E-Tree: An Efficient Indexing Structure for Ensemble Models on Data Streams

Peng Zhang, Chuan Zhou, Peng Wang, Byron J. Gao, Xingquan Zhu, and Li Guo

Abstract—Ensemble learning is a common tool for data stream classification, mainly because of its inherent advantages of handling large volumes of stream data and concept drifting. Previous studies, to date, have been primarily focused on building accurate ensemble models from stream data. However, a linear scan of a large number of base classifiers in the ensemble during prediction incurs significant costs in response time, preventing ensemble learning from being practical for many real-world time-critical data stream applications, such as Web traffic stream monitoring, spam detection, and intrusion detection. In these applications, data streams usually arrive at a speed of GB/second, and it is necessary to classify each stream record in a timely manner. To address this problem, we propose a novel *Ensemble-tree* (E-tree for short) indexing structure to organize all base classifiers in an ensemble for fast prediction. On one hand, E-trees treat ensembles as spatial databases and employ an *R-tree* like height-balanced structure to reduce the expected prediction time from linear to sub-linear complexity. On the other hand, E-trees can be automatically updated by continuously integrating new classifiers and discarding outdated ones, well adapting to new trends and patterns underneath data streams. Theoretical analysis and empirical studies on both synthetic and real-world data streams demonstrate the performance of our approach.

Index Terms—Stream data mining, classification, ensemble learning, spatial indexing, concept drifting

1 Introduction

Data stream classification represents one of the most important tasks in data stream mining [1], [2], which has been popularly used in real-time intrusion detection, spam filtering, and malicious website monitoring. In the applications, data arrive continuously in a stream fashion, timely predictions in identifying malicious records are of essential importance.

Compared to traditional classification, data stream classification is facing two extra challenges: large/increasing data volumes and drifting/evolving concepts [3], [4]. To address these challenges, many ensemble-based models have been proposed recently, including weighted classifier ensembles [5], [6], [7], [8], [9], incremental classifier ensembles [10], classifier and cluster ensembles [11], to name a few. While these models vary from one to another, they share striking similarity in their design: *using divide-and-conquer techniques to handle large volumes of stream data with concept drifting*. Specifically, these ensemble models partition continuous stream data into small data chunks, build one or multiple light-weight base classifier(s) from each chunk, and combine base classifiers in different ways for prediction. Such an ensemble learning design enjoys a number of advantages such as scaling well, adapting quickly to new concepts, low variance errors, and ease of parallelization. As a result, ensemble has become one of the most popular techniques in data stream classification.

To date, existing works on ensemble learning in data streams mainly focus on building accurate ensemble models. Prediction efficiency has not been concerned mainly because (1) prediction typically takes linear time, which is sufficient for general applications with undemanding prediction efficiency. (2) Existing works only consider combining a small number of base classifiers, e.g., no more than 30 [5], [11], [12]. However, there are increasingly more real-world applications where stream data arrive intensively in large volumes. In addition, the hidden patterns underneath data streams may change continuously, which requires a large number of base classifiers to capture various patterns and form a quality ensemble. Such applications call for fast sub-linear prediction solutions.

Motivating example. In online webpage stream monitoring, ensemble learning can be used to identify malicious pages from normal pages, both arriving continuously, in real time. We deployed a detection system on a Linux machine with 3 GHz cpu and 2 GB memory. Each day, a batch of base classifiers are trained using decision trees (C4.5 algorithm [13]) with all base classifiers being combined to classify pages in the next day. In our experiments, in total 120 days of stream data [14] were used.

The curve in Fig. 1 summarizes our experimental results, showing the typical linear relationship between prediction time and ensemble size. For example, it takes an ensemble with 50 members 0.083 second to classify a page. Suppose that the webpage stream flows 10,000 pages per second,

- *P. Zhang, C. Zhou, P. Wang and L. Guo are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China. E-mail: {zhangpeng, zhouchuan, wangpeng, guoli}@iie.ac.cn.*
- *B.J. Gao is with the Department of Computer Science, Texas State University, 601 University Drive, San Marcos, TX 78666. E-mail: bgao@txstate.edu.*
- *X. Zhu is with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431. E-mail: xqzhu@cse.fau.edu.*

Manuscript received 30 Apr. 2013; revised 3 Dec. 2013; accepted 4 Dec. 2013. Date of publication 9 Jan. 2014; date of current version 23 Dec. 2014. Recommended for acceptance by G. Cormode.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2298018

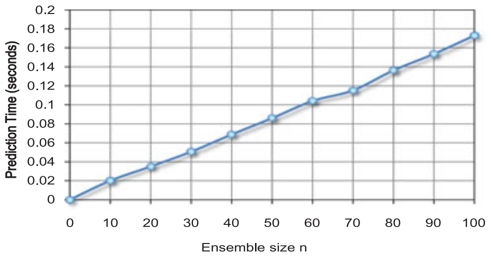


Fig. 1. Prediction time versus ensemble size.

which is common for backbone networks. Then, monitoring all stream records requires a sophisticated parallel computing architecture having 830 processors!

To achieve sub-linear time for prediction, a practical approach is to explore shared patterns among all base classifiers. Without loss of generality, suppose that base classifiers of an ensemble are built using decision trees. Then, each base classifier is comprised of a batch of *decision rules*. Each decision rule covers a rectangular area in the decision space, and can be considered as a *spatial object* in the decision space. This way, each base classifier is converted to a batch of spatial objects, and an ensemble model is converted to a *spatial database* as shown in Fig. 2. By doing so, the problem can be reduced to exploring shared patterns among all spatial objects (base classifiers) in the spatial database (ensemble model).

Challenges. Spatial databases have been extensively studied in the past several decades. Many spatial indexing structures exist that utilize shared patterns among spatial data to reduce query and update costs [15]. Examples include *R-tree* [16], *R*-tree* [17], *R+-tree* [18], *M-tree* [19], to name a few [20]. Generally, these methods can be used to index ensemble models. However, compared to traditional spatial databases, indexing ensemble models has its unique characteristics that pose non-trivial technical challenges:

- *Complex indexing objects.* Existing spatial indexing methods are designed for *conventional spatial data* such as image, map, and multimedia data. The indexing objects for ensemble models are *decision rules* with much richer information, including class labels, class probability distribution, and weights of classifiers.
- *Different objectives.* Conventional spatial indexing aims at fast retrieval and updating of spatial data. Ensemble model indexing aims at fast prediction of incoming stream records.
- *Changing data distributions.* Due to concept drifting, hidden patterns underneath stream data change continuously. Therefore a decision region in ensemble models may be associated with different class labels.

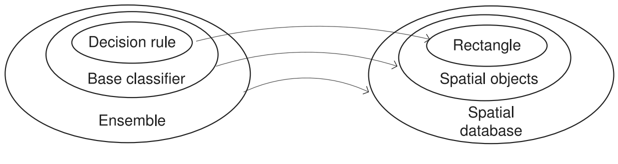


Fig. 2. Mapping an ensemble model to a spatial database.

TABLE 1
List of Symbols

Symbol	Description
E	classifier ensemble
C_i	a base classifier in ensemble E
N	number of decision rules (rectangular objects)
Θ	rectangle bounding of a decision rule
M	node capacity
m	minimal entries in a node
R_{ij}	the jth decision rule of the ith base classifier in ensemble
h	E-tree height
L_{ij}	average length on the j-th axis of a level-i node
$I.l$	length of an interval I
$I.s(e)$	starting (ending) point of an interval I
c	concept drifting rate
γ	threshold of class label decision
Λ	number of equal partitions on the range $[0, \lambda]$
μ, η	boundaries of two nodes incurred by insertion
M_η	the largest ending point of the first u interval
$E_{\mu\eta}$	probability of an interval equals $[\mu\frac{\lambda}{\Lambda}, \eta\frac{\lambda}{\Lambda}]$
$E^u_{\mu\eta}$	probability of the u-th interval equals $[\mu\frac{\lambda}{\Lambda}, \eta\frac{\lambda}{\Lambda}]$
$E^u_{\mu\eta(\leq\xi)}$	probability that both $E^u_{\mu\eta}$ and $M_\eta \leq \xi\frac{\lambda}{\Lambda}$ stand

Our solution. In light of these challenges, in this paper we propose a novel *Ensemble-tree* (E-tree for short) structure that organizes base classifiers in a height-balanced tree structure to achieve logarithmic time complexity for prediction. Technically, an E-tree has three key operations: (1) Search: traverse an E-tree to classify an incoming stream record x; (2) Insertion: Integrate new classifiers into an E-tree; (3) Deletion: Remove outdated classifiers from an E-tree. As a result, the E-tree approach not only guarantees a logarithmic time complexity for prediction, but is also able to adapt to new trends and patterns in stream data.

The rest of the paper is structured as follows. Section 2 introduces the ensemble indexing problem. Section 3 describes the main structure and key operations of E-trees. Section 4 studies the theoretical aspects of E-trees. Section 5 reports experiments. Section 6 surveys related work, and we conclude the paper in Section 7. Important symbols used in the paper are summarized in Table 1.

2 Problem Description

Consider a two-class data stream S consisting of an infinite number of records $\{(x_i, y_i)\}$, where $x_i \in \mathbb{R}^d$ is a d-dimensional attribute vector, and $y_i \in \{normal, abnormal\}$ is the class label, which is unobservable unless the sample is properly labeled. Suppose that we have built n base classifiers $C_1, C_2, \ldots, C_n$ from historical stream data using a decision tree algorithm (such as C4.5). All the n base classifies are combined together as an ensemble classifier E. Each base classifier C_i $(1 \leq i \leq n)$ is comprised of l decision rules R_{ij} $(1 \leq j \leq l)$ represented by conjunction literals (i.e., rules are expressed as "$if \ldots then \ldots$"). Then, there are $N = n \times l$ decision rules in the ensemble E. The *aim* of this paper is to generate accurate prediction for an incoming stream record x, using the ensemble model E, with sub-linear time complexity $O(log(N))$.

