merat, A.H. Jamson, F.C. Lai, M. Daly, O.M. Carsten, Transition to manual: Driver behaviour when resuming control from a highly automated vehicle, Transp. Res. F 27 (2014) 274–282.

[50] J. Vaidya, A survey of privacy-preserving methods across vertically partitioned data, in: Privacy-Preserving Data Mining, Springer, 2008, pp. 337–358.