In order to achieve this goal, we first convert each base classifier C_i $(1 \leq i \leq n)$ into a batch of spatial objects o_{i_j} $(1 \leq j \leq l)$. This way, the ensemble model E is converted to a spatial database A_E containing all spatial objects o_{i_j} $(1 \leq i \leq n, 1 \leq j \leq l)$. As a result, the original problem is

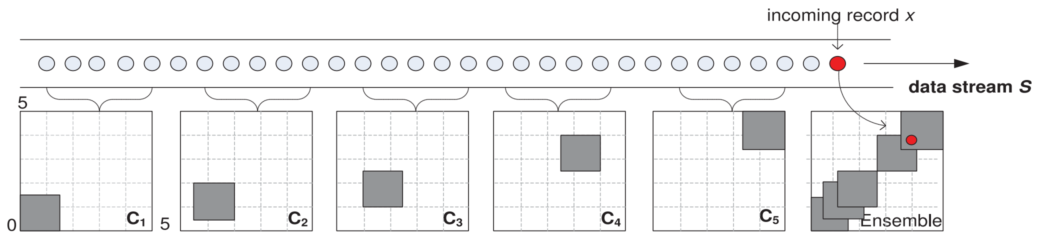


Fig. 3. Mapping ensemble E in Example 1 to spatial database A_E.

reduced to *classifying each incoming stream record x by searching over the spatial database A_E*.

In the following, we use Example 1 to illustrate the mapping from the ensemble model E to the spatial database A_E. This example will be used throughout the paper.

Example 1. Consider a Web monitoring stream S with two classes (normal versus abnormal). Suppose that each stream record has two attributes r_1, r_2, where $r_i \in [0, 5]$, and the most recent five base classifiers $C_1, C_2, \ldots, C_5$ built along the stream are incorporated in the ensemble model E, with decision rules for the base classifiers listed in Table 2. For simplicity, each classifier contains one literal clause (in an if-$then$ expression) corresponding to the target abnormal class. The goal is to classify each incoming record x using the ensemble E.

Now we demonstrate the conversion of the ensemble model E to a spatial database A_E. As shown in Fig. 3, the whole decision space is a two-dimensional rectangle $A = (0, 0, 5, 5)$. For each base classifier, a small gray rectangle is used to represent its decision rule for the target abnormal class. Besides, due to concept drifting, the gray rectangles associated with the five classifiers drift from the bottom left corner to the upper right corner in the decision space A. By doing so, the ensemble model can be represented by a batch of spatial objects (i.e., the gray rectangles) generated from all base classifiers. These spatial objects constitute the spatial database A_E. Given a small circle $x = (4, 4)$ as an incoming record, we want to classify x as accurate and as fast as possible by searching over A_E.

3 E-Tree Indexing Structure

In this section, we introduce the E-tree data structure and its three key operations, *Search*, *Insertion*, and *Deletion* for maintaining E-trees.

3.1 Basic Structure of E-Trees

An E-tree, as shown in Fig. 4, is a height-balanced tree mainly consisting of two components: an *R-tree* like structure T on the right-hand side storing all decision rules of the ensemble, and a *table structure* on the left-hand side storing all classifier-level information of the ensemble, such as IDs and weights of classifiers. The two structures are connected together by linking each classifier in the table to its corresponding decision rules in the tree.

The tree structure consists of two different types of nodes: leaf nodes and pivot nodes. Similar to R-trees, each leaf node contains a batch of decision rules that are located in a heavily overlapping area in the decision space. Each decision rule can be represented as a special type of spatial object of the following form:

$$(\Theta, classifier\_id, sibling), \tag{1}$$

where Θ denotes a rectangle bounding the decision rule, $classifier\_id$ denotes the classifier generating the decision rule, and $sibling$ denotes the memory address of the next decision rule generated by the same classifier. Generally, a decision rule covers a closed space $\Theta = (\Theta_0, \Theta_1, \ldots, \Theta_d)$ with each Θ_i representing a closed bounded interval along dimension i. On the other hand, a pivot node in the tree structure contains entries in the form of

$$(\Theta, child\_pointer), \tag{2}$$

where Θ is the smallest rectangle covering all decision rules in its child nodes, and $child\_pointer$ references its child node. A non-root node contains between m and M entries. The root node has at least two entries except a leaf node.

The table structure contains information about all base classifiers in an ensemble with each entry denoted by

$$(classifier\_id, weight, pointer), \tag{3}$$

where the first item represents the classifier ID in the ensemble, the second item represents the classifier's weight, and the last item denotes the memory address of its first component decision rule in the tree structure.

It is worth noting that while R-trees index conventional spatial objects such as image, map, and multimedia data, E-trees index a new type of spatial data, decision rules, with the following constraints:

- All decision rules are built in a continuous attribute space. In case that a discrete attribute is observed, we can convert it to a continuous attribute by severely penalizing its difference. For example, if a binary attribute r_3 $(True, False)$ appears in Example 1, and the decision rule in C_1 is "*if* $(r_1 \le 1.5) \wedge (r_2 \le 1.5)$ $\wedge (r_3 = True)$ *then abnormal; otherwise normal*", the decision rule in C_2 is "*if* $(0.5 \le r_1 \le 2) \wedge (0.5 \le r_2 \le 2) \wedge (r_3 = False)$ *then abnormal; otherwise normal*",

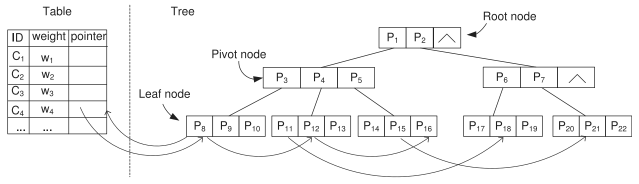


Fig. 4. An illustration of E-tree (some links are omitted).

TABLE 2
The Five Base Classifiers in Example 1

ID	Decision Rules
C_1	if $(r_1 \leq 1.5) \wedge (r_2 \leq 1.5)$ then abnormal; otherwise normal
C_2	if $(0.5 \leq r_1 \leq 2) \wedge (0.5 \leq r_2 \leq 2)$ then abnormal; otherwise normal
C_3	if $(1 \leq r_1 \leq 2.5) \wedge (1 \leq r_2 \leq 2.5)$ then abnormal; otherwise normal
C_4	if $(2.5 \leq r_1 \leq 4) \wedge (2.5 \leq r_2 \leq 4)$ then abnormal; otherwise normal
C_5	if $(3.5 \leq r_1 \leq 5) \wedge (3.5 \leq r_2 \leq 5)$ then abnormal; otherwise normal

then the first decision rule on abnormal class will be $(0, 0, 0, 1.5, 1.5, 0)$, while the second one will be $(0.5, 0.5, 1, 2, 2, 1)$. However, any change of r_3 will be heavily penalized by assigning a much heavier weight.

- Each decision rule covers a closed space. In case a decision rule covers only a partially-closed space, a lower or upper bound of the decision space will be added to make it closed. For example, in Example 1, decision rule R_{11} from classifier C_1 is a half-closed rule $(r_1 \leq 1.5) \wedge (r_2 \leq 1.5)$, then the lower bound $r_1 = 0, r_2 = 0$ will be added to make it a closed space $(0 \leq r_1 \leq 1.5) \wedge (0 \leq r_2 \leq 1.5)$. Moreover, if a decision rule covers only k $(k < d)$ dimensions, it will be expanded over the remaining $(d - k)$ dimensions. For example, for a decision rule defined in partially closed space $\mathcal{R} = (0 \leq r_1 \leq 1.5)$, we will expand $\mathcal{R}$ to a closed space as $\mathcal{R}' = (0 \leq r_1 \leq 1.5) \wedge (0 \leq r_2 \leq 5)$.
- All decision rules will make "hard" decisions. For example, a Web page has 100 percent chance of being malicious or not. Therefore, there are no explicit items in Eqs. (1) and (2) to define the posterior class distributions.
- We only consider binary classification at this time. The extension to multi-class problems are further addressed in Section 3.6. We only index decision rules from one class (the minor class containing fewer decision rules). For multi-class, an intuitive method is to combine multiple E-trees by using a one-against-one/ one-against-all strategy.

Example 2. Fig. 5 shows the E-tree structure for the ensemble model used in Example 1. For simplicity, some links from leaf nodes to the table structure are omitted.

3.2 Search Operation

Each time a new record x arrives, a search operation is invoked to predict a class label for x. The algorithm first traverses the E-tree and finds decision rules in the leaf node(s) covering record x. Then it calculates the class label for x by combining decisions from all the retrieved rules,

$$y_x = sgn\left(\sum\nolimits_{i=1}^{u} w_i P(C_{index}|x),\ \gamma\right), \qquad (4)$$

where y_x denotes the class label of x, u is the total number of retrieved decision rules covering x, $sgn(a, \gamma)$ is a threshold function that decides x's class label by comparing a and γ, and C_{index} is the indexing class in the tree. Recall that only the minor class with fewer decision rules is indexed in E-trees. For instance, in Example 1, C_{index} represents the abnormal class. $P(C_{index}|x)$ is a "hard" posterior probability of either 0 or 1.

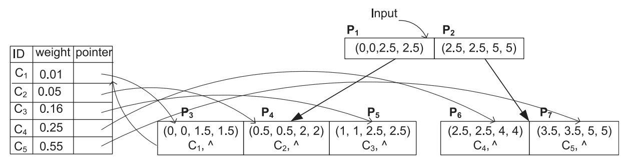


Fig. 5. E-tree for the ensemble model in Example 1.

Algorithm 1 lists detailed procedures of the search operation. The algorithm performs a depth-first search over the tree. To derive a class label for x, it first traverses along the branches whose rectangles cover x, and then calculates the class label for x using Eq. (4). We use Example 3 derived from Example 1 to explain the search process.

Algorithm 1: Search

```
Input  : E-tree $T$, stream record $x$, parameter $\gamma$
Output: $x$'s class label $y_x$
Initialize(stack) ; // initialize a stack $U \leftarrow \emptyset$ ; // $U$
records all rules covering $x$ $P \leftarrow T.tree.root$ ;
// get the root of the tree
foreach entry $R \in T$ do
    push(stack, $R$) ;
while stack $\neq \emptyset$ do
    $e \leftarrow pop(stack)$ ;
    if  e is an entry of a leaf then
        $U \leftarrow U \cup e$ ;
    else
        $P \leftarrow e.child$ ;
        foreach entry $e \in P$ do
            if $x \in e$ then
                push (stack, $e$) ;
foreach entry $e \in U$ do
    find its weights in the table structure ;
$y_x \leftarrow$ Call Eq. (4) to calculate $x$'s class label ;
Output $y_x$ ;
```

Example 3. Suppose $x = (4, 4)$ is an incoming record as shown in Example 1, and the parameter γ in Eq. (4) is 0.5. The search algorithm first compares x with entries P_1 and P_2 in the root node, and then descends along entry P_2 that covers x. After that, it finds that both entries (P_6 and P_7) in P_2's child cover x. Next, it obtains the weights for entries P_6 and P_7 in the table structure through their $classifier\_id$ entries. Based on these results, the probability of x belonging to the abnormal class is

$$w_4 P(abnormal|x) + w_5 P(abnormal|x) = 0.75 > 0.5.$$

Therefore, x is predicted as abnormal. At the worst case, a maximum of four comparisons are needed (P_1, P_2, P_6, and P_7) in this example. Compared to a linear scan that needs five comparisons, E-tree achieves a 20 percent improvement.

3.3 Insertion Operation

Insertions are used to integrate new base classifiers into the ensemble model, so that the ensemble model can adapt to new trends and patterns in data streams. Algorithm 2 lists detailed procedures of the insertion operation. When a new classifier C arrives, a new entry associated with C is added into the table structure. Meanwhile, its decision rules R are inserted into the tree structure one by one, and linked together by their pointer entries.

Algorithm 2: Insertion

```
Input  : E-tree $T$, classifier $C$, parameters $m$, $M$
Output: Updated E-tree $T'$
$P \leftarrow T.tree.root$ ;          // get the root of the tree
foreach decision rule $R \in C$ do
    $L \leftarrow searchLeaf(R, P)$ ;       // $L$ contains $R$ ;
    if $L.size < m$ then
        $L \leftarrow L \cup R$ ;
        $T' \leftarrow updateParentNode(L)$ ;
    else
        $< P_L, P_R > \leftarrow splitNode(L, R)$ ;
        $T' \leftarrow adjustTree(T, P_L, P_R)$ ;
Insert $C$ into the table structure ;
Output $T'$ ;
```

Inserting R into the tree structure is similar to the insertion operation in R-trees. First, a *searchLeaf(R, T)* function is used to find a leaf node to insert each decision rule R. This function searches from the root node, and traverses the branches covering R. Once a leaf node (e.g., P_L) is found, the algorithm will investigate whether node P_L has spare room for inserting R. If it contains less than M entries, R will be successfully inserted, and a function *updateParentNode(P_L)* will be invoked to revise the corresponding entry in the parent node to cover the minimum bounding rectangle (MBR) of the new node. On the other hand, if node P_L is full, a *splitNode(P_L)* function will be invoked to split the current leaf node.

The most critical step in the insertion operation is node splitting. Similar to R-trees, in principle *the total area of the two covering rectangles after a split should be minimized*. Compared to existing methods, node splitting in E-trees faces a new challenge which is that a decision rule may contain discrete attributes that cannot be changed during area enlargement. To solve this problem, a much heavier weight is assigned to each discrete attribute during node splitting to avoid enlargement or shrinkage. Formally, when a new entry R is added into node P incurring a split, the following objective function should be optimized:

$$< P_L, P_R > = \underset{< X,Y > | X,Y \in R \cup P}{argmin} \quad (X.\Theta + Y.\Theta), \tag{5}$$

where X and Y are variables, and P_L and P_R are the new nodes containing R and all entries in P. We want to separate the entries in $R \cup P$ into two classes with the total area of the two covering rectangles being minimized. Obviously, this is far from triviality. The above optimization problem can be formally written as in

$$S^* = \arg\min_{S} \ \{f(S) := Area(S) + Aera(\overline{S})\}, \tag{6}$$

where S is a nontrivial subset and $S \subset R \cup P$ (i.e., $S \neq \emptyset$ and $R \cup P$), $Area(S)$ is the area of the minimal rectangle that covers S, and $\overline{S}$ is the complementary set of S. The optimum solutions should satisfy $P_L = S^*$ and $P_R = \overline{S^*}$.

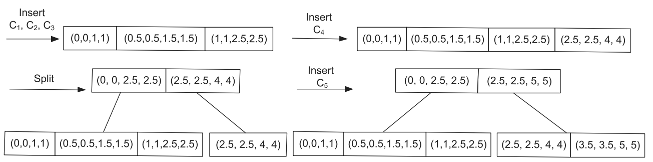


Fig. 6. Inserting the five classifiers in Example 1 to E-tree.

Lemma 1. *The set function $f(S)$ in Eq. (6) is symmetric submodular.*

Proof. See Appendix A.1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2014.2298018. □

The symmetric submodular property guarantees that the convexity of the discrete set function in Eq. (6) [21], and minimizing a convex function can be achieved in $O(M^3)$ time. Notice that if M is very large, the problem would be very expensive to solve. As a solution, we make use of a greedy heuristic that first randomly select two entries in $R \cup P$ with the largest distance, and then cluster all the remaining $m - 1$ entries in $R \cup P$ into the two classes.

In the following we use Example 4 to explain the insertion and node splitting of the five classifiers in Example 1. This process is also illustrated in Fig. 6. Due to page limitations, we omit the table structure of the E-tree.

Example 4. We insert five classifiers in Example 1 one by one. Suppose parameters $M = 3$ and $m = \lfloor \frac{M}{2} \rfloor = 1$. The first three classifiers can be inserted by simply adding them to the root node. Upon inserting C_4, the number of entries in the root node is four, which exceeds M. Thus, the greedy splitting algorithm with initial points $(0, 0, 1, 1)$ and $(2.5, 2.5, 4, 4)$ will be invoked to partition the root node into two leaf nodes. Besides, a new root node with two entries is generated to index the two leaf nodes. In the end, C_5 is inserted into the right leaf node directly as the node size is smaller than M.

3.4 Deletion Operation

The deletion operation discards outdated classifiers when the E-tree reaches its capacity. For example, if the largest classifier size is set to four in Example 1, C_1 will be discarded from the E-tree when C_5 arrives.

Possibly two different deletion methods can be adopted. The first one resembles deletions in B-trees. It merges the under-full nodes to one of the siblings resulting in the least area increase. The second one resembles deletions in R-trees and performs *delete-then-insert*. It first deletes the under-full node, then inserts the remaining entries into the tree using the insertion operation.

The second method is advantageous in that: (1) It is easy to implement; and (2) Re-insertion will incrementally refine the spatial structure of the tree. In addition, it has been shown [16] that the B-tree method may cause excessive node splits. Therefore, we use the second method for our deletion operation. Specifically, if a classifier C needs to be deleted from the E-tree, we first find its classifier ID in the table structure using function *searchClassifier(C, A)*, and then delete its component

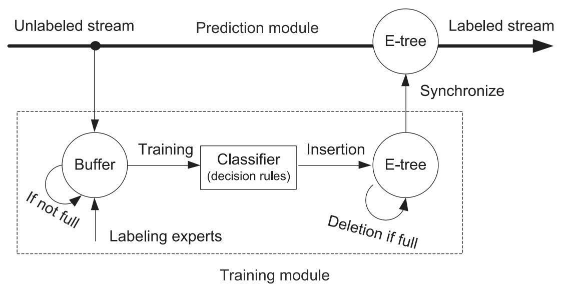


Fig. 7. Architecture of ensemble learning with E-trees.

decision rules by traversing the pointer entries. After each deletion, if a leaf node has less than m entries, the node will be deleted and re-inserted into the E-tree. The update will propagate to the upper level unless the $[m, M]$ condition is met. Algorithm 3 gives the details of the deletion operation.

```
Algorithm 3: Deletion
  Input  : E-tree $T$, classifier $C$, parameters $m$, $M$
  Output: Updated E-tree $T'$
  $P \leftarrow T.tree.root$ ;          // get the root of the tree
  $A \leftarrow T.table.ref$ ;          // get the reference
  $R \leftarrow searchClassifier(C, A)$ ;          // classifier $C$
  $P \leftarrow R.pointer$ ;
  while $P \neq \emptyset$ do
      $L \leftarrow P.node()$ ; // leaf node $L$ contains rule $P$
      $q \leftarrow P.sibling$ ;
      $L' \leftarrow deleteEntry(L, P)$ ;
      if $L'.size < m$ then
          $T' \leftarrow deleteNode(T, L')$ ;          // deletion ;
          foreach entry $e \in L'$ do
              $T' \leftarrow insertRule(e, T)$ ;
      $P \leftarrow q$ ;
  Output $T'$ ;
```

3.5 Ensemble Learning with E-Trees

Fig. 7 shows the architecture of ensemble learning with E-trees on data streams. The training module maintains an E-tree that is constantly updated by calling the *insertion* and *deletion* operations. The prediction module contains a synchronized copy of the E-tree to make online predictions by calling the *search* operation.

For each incoming stream record, on one hand, the prediction module will call the search operation to predict its class label. On the other hand, the record will be stored in a buffer of the training module, where it will be labeled by experts (either human experts or intelligent labeling machines). Note that the labeling process is time-consuming, and only a small portion of incoming records can be labeled. In order to provide uniform labeling, a practical approach is to set a sampling frequency parameter that matches the labeling speed. Once the buffer is full, all labeled records will be used to build a new classifier, which is inserted into the E-tree by calling the insertion operation. In case the E-tree is full, an outdated classifier will be deleted by calling the deletion operation. The updated E-tree will be synchronized to the E-tree copy in the prediction module.

3.6 E-Forests

E-trees aim to solve binary classification problems. In this subsection, we propose E-forests which combine multiple E-trees for multi-class classification.

E-forests are inspired by the one-against-all structure used in multi-class support vector machines (SVMs). In an E-forest, each component E-tree is associated with one single class, and a prediction operation sequentially goes through all component trees in the forest. The best-case scenario would be that the first component tree returns a positive answer and acquires the class label. As a result, the prediction takes the same amount of time as a single tree. In the worst case, the prediction has to go through the whole forest, which demands extra time that may not meet the needs of online classification.

To avoid worst-case scenarios, a practical solution is to adaptively order the component trees in the forest to comply with data distributions in streams. Ordering of trees on streams can be induced to the problem of *pipelined filter ordering on streams* [22], where various greedy and randomized algorithms can be used for fast computation.

>Algorithm 4 lists the E-forest label search algorithm. First, we combine all component E-trees in the forest. Then, for each incoming record x, the algorithm goes through the forest, until the search result $prob(x)$ meets the given threshold γ. The algorithm terminates with the ultimate class label for record x.

```
Algorithm 4: E-forests
  Input  : E-trees $T_1 \cdots T_n$, stream record $x$, parameter $\gamma$
  Output: $x$'s class label $y_x$
  E-forest $\leftarrow$ Initialize($T_1, \ldots, T_n$) ;
  $y_x \leftarrow NULL$ ;
  while $E\text{-}forest \neq \emptyset$ do
      $Tree_i \leftarrow$ obtainTree(E-forest) ;
      $prob(x) \leftarrow$ Search($Tree_i, x$ ) ;
      if $prob(x) \geq \gamma$ then
          $y_x \leftarrow C_i$, break; ;
      else
          E-forest $\leftarrow$ E-forest $\setminus$ $Tree_i$ ;
  Output $y_x$ ;
```

3.7 Why Ensemble Classifiers Work

We now theoretically prove that ensemble classifiers have a lower expected error rate than a single classifier on data streams with dynamically changing/evolving data distributions (i.e., concept drifting).

Assume that we have built a batch of classifiers C_k $\{k = 1, \ldots, n\}$ from historical stream data, and we are observing an incoming test example (x, y). Let $Error_{ensemble}$ be the expected error of ensemble classifier, and $Error_{random}$ be the expected error of a random single classifier, then we have Theorem 1 as follows:

Theorem 1. *The expected error rate of ensemble classifier is smaller than that of a random single classifier, i.e.,* $Error_{ensemble} \leq Error_{random}$.

Proof. See Appendix A.2, available in the online supplemental material. □

Theorem 1 follows the fact that ensemble classifiers always exhibit smaller variance error than a random single classifier, especially in unstable learning environments.

4 Theoretical Analysis

In this section, we present theoretical studies on the properties and performance of E-trees.

4.1 Online Label Search

When classifying a stream record x using an E-tree, the ideal scenario is that all target decision rules covering x are located in one single leaf node. This way, the search algorithm only needs to traverse one path down the tree to make the final prediction for x. In the worst scenario, if the target decision rules span all leaf nodes, the search algorithm will have to traverse all possible paths in order to classify x. In light of this observation, the key to analyzing E-tree's performance is to estimate the probability that x's target decision rules are located in one single leaf node.

Assume decision rules are mutually independent (dependency between rules will only improve E-tree's performance as will be discussed shortly), the probability that two rules are located in the same leaf node is proportional to the overlapping region between two hyper-rectangles representing the two rules. Thus, the problem becomes that *given a set of decision rules, how to estimate the size of their overlapping region*?

To answer this question, consider two decision rules R_a and R_b in the label search space $S = S_1 \times \cdots \times S_d$, where $S_i \in [0, s_i]$, $1 \le i \le d$. Since not all rules cover all the d-dimensions, rules R_a and R_b may overlap in some dimensions. Without loss of generality, assume rule R_a is defined in r_a dimensions $S_1 \times S_2 \times \cdots \times S_{r_a}$, with the i^{th} $(1 \le i \le r_a)$ dimension S_i spanning in a_i region (out of the whole domain $[0, s_i]$), and rule R_b is defined in r_b dimensions $S_{1'} \times S_{2'} \times \cdots \times S_{r_b}$, with the j^{th} $(1 \le j \le r_b)$ dimension S_j spanning in b_j region. Due to temporal correlations of data streams, it is often the case that R_a and R_b overlap on r $(r < r_a, r < r_b)$ dimensions. Then, we have Theorem 2:

Theorem 2. *Given two decision rules R_a and R_b, the probability that they are located in the same leaf node is*

$$\varphi(R_a, R_b) \propto \prod_{i=1}^{r} \frac{a_i \cdot b_i}{s_i \cdot s_i} \prod_{j=r+1}^{r_a} \frac{a_j}{s_j} \prod_{k=r+1}^{r_b} \frac{b_k}{s_k}. \tag{7}$$

Proof. See Appendix A.3, available in the online supplemental material. □

From Eq. (7), we can infer that the higher the data dimensionality, the less possibility the two decision rules will be located in the same node. Specifically, we have Lemma 2 as follows:

Lemma 2. *Given decision rules R_a, R_b and R'_a, R'_b, let the data dimensionality $r'_a \ge r_a$ and $r'_b \ge r_b$, then $\varphi(R'_a, R'_b) \le \varphi(R_a, R_b)$.*

Proof. See Appendix A.4, available in the online supplemental material. □

Lemma 3. *Consider n decision rules, assume the number of data dimension increases by $\Delta(r)$, then the increased query time ΔT is bounded by $\Delta T \in [\rho(t)^{\Delta(r)}, \rho(t)^{n\Delta(r)}]$, where $\rho(t)$ is the average query time in one dimension. Moreover, let ξ be the overlapping rate between $\Delta(r)$ attributes, then the increased query cost is $\rho(t)^{n(1-\xi)\Delta(r)+\xi\Delta(r)}$.*

Proof. See Appendix A.5, available in the online supplemental material. □

Lemma 3 describes the relationship between query cost and data dimensionality. The query cost increase depends on the attribute overlapping rate. In the worst case, the data are sparsely distributed in a disjoint high-dimensional space, then the query cost increases exponentially with the increase of data dimensionality. In the best case, the data are densely distributed in a low-dimensional subspace, then the query cost increases linearly with respect to the increase of the dimensionality.

Based on the above analysis, we study important issues on the number of comparisons required for classifying x under two extreme conditions: (1) All target decision rules are located in the same leaf node; and (2) Target decision rules are uniformly distributed across all leaf nodes.

Formally, consider an ensemble model E containing n classifiers, with each classifier having l decision rules defined for the target indexing class. Then we have the following lemmas:

Lemma 4. *The E-tree structure for ensemble model E has $\lceil \frac{n \cdot l}{M} \rceil$ leaf nodes, $\frac{1}{M-1}(\lceil \frac{nl}{M} - 1 \rceil)$ pivot nodes, and $log_M \lceil \frac{n \cdot l}{M} \rceil + 1$ levels.*

Proof. See Appendix A.6, available in the online supplemental material. □

We now analyze the prediction time under two extreme distributions for target decision rules.

Lemma 5. *In case that all target rules are located in the same node, the worst case prediction time is $O(Mlog_M \lceil \frac{n \cdot l}{M} \rceil)$ for each record.*

Proof. See Appendix A.7, available in the online supplemental material. □

Lemma 6. *In case that target rules are uniformly distributed across m leaf nodes, the worst case prediction time is $O(m \cdot Mlog_M \lceil \frac{n \cdot l}{M} \rceil)$ for each record.*

Proof. See Appendix A.8, available in the online supplemental material. □

Based on Lemma 6, the prediction time can be large if m is large because there are m paths to traverse. However, in most real-world applications m is small, because stream data exhibit temporal correlations. Decision rules trained from temporally correlated data often share the same dimensions. Thus the r values in *Theorem 2* are large. In other words, $m \longrightarrow m_0$, where $m_0 \ll \lceil \frac{n \cdot l}{M} \rceil$. Consequently, the worst case scenario can be avoided and a logarithm prediction time can be achieved in most real-world applications.

4.2 E-Tree Traversal Analysis

Tree traversal plays a key role in the *insertion* and *deletion* operations. In this part, we analyze the traversal cost for E-trees. Considering an E-tree storing N decision rules $\{R_1, \ldots, R_N\}$ summarized from a data stream, when a new

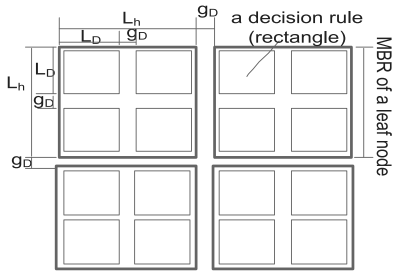


Fig. 8. Regular data model for 2D decision rules ($N = 16$).

decision rule R_x arrives, *we aim to calculate the query cost (the number of node accesses) of inserting R_x into the tree.*

Intuitively, let h be the height of the tree, L_{ij} the average minimum bounding rectangle extent of level-i $(1 \le i \le h)$ nodes along dimension j $(1 \le j \le d)$, and N_i the number of level-i nodes. Then, the expected query cost of inserting rectangle R_x is the summation of all intersections of R_x along the top-down search path, as shown in

$$NA(R_x) = \sum_{i=1}^{h} \left\{ N_i \cdot \left[\prod_{j=1}^{d} (L_{ij} + R_x.r_j) \right] \right\}, \tag{8}$$

where $R_x.r_j$ refers to the MBR extent length along the j^{th} $(1 \le j \le d)$ dimension for rectangle R_x (NA stands for node accesses). The above equation, albeit straightforward, has limited practical applicability for E-tree analysis. This is because the extents of all nodes in trees, L_{ij}, are usually not available in advance and, even if they are known (e.g., for static data), their summation may lead to expensive estimation overhead.

To estimate L_{ij}, we first borrow the concept of *regular uniform data model*, proposed by Theodoridis and Sellis [23] in R-tree analysis, as shown in Fig. 8, where

1) All rectangles share the same extent length L_D along each dimension.

2) They align into $N^{1/d}$ (N is the data set cardinality) lows/columns with a gap g_D between two consecutive rows/columns.

For example, given a collection of rectangles with density D, as shown in Fig. 8, all rectangles share the same extent length L_D and gap space g_D,

$$L_D = \left(\frac{D}{N}\right)^{\frac{1}{d}}, \quad g_D = \frac{1 - D^{1/d}}{N^{1/d} - 1}.$$

Under the *regular uniform data*, we have Lemma 7 below.

Lemma 7. *In E-trees, based on the* regular uniform data model, *the node extent at level $(i + 1)$ is*

$$L_{i+1} = f^{-1/d} L_i - g_D(1 - f^{-1/d}), \tag{9}$$

where f is the average node fanout, i.e., the number of entries in a node ($f = 4$ in Fig. 8).

Proof. See Appendix A.9, available in the online supplemental material. □

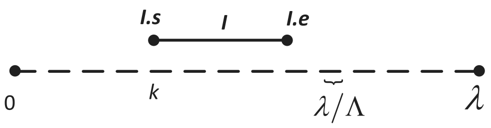


Fig. 9. The length distribution of interval I in range $[0, \lambda]$.

The *regular uniform data* model provides a basic tool for E-tree analysis. However, the model is based on the strict assumption that *decision rules at the same level share similar size*, which is impractical for data streams with drifting/evolving concepts. In the following, we relax the assumption for arbitrary extent analysis.

We first analyze the *extent distribution function* for dynamically changing decision rules, and then predict the number of node accesses by using a new evaluation function *extent random walk*.

4.2.1 Extent Distribution Function

In this part, we derive the extent distribution function of a decision rule R_x under concept drifting rate c. For simplicity, we only consider the discrete distribution function on one dimension r_j $(1 \le j \le d)$. Let $I.l \in [0, \lambda]$ be the length of decision rule R_x on dimension r_j. The range $[0, \lambda]$, as shown in Fig. 9, is divided into Λ equal partitions, each partition having length λ/Λ.

Due to concept drifting, the length of interval I is

$$I.l = |I.e - I.s|, \tag{10}$$

may randomly change length by step size λ/Λ, where $I.s$ and $I.e$ are the starting and ending points of the interval I.

The concept drifting, as shown in Fig. 10, can be described as a 1D random walk, under the state transition probability T, which can be summarized in Lemma 8.

Lemma 8. *For an interval I, let the matrix $\mathbb{T}$ in Eq. (11) be the state transition probability matrix of $I.s(e)$, then I has probability c changing its length within a random step (i.e., the concept drifting rate is c)*

$$\mathbb{T} = \begin{pmatrix} \sqrt{1-c} & 1-\sqrt{1-c} & 0 & 0 & \cdots \\ \frac{1-\sqrt{1-c}}{2} & \sqrt{1-c} & \frac{1-\sqrt{1-c}}{2} & 0 & \cdots \\ 0 & \frac{1-\sqrt{1-c}}{2} & \sqrt{1-c} & \frac{1-\sqrt{1-c}}{2} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & 0 & \frac{1-\sqrt{1-c}}{2} & \sqrt{1-c} & \frac{1-\sqrt{1-c}}{2} \\ \cdots & 0 & 0 & 1-\sqrt{1-c} & \sqrt{1-c} \end{pmatrix}. \tag{11}$$

Proof. See Appendix A.10, available in the online supplemental material. □

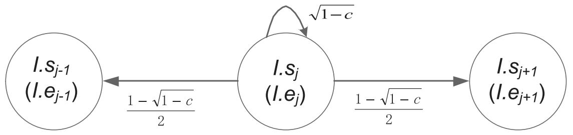


Fig. 10. The state transition probability of a Markov chain corresponding to concept drifting rate c, where an interval I has probability c lengthen/shorten its length, and probability $(1 - c)$ remains unchanged.

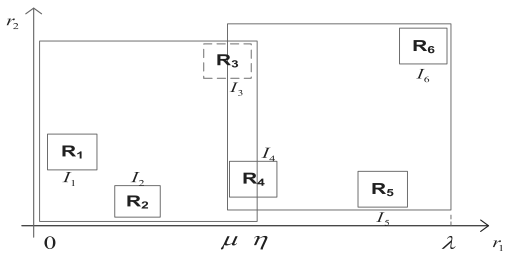


Fig. 11. Node split after inserting rule $R_3 (M = 5)$.

Lemma 9. *The distribution density $P(I.l = k\lambda/\Lambda)$ can be calculated as in*

$$\mathbb{P}(I.l = k\lambda/\Lambda) = \begin{cases} \frac{2\Lambda-1}{2\Lambda^2}, & k = 0; \\ \frac{2(\Lambda-k)}{\Lambda^2}, & k = 1, 2, \ldots, \Lambda - 1; \\ \frac{1}{2\Lambda^2}, & k = \Lambda. \end{cases} \tag{12}$$

Proof. See Appendix A.11, available in the online supplemental material. □

4.3 Extent Random Walk Function

In this part, we estimate the expected node extents L_{ij} for each level i along each dimension j. The problem can be described as, *given a set of decision rules satisfying the distribution in Eq. (12), how to estimate L_{ij}, without actually performing the exact extent calculation*?

The problem can be simplified as in Fig. 11. Assume that a data set contains five decision rules denoted by the solid rectangles. When a new rule R_3 (dashed rectangle) arrives, a split operation will be invoked leading to two new leaf nodes. The purpose is to estimate the average extents of the two generated nodes. Obviously, the extents of the two nodes along dimension r_1 are decided by boundaries μ and η, i.e., the length of the first node is $[0, \eta]$, and the second one is $[\mu, \lambda]$. In the following, *we aim to compute the expected values of points μ and η.*

Formally, assume that the range $[0, \lambda]$ is divided into Λ equal partitions with $\Lambda + 1$ stamps. Again, let $I_i = [I_i.s, I_i.e]$ $(1 \le i \le M + 1)$ be $M + 1$ independent 1D intervals, where M is the node capacity and $I_i.s(I_i.e)$ is the coordinate of the starting(ending) point of interval I_i. For each interval I_i $(1 \le i \le M + 1)$, we assume

1) Its starting point $I_i.s$ and ending point $I_i.e$ fall on above stamps.
2) Its length $I_i.l = \lambda_i.e - \lambda_i.s$ satisfies the length distribution function in Eq. (11), denoted by $\pi_k := \mathbb{P}(I.l = k\lambda/\Lambda)$.
3) Its starting point $I_i.s$ distributes uniformly in the range $\{0 \cdot \frac{\lambda}{\Lambda}, 1 \cdot \frac{\lambda}{\Lambda}, \ldots, \lambda - I_i.l\}$.

We refer to the above three conditions as *the generative rule* of the 1D interval for decision rules. Because intervals $I_i (1 \le i \le M + 1)$ are independently and identically distributed (i.i.d.), we use I to represent I_i. For the purpose of length comparison, we give the following definition:

Definition 1. *For two intervals I and J, we define*

- $I < J$*: if $I.s < J.s$, or, $I.s = J.s$ and $I.e < J.e$;*
- $I = J$*: if $I.s = J.s$ and $J.e = J.e$;*
- $I > J$*: if $I.s > J.s$, or, $I.s = J.s$ and $I.e > J.e$.*

Based on the above definition, for any two intervals, we first compare their starting points and ending points. We sort these intervals to get a new ordered sequence $I_{(1)}, I_{(2)}, \ldots, I_{(M+1)}$, where $I_{(i)} \le I_{(i+1)}$. In the new sequence, $I_{(i)}.s < I_{(i+1)}.s$ or, $I_{(i)}.s = I_{(i+1)}.s$ and $I_{(i)}.e \le I_{(i+1)}.e$. For convenience, we use $I_1, I_2, \ldots, I_{M+1}$ to denote the ordered statistics.

Assume that these intervals are split into two nodes, such that the first node contains $I_1, I_2, \ldots, I_u$, and the second one contains $I_{u+1}, \ldots, I_{M+1}$, where $u \in \{1, 2, \ldots, M + 1\}$. Let M_η be the largest ending point of the first u intervals

$$M_\eta := \max_{i=1,\ldots,u} I_i.e \tag{13}$$

The purpose is to estimate the expected values of both $\mathbb{E}(\lambda - I_{u+1}.s)$ and $\mathbb{E}(M_\eta)$.The former corresponds to the variable μ, and the latter η.

Theorem 3 (Expected value of $\mathbb{E}(\lambda - I_{u+1}.s)$).

$$\mathbb{E}(\lambda - I_{u+1}.s) = \lambda - \frac{\lambda}{\Lambda} \sum_{\mu=0}^{\Lambda} \sum_{\eta=\mu}^{\Lambda} \mu \cdot E_{\mu\eta}^{(u+1)}, \tag{14}$$

where $E_{\mu\eta}^{(u+1)}$ denotes the probability that the $(u + 1)$-th order statistic is equal to $[\mu \cdot \frac{\lambda}{\Lambda}, \eta \cdot \frac{\lambda}{\Lambda}]$.

Proof. See Appendix A.12, available in the online supplemental material. □

Theorem 4 (Expected value of $\mathbb{E}(M_\eta)$).

$$\mathbb{E}(M_\eta) = \lambda - \frac{\lambda}{\Lambda} \sum_{\xi=0}^{\Lambda-1} \sum_{\mu=0}^{\xi} \sum_{\eta=\mu}^{\xi} E_{\mu\eta(\le\xi)}^{(u)}, \tag{15}$$

where $E_{\mu\eta(\le\xi)}^{(u)}$ to denote the probability that the (u)-th order statistic is equal to $[\mu \cdot \frac{\lambda}{\Lambda}, \eta \cdot \frac{\lambda}{\Lambda}]$ and $M_\eta \le \xi \frac{\lambda}{\Lambda}$.

Proof. See Appendix A.13, available in the online supplemental material. □

4.4 Expected Query Cost

Based on the expected lengths of $\mathbb{E}(\lambda - I_{u+1}.s)$ and $\mathbb{E}(M_\eta)$, we derive in this part the expected query response time for an arbitrary query R_x with extent length $R_x.r_j = \pi$. Specifically, we first derive the expected extent length L_{ij} based on $\mathbb{E}(\lambda - I_{u+1}.s)$ and $\mathbb{E}(M_\eta)$ in *Lemma 10*, and then estimate in *Theorem 5* the average access time for the query R_x under the access cost model given in Eq. (8).

Lemma 10 (Expected value of extent length L_{ij}).

$$E(L_{ij}) = \lambda - \frac{\lambda}{2\Lambda} \left\{ \sum_{\xi=0}^{\Lambda-1} \sum_{\mu=0}^{\xi} \sum_{\eta=\mu}^{\xi} E_{\mu\eta(\le\xi)}^{(u)} + \sum_{\mu=0}^{\Lambda} \sum_{\eta=\mu}^{\Lambda} \mu E_{\mu\eta}^{(u+1)} \right\}. \tag{16}$$

Proof. See Appendix A.14, available in the online supplemental material. □

By plugging the above result into Eq. (8), we have the following conclusion:

Theorem 5 (Expected query cost). *Given a random query R_x with expected extent length π, we have*

$$NA(R_x) = \sum_{i=1}^{h} \prod_{j=1}^{d} N_i \Bigg\{ \lambda - \frac{\lambda}{2\Lambda} \Bigg[\sum_{\xi=0}^{\Lambda-1} \sum_{\mu=0}^{\xi} \sum_{\eta=\mu}^{\xi} E^{(u)}_{\mu\eta(\leq\xi)} + \sum_{\mu=0}^{\Lambda} \sum_{\eta=\mu}^{\Lambda} \mu E^{(u+1)}_{\mu\eta} \Bigg] + \pi \Bigg\}$$

Proof. See Appendix A.15, available in the online supplemental material. □

Intuitively, the above theorem reveals that the query cost depends on the tree height h, the number of nodes at each level N_i (which ultimately depends on the node capacity M), the query length π, and the distribution of rectangles in the decision space (the [. . .] part). When the rectangles are uniformly distributed and the query length π is much smaller than the window size λ, the overlapping of the query window and the observed rectangles will be small, leading to low query cost. For example, as shown in our experiments Figs. 19, 20, and 21, available in the online supplemental material, when the query length $r = 0.01$ and the window size $\lambda = 0.01$, the query cost reaches its minimum value of ~0.03 s. Generally, if the query length $\pi \leq \lambda - E(L_{ij})$, then the expected query cost is no more than $O(hM\lambda^d)$, where M is the node capacity.

The 1D random interval and its properties in R*-trees have been studied by Tao and Papadias [24]. Compared to their work, we make three contributions:

(1) We give a rigorous definition, in *Definition 1*, for 1D random interval ranking and related statistics.

(2) We derive $\mathbb{E}(\lambda - I_{u+1}.s)$, in *Theorem 3*, when the extent distribution function $F(\xi)$ is a discrete distribution, in contrast to the continuous distribution in their work.

(3) We calculate $\mathbb{E}(M_\eta)$ through the probability $E^{(\mu)}_{\mu\eta(\leq\xi)}$ instead of $P(M_s = \lambda L/\Lambda, M_\eta \leq \xi L/\Lambda)$, leading to a better estimation statistic, given in *Theorem 4*.

5 Experiments

In this section, we present extensive experiments on both synthetic and real-world data streams to validate the performance of E-trees with respect to prediction time, memory usage, and prediction accuracy. All experiments are conducted on a Linux machine with 3 GHz CPU and 2 GB memory. The E-tree source code can be downloaded from http://streammining.org.

5.1 Methodology

Benchmark data streams. Three real-world streams [25] and one synthetic steam from the UCI repository were used [25]. Table 3 lists these streams after using the F-Score feature selection [26]. The synthetic stream is similar to Example 1. It was generated as follows. First, we randomly generated a collection of stream data $\{\ldots, (x_i, y_i), \ldots\}$, where $x_i \in \mathbb{R}^d$ is a d-dimensional vector with the j^{th}element $x_{ij} \in [0, 1]$, and $y_i \in \{normal, abnormal\}$ is the class label. Thus the decision space is a d-dimensional hyper-cube. The class of each stream record is defined by the rule that if $a \leq x_i \leq b$, where $a \in [0, 1]^d, b \in [0, 1]^d, a_i < b_i$ for each $i \in [1, d]$, then "abnormal"; otherwise, "normal". That is to say, we defined a small sub-cube in the decision space as the abnormal class. To simulate concept drifting, after generating a data chunk of D records, we randomly selected its $j^{th}, (1 \leq j \leq d)$ dimension a_j, b_j to change to $a_j + u\alpha, b_j + u\alpha$, where $u = \{-1, 1\}$ controls the direction of the change and $u = -1$ with probability v. If the upper or lower bound of the decision space is reached, u will change its value. We initially set $v = 50\%, \alpha = 0.1$ and $b - a = 0.5$.

TABLE 3
Real-World Data Streams

Name	Instances	Attributes (Feature selection)	Areas
intrusion detection	4000000	11	Security
spam detection	460001	15	Security
malicious url detection	488420	14	Security

Benchmark methods. We implemented four ensemble methods for the purpose of online label search comparisons. 1) Global E-tree (*GE-tree*), which has no upper bound on the number of base classifiers in an ensemble. For each new base classifier, GE-tree simply integrates it into the ensemble and deletions are never invoked. 2) Local E-tree (*LE-tree*). Different from GE-tree, LE-tree sets an upper bound on the ensemble size. Once the upper bound is reached, the oldest classifier will be removed from the ensemble. Obviously, E-trees are in the LE-tree category. 3) Global Ensemble (*G-Ensemble*). A traditional linear scan is used during prediction. Classifiers will be added into the ensemble model continuously without deletions. 4) Local Ensemble (*L-Ensemble*). Similar to LE-tree, an upper bound on the ensemble size is set. In all the four methods, C4.5 is used to generate decision rules from data streams. All decision rules make "hard" decisions.

For E-tree traversal analysis, we compared our method with four analytical approaches for R-trees. 1) The TS model proposed by Theodoridis et al. [27], which is used to analyze nonuniform distributed rectangular objects. 2) The fractal method proposed by Proietti and Faloutsos [28], which is used to evaluate rectangular objects obeying power laws. 3) The selectivity method [29] and 4) the ERF model [24], which computes node extents as a function of node splits.

Measures. Three measures are used for online query evaluation. 1) Time cost. By using a height-balanced tree to index all classifiers in the ensemble, E-trees are expected to achieve a much lower computational cost than traditional ensemble models. 2) Memory cost. E-trees are expected to consume a larger but affordable size of memory space. 3) Accuracy. E-trees are expected to achieve the same prediction accuracy as original ensemble models.

For E-tree traversal analysis, we measure the average relative error in answering a workload of 200 decision rules with the same parameter, denoted by $(1/200) \cdot \sum_{i=1}^{200} | est_i - act_i | / act_i$, where est_i and act_i are the estimated and actual costs of the i^{th} query $(1 \leq i \leq 200)$.

5.2 Experimental Results

We compare four methods under different parameter settings with respect to number of classifiers n, parameter M, and the target indexing class. By default, the chunk size is

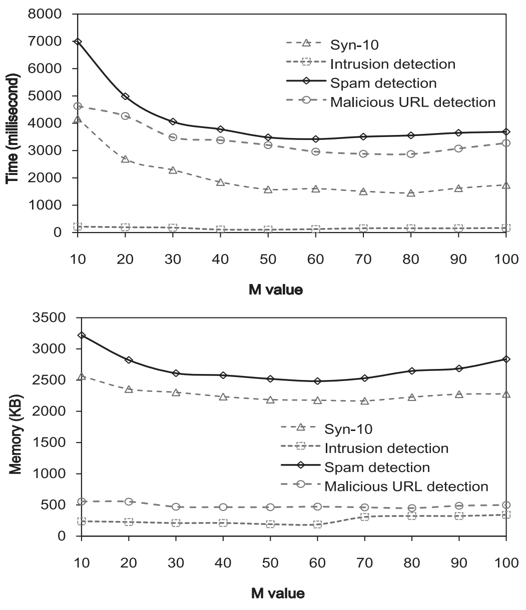


Fig. 12. Comparisons *w.r.t.* parameter M.

set to 10,000, parameter γ is set to 0.5, the ensemble size is set to 30 for the local methods and 100 for the global methods. In addition, all base classifiers are weighted by an averaging weighting scheme.

Parameter M. This parameter controls the node size and is the most important parameter for E-trees. If a node contains more than M entries, node splitting will be invoked. Ideally, M should be set such that splits do not happen when the entries in a node heavily overlap each other. Fig. 12 shows E-tree's performance with respect to different M values on both synthetic and real-world data streams. From the results we have the following observations. When M increases at the very early stage (e.g., from 10 to 30), both the prediction time and memory cost decrease significantly. After that, the benefit becomes marginal and then turns negative if M continuously increases. For example, in the spam data set, when M increases from 10 to 40, the prediction time drops sharply from 6,990 to 3,780 ms, meanwhile the memory cost also drops quickly from 3,218 to 2,520 KB. On the other hand, when M increases from 60 to 100, both the prediction and memory costs increase.

Indeed, increasing M values at an early stage reduces the numbers of pivot nodes and leaf nodes. As a result, entries that overlap heavily can be stored in the same node, leading to reduced search and storage costs. However, when M continues to increase, entries that slightly overlap with each other will be mistakenly stored in the same node, leading to extra comparisons on each node and increased search time. Therefore, a satisfactory M value should neither be too large nor too small. In the following experiments M was set to 30.

TABLE 4
Indexing Different Classes

Data stream	Class	Rules	LE-tree Time (ms)	LE-tree Mem. (KB)	GE-tree Time (ms)	GE-tree Mem. (KB)
Syn-10	Abnormal	**1234**	**1659**	**70.25**	**4712**	**3082**
	Normal	2238	3149	81.69	5921	4085
Intrusion detection	Abnormal	2308	1250	20.8	4850	340.50
	Normal	**20**	**62**	**1.50**	**169**	**114.90**
Spam detection	Spam	1172	4362	46.38	6532	3458
	Non-spam	**1089**	**3484**	**46.29**	**5933**	**2547.07**
Malicious URL detection	Malicious	**168**	**1099**	**35.04**	**1206**	**147.26**
	Benign	177	2815	37.52	3166	149.48

Ensemble size n. To evaluate how E-trees improve predicting efficiency, we compared the LE-tree and GE-tree methods by varying ensemble size n. From the results in Fig. 13, we can come to two important conclusions: (1) E-trees can significantly reduce the prediction cost. For example, in the malicious URL detection data stream, when $n = 100$, E-tree took 39,166 ms to classify a record, about four times faster than the original ensemble model, which took 172,937 ms. (2) As far as the memory cost is concerned, E-trees consume a larger but affordable memory space comparing to the original ensemble models. More results are reported in Appendix B, available in the online supplemental material.

Target indexing class. As discussed in Section 3.1, for a binary classification problem, E-trees only index decision rules from the target class. This leads to a question on which class should be selected as the target class? To answer this question, we conducted a series of experiments presented in Table 4. From the results we can observe that, indexing the minor class will enhance E-tree's performance. For example, in the intrusion detection data stream, there are 20 decision rules for "normal" and 2,308 for "abnormal". Obviously, indexing "normal" will significantly reduce the prediction time. Thus, for binary classification the minor class should be chosen as the target class.

Ensemble versus single tree. We compare ensemble and single tree on synthetic data sets where data chunk size varies from 100 to 2,000, and ensemble size equals to 30. Fig. 14 shows test results w.r.t. time cost (the left Y-axis) and prediction error rate (the right Y-axis). From the results, we can observe that compared to a single tree it always takes more time for ensemble trees to make predictions. On the other

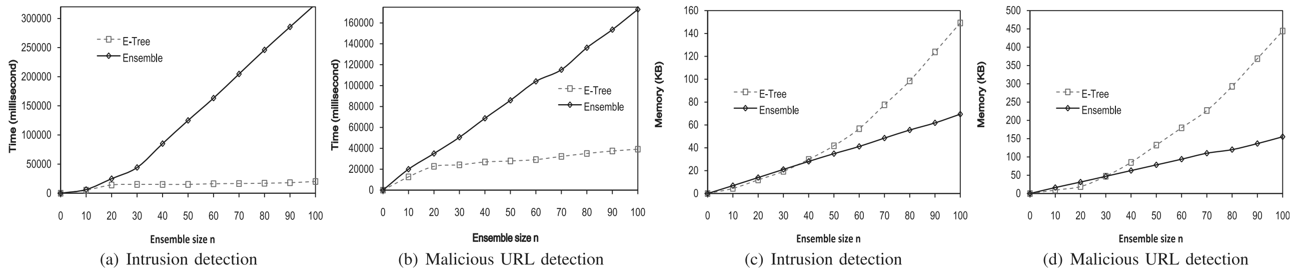


Fig. 13. E-trees versus original ensembles *w.r.t.* prediction time (a, b) and memory consumption (c, d). Obviously, E-trees significantly reduce the prediction time with affordable memory. More results are reported in Appendix B, available in the online supplemental material.

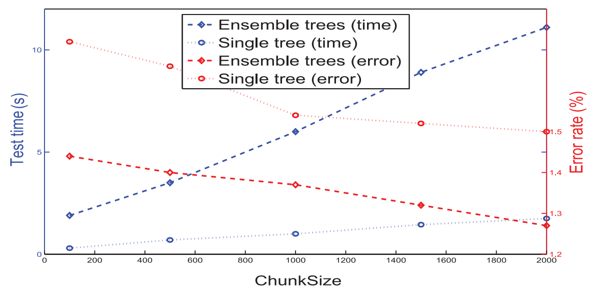


Fig. 14. Ensemble classifiers versus single classifier.

hand, ensemble trees enjoys lower prediction error rates than a single tree. Therefore, we can come to the conclusion that ensemble classifiers are a compromise solution between prediction error rate and test time costs, i.e., *ensemble classifiers obtain lower error rate at a cost of heavier time cost*. The conclusion motivates the proposed E-tree indexing structure, where the time cost can be reduced to logarithmic magnitude without increasing error rate.

Online label query. In Table 5, we compare the four benchmark methods on 10 data sets. From the results, we can observe that: (1) LE-tree performs better than GE-tree with respect to both prediction time and memory cost. For example, in the Syn-10 data stream, LE-tree is nearly three times faster than GE-tree, meanwhile takes less memory space. (2) LE-tree performs better than L-Ensemble with respect to prediction time. Besides, LE-tree achieves the same prediction accuracy as L-Ensemble. For example, in the URL data stream, LE-tree only takes 3 percent prediction time of L-Ensemble, but achieves the same prediction error rate 5.60 percent. On the other hand, although LE-tree consumes more memory than L-Ensemble, it is still an efficient method, continuously deleting outdated classifiers to release memory space. This guarantees that LE-tree will not be too large to be stored in main memory. (3) LE-tree performs better than G-Ensemble *w.r.t.* prediction time and memory cost. It is obvious that E-tree, which indexes the most recent classifiers, outperforms G-Ensemble, which linearly scans all historical classifiers during prediction.

6 Related Work

Stream classification. Existing data stream classification models can be roughly categorized into two groups: online/incremental models [30], [31] and ensemble learning [5], [6], [7], [8], [9], [10], [11], [32]. The former aims to build a single sophisticated model that can be continuously updated. Examples include the Very Fast Decision Tree (VFDT) model and the incremental SVM model. Ensemble learning employs a divide-and-conquer strategy. It first splits continuous data streams into small data chunks, and then builds light-weight base classifiers for these chunks. Ensemble models scale well to large volumes of stream data, adapt quickly to new concepts, achieve a lower variance error, and are easy to be parallelized [10]. Due to these advantages, ensemble learning has become a common tool for data stream classification.

Stream indexing. A stream classification model can be considered as a special query model that queries for class labels of incoming stream records. However, it differs from traditional query models, such as *aggregate query* [33] that aims to obtain statistical results from data streams, and *Boolean expression query* [34], [35] where queries appear as DNF or CNF expressions [34]. Classification queries involve more complex *if-then* rules that may contain both continuous and discrete attributes. They also have to face decision conflicts due to concept drifting. These differences lead to different indexing techniques. Other stream indexing methods exist that index multimedia data [36] or micro-clusters [37] on data streams. However, none of them considers the problem of indexing classifiers for anytime data stream classification.

K-d trees and M-trees. K-d trees, M-trees and R-trees are popularly used space-partitioning structures for query high dimensional data. The major difference is that K-d trees and M-trees are designed for organizing points, while R-tree is designed for organizing rectangles. Because decision trees generally cover rectangular areas, the proposed E-trees extend R-trees structure. On the other hand, the K-d trees and M-trees can be used in the instance-based learning models and SVM models. For example, our recent work [38] proposes a new L-tree structure that extends M-trees to index instance-based learning on data streams.

Spatial indexing. E-trees are originated from R-trees, which have been extensively studied in the past several decades, and many variants have been proposed to enhance its performance, such as the *R^*-trees* [17], *R^+-trees* [18], *Hilbert R-trees* [39], and *SS-trees* [40]. Techniques in these works can be employed to enhance E-trees' performance.

R-tree analysis. The earliest R-tree analytical models, such as the regular model [23] and the 1D R/R+-trees model [41],

TABLE 5
Comparisons with Benchmark Methods

Data stream	LE-tree			L-Ensemble			GE-tree			G-Ensemble		
	Time (ms)	Memory (KB)	Error (%)	Time (ms)	Memory (KB)	Error (%)	Time (ms)	Memory (KB)	Error (%)	Time (ms)	Memory (KB)	Error (%)
Syn-5	**1712**	12.29	4.37	32360	**1.67**	4.37	3001	2320.97	4.01	345140	121.42	4.01
Syn-10	**1659**	70.25	7.05	37125	**1.63**	7.05	4712	3082	6.41	345140	690.42	6.41
Intrusion	**62**	1.50	1.03	337	**0.93**	1.03	169	114.90	0.96	444.50	107.76	0.96
Spam	**3484**	46.29	8.28	47406	**3.15**	8.28	5933	2547.07	9.18	368969	155.17	9.18
URL	**1099**	35.04	5.64	35890	**0.93**	5.64	1206	147.26	5.64	136891	78.27	5.64
Adult	**6523**	80.80	19.66	20361	**6.37**	19.66	7452	2135.06	19.59	449078	800.14	19.59
Magic	**3145**	40.45	7.18	36250	**3.07**	7.18	7263	1378.14	4.74	336880	401.40	4.74
Winered	**3657**	46.50	22.32	31250	**3.57**	22.32	11220	632.56	23.38	305950	463.10	23.38
Winewhite	**3031**	118.50	23.43	38120	**2.95**	23.43	8849	5825.37	23.43	350640	1070.60	23.43
Census	**782**	261.70	5.77	133600	**0.76**	5.77	1153	9375	5.73	1234840	2518.80	5.73

assumed that the extents of a node satisfy uniform distributions in all dimensions, which limited their practical applicability. Later, the assumption was relaxed to nonuniform data analysis models [23], [28], [29], based on the rationale that objects within a sufficiently small region are almost uniform. These models, however, need analytic expressions of node size, which cannot meet the needs of dynamic spatial/temporal data, where the extents follow certain probabilistic distribution that is related to the length distribution of objects' lifespan. A recent work [24] proposed the *extent regression model* (ERM) that can analyzes R/R*-trees under arbitrary extent distributions.

Ensemble pruning. For the purpose of reducing computational and memory costs, ensemble pruning [42], [43] searches for a good subset of all ensemble members that perform as good as the original ensemble. A basic assumption in ensemble pruning is that all the base models share a unique global probability distribution. Therefore, some base models can be discarded from the ensemble to reduce prediction costs. However, in data streams with dynamic changing concepts, it is difficult to predict which base model(s) can be discarded from the ensemble.

Online trees. Various online decision trees have been proposed for real-time data stream classification/regression. For example, the option trees [44] have been modified for data stream classification [45] and regression [32] by enabling continuous learning and any-time prediction. On the other hand, ensembling multiple sophisticated trees is also an important research topic. For example, the online random forests [46] were proposed for online computer vision and machine learning applications, where the off-line RFs are modified to cater sequentially arriving data and continuously changing data distribution. However, these online trees mainly focus on enabling/accelerating the online training/learning part, while neglecting the online testing/prediction part. Therefore, our work can be viewed as an complement work to theirs.

Classifier indexing. Indexing classifiers can enhance system performance in data intensive applications. For example, in search engines and relevance feedback systems, a query can be a ranking function learned by SVMs. Processing the query to find top-k results requires evaluating the entire database by the query function. To alleviate the query response time, several inexact/exact indexing solutions were proposed for SVMs [47], [48]. Compared to these work, this paper extends the idea of indexing one classifier to multiple classifiers, where exploring shared patterns among classifiers plays a key role in improving the system performance.

7 Conclusions

In this paper, we proposed a novel E-tree indexing structure for sublinear time complexity for classifying high speed stream records. The main contributions of this study are threefold: (1) We formulate and address the prediction efficiency problem for ensemble models on data streams, which is a legitimate research problem well motivated by increasing real-time applications. (2) Our solution converts ensemble models into spatial databases and applies spatial indexing techniques to achieve sub-linear prediction. This novel technique can be extended to other data stream classification models besides ensemble learning, or general classification models that require timely prediction. (3) The proposed E-tree evaluation method can be extended to spatial/temporal data analysis where data nodes have arbitrary extents.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 61003167), China's 863 project (No. 2011AA010703) and the Strategic Leading Science and Technology Projects of Chinese Academy of Sciences (No. XDA06030200).

References

[1] C. Aggarwal, *Data Streams: Models and Algorithms*. Springer, 2006.

[2] P. Zhang, J. Li, P. Wang, B. Gao, X. Zhu, and L. Guo, "Enabling Fast Prediction for Ensemble Models on Data Streams," *Proc. 17th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2011.

[3] M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints," *IEEE Trans. Knowledge and Data Eng.*, vol. 23, no. 6, pp. 859-874, June 2011.

[4] J. Gao, R. Sebastiao, and P. Rodrigues, "Issues in Evaluation of Stream Learning Algorithms," *Proc. 15th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2009.

[5] H. Wang, W. Fan, P. Yu, and J. Han, "Mining Concept-Drifting Data Streams Using Ensemble Classifiers," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2003.

[6] W. Street and Y. Kim, "A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification," *Proc. Seventh ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2001.

[7] P. Zhang, X. Zhu, Y. Shi, L. Guo, and X. Wu, "Robust Ensembler Learning for Mining Noisy Data Streams," *Decision Support Systems*, vol. 50, no. 2, pp. 469-479, 2011.

[8] J. Kolter and M. Maloof, "Using Additive Expert Ensembles to Cope with Concept Drift," *Proc. 22nd Int'l Conf. Machine Learning (ICML)*, 2005.

[9] J. Gao, W. Fan, and J. Han, "On Appropriate Assumptions to Mine Data Streams: Analysis and Practice," *Proc. Seventh IEEE Int'l Conf. Data Mining (ICDM)*, 2007.

[10] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavalda, "New Ensemble Methods for Evolving Data Streams," *Proc. 15th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2009.

[11] P. Zhang, X. Zhu, J. Tan, and L. Guo, "Classifier and Cluster Ensembles for Mining Concept Drifting Data Streams," *Proc. IEEE 10th Int'l Conf. Data Mining (ICDM)*, 2010.

[12] X. Zhu, P. Zhang, X. Lin, and Y. Shi, "Active Learning from Stream Data Using Optimal Weight Classifier Ensemble," *IEEE Trans. System, Man, Cybernetics, Part B: Cybernetics*, vol. 40, no. 4, pp. 1-15, Dec. 2010.

[13] J. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[14] J. Ma, L. Saul, S. Savage, and G. Voelker, "Identifying Suspicious URLs: An Application of Large-Scale Online Learning," *Proc. 26th Ann. Int'l Conf. Machine Learning (ICML)*, 2009.

[15] R. Guting, "An Introduction to Spatial Database Systems," *VLDB J.*, vol. 3, no. 4, pp. 357-399, 1994.

[16] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 1984.

[17] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 1990.

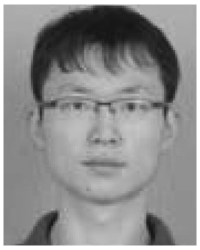
[18] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R$^+$-tree: A Dynamic Index for Multi-Dimensional Objects," *Proc. 13th Int'l Conf. Very Large Data Bases (VLDB)*, 1987.

[19] P. Ciaccia, M. Patella, and P. Zezula, "M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces," *Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB)*, 1997.

[20] T. Sellis, N. Roussopoulos, and C. Faloutsos, “Multi-Dimensional Access Methods: Trees have Grown Everywhere,” *Proc. 23rd Int’l Conf. Very Large Data Bases (VLDB)*, 1997.
[21] M. Queyranne, “Minimizing Symmetric Submodular Functions,” *Math. Programming*, vol. 82, pp. 3-12, 1998.
[22] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom, “Adaptive Ordering of Pipelined Stream Filters,” *Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD)*, 2004.
[23] Y. Theodoridis and T. Sellis, “A Model for the Prediction of R-Tree Performance,” *Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, 1996.
[24] Y. Tao and D. Papadias, “Performance Analysis of R*-Trees with Arbitrary Node Extents,” *IEEE Trans. Knowledge and Data Eng.*, vol. 16, no. 6, pp. 653-668, June 2014.
[25] A. Asuncion and D. Newman, “UCI Machine Learning Repository,” http://mlearn.ics.uci.edu/databases/, 2007.
[26] C. Chang and C. Lin, “LIBSVM: A Library for Support Vector Machines,” http://www.csie.ntu.edu.tw/cjlin/libsvm, 2001.
[27] Y. Theodoridis, E. Stefanakis, and T. Sellis, “Efficient Cost Models for Spatial Queries Using R-Trees,” *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 1, pp. 19-32, Jan./Feb. 2000.
[28] G. Proietti and C. Faloutsos, “I/O Complexity for Range Queries on Region Data Stored Using an R-Tree,” *Proc. 15th Int’l Conf. Data Eng. (ICDE)*, 1999.
[29] S. Acharya, V. Poosala, and S. Ramaswamy, “Selectivity Estimation in Spatial Databases,” *Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD)*, 1999.
[30] P. Domingos and G. Hulten, “Mining High-Speed Data Streams,” *Proc. Sixth ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining (KDD)*, 2000.
[31] C. Domeniconi and D. Gunopulos, “Incremental Support Vector Machine Construction,” *Proc. IEEE Int’l Conf. Data Mining (ICDM)*, 2001.
[32] E. Ikonomovska, J. Gama, B. Zenko, and S. Dzeroski, “Speeding-Up Hoeffding-Based Regression Trees with Options,” *Proc. 28th Int’l Conf. Machine Learning (ICML)*, 2011.
[33] S. Babu and J. Widom, “Streamon: An Adaptive Engine for Stream Query Processing,” *Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD)*, 2004.
[34] S. Whang and H. Molina, “Indexing Boolean Expressions,” *Proc. VLDB Endowment*, vol. 2, pp. 37-48, 2009.
[35] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram, “Scalable Ranked Publish/Subscribe,” *Proc. VLDB Endowment*, vol. 1, pp. 451-462, 2008.
[36] Z. Liu, S. Parthasarathy, A. Ranganathan, and H. Yang, “Near-Optimal Algorithms for Shared Filter Evaluation in Data Stream Systems,” *Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD)*, pp. 133-145, 2008.
[37] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, “The Clustree: Indexing Micro-Clusters for Anytime Stream Mining,” *Knowledge and Information Systems*, vol. 29, pp. 249-272, 2010.
[38] P. Zhang, B. Gao, X. Zhu, and L. Guo, “Enabling Fast Lazy Learning for Data Streams,” *Proc. IEEE 11th Int’l Conf. Data Mining (ICDM)*, 2011.
[39] I. Kamel and C. Faloutsos, “Hilbert R-Tree: An Improved R-Tree Using Fractals,” *Proc. 20th Int’l Conf. Very Large Data Bases (VLDB)*, pp. 500-509, 1994.
[40] D. White and R. Jain, “Similarity Indexing with the SS-Tree,” *Proc. IEEE 12th Int’l Conf. Data Eng. (ICDE)*, 1996.
[41] C. Faloutsos, T. Sellis, and N. Roussopoulos, “Analysis of Object Oriented Spatial Access Methods,” *Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD)*, 1987.
[42] Y. Zhang, S. Burer, and W. Street, “Ensemble Pruning via Semi-Definite Programming,” *J. Machine Learning Research*, vol. 7, no. 2006, pp. 1315-1338, 2006.
[43] Z. Lu, X. Wu, X. Zhu, and J. Bongard, “Ensemble Pruning via Individual Contribution Ordering,” *Proc. 16th ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining (KDD)*, 2010.
[44] R. Kohavi and C. Kunz, “Option Decision Trees with Majority Votes,” *Proc. 14th Int’l Conf. Machine Learning (ICML)*, 1997.
[45] B. Pfahringer, G. Holmes, and R. Kirkby, “New Options for Hoeffding Trees,” *Proc. Australian Joint Conf. Artificial Intelligence*, 2009.
[46] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, “On-Line Random Forests,” *Proc. 12th Int’l Conf. Computer Vision Workshops (ICCV Workshops)*, 2009.
[47] H. Yu, I. Ko, Y. Kim, S. Hwang, and W. Han, “Exact Indexing for Support Vector Machines,” *Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD)*, 2011.
[48] N. Panda and E. Chang, “Exploiting Geometry for Support Vector Machine Indexing,” *Proc. SIAM Int’l Conf. Data Mining (SDM)*, 2005.

Peng Zhang received the PhD degree in computer science from the Chinese Academy of Sciences in July 2009. He is an associate professor at the Institute of Information Engineering, Chinese Academy of Sciences. He was a postdoctoral research associate at Texas State University and a research assistant professor at Florida Atlantic University. His research interests include data stream mining, social network analysis, and convex optimization.

Chuan Zhou received the PhD degree in mathematics from the Chinese Academy of Sciences in 2013. He is currently a postdoctoral research associate at the Institute of Information Engineering, Chinese Academy of Sciences. His research interests include applied stochastic processes and their applications in machine learning, information retrieval, and data mining.



Peng Wang received the bachelor’s degree from the Department of Computer Science, Beijing University of Posts and Telecommunications. He is currently working toward the PhD degree at the Institute of Information Engineering, Chinese Academy of Sciences. His research interests include data stream mining and social network mining.

Byron J. Gao received the BSc and PhD degrees in computer science from Simon Fraser University, Canada, in 2003 and 2007, respectively. He was a postdoctoral fellow at the University of Wisconsin-Madison before joining Texas State University in 2008. His research spans several related fields including data mining, databases, information retrieval, and bioinformatics.

Xingquan Zhu is an Associate Professor in the Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University. In 2014, he began serving a second term as an Associate Editor of the *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (previously from 2008 to 2012).

Li Guo is a professor with the Institute of Information Engineering, Chinese Academy of Sciences. She is also the chairman of the Intelligent Information Processing Research Center, Institute of Information Engineering, Chinese Academy of Sciences. Her research interests include data stream management systems and information security.