Enabling Fast Prediction for Ensemble Models on Data Streams

Peng Zhang[†,⋆], Jun Li[†], Peng Wang[†], Byron J. Gao[‡], Xingquan Zhu[§], Li Guo[†,⋆]

[†] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190, China
[⋆] National Engineering Laboratory for Information Content Security, Beijing, China
[‡] Department of Computer Science, Texas State University - San Marcos, TX 78666-4616, USA
[§] QCIS, Faculty of Eng. & IT, University of Technology, Sydney, NSW 2007, Australia
{zhangpeng, lijun, wpeng, guoli}@ict.ac.cn, bgao@txstate.edu, xqzhu@it.uts.edu.au

ABSTRACT

Ensemble learning has become a common tool for data stream classification, being able to handle large volumes of stream data and concept drifting. Previous studies focus on building accurate prediction 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the other hand, E-trees can automatically update themselves by continuously integrating new classifiers and discarding outdated ones, well adapting to new trends and patterns underneath data streams. Experiments on both synthetic and real-world data streams demonstrate the performance of our approach.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithm, Performance

Keywords

Stream data mining, stream classification, ensemble learning, spatial indexing, concept drifting

1. INTRODUCTION

Data stream classification represents one of the most important tasks in stream data mining [1]. Compared to traditional classification, data stream classification faces two additional challenges: large/increasing data volumes and drifting/evolving concepts [27, 32]. To address these challenges, many ensemble-based models have been proposed recently, including weighted classifier ensembles [14, 26, 31, 16, 35], incremental classifier ensembles [4], classifier and cluster ensembles [33], to name a few. While these models vary from one to another, they share the same fundamental idea: *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 all 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, it has become one of the most popular techniques in data stream classification.

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 undemanding applications. (2) Existing works only consider combining a small number of base classifiers, *e.g.*, no more than 30 [14, 32, 33, 35, 35]. 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 Web page stream monitoring, ensemble learning can be used to identify malicious pages from normal pages, both arriving continuously, in real time. We implemented a detection system on a Linux machine with 3GHz cpu and 2GB memory. Each day, a batch of base classifiers are trained using decision trees (C4.5 algorithm [22, 30]) with all base classifiers combined to classify pages in the next day. In our experiments, in total 120 days of stream data [19] were used.

The curve in Fig. 1 summarizes our experimental results, showing the typical linear relationship between prediction time and ensemble size. For example, an ensemble with 50 members takes 0.083 second to classify a page. Suppose that the Web page stream flows 10,000 pages per second, which is common for backbone networks. Then, monitoring all stream records requires a sophisticated parallel computing architecture having 830 processors!

To achieve sub-linear prediction, a practical approach is to explore shared patterns among all base classifiers. Without loss of

KDD'11, August 21–24, 2011, San Diego, California, USA.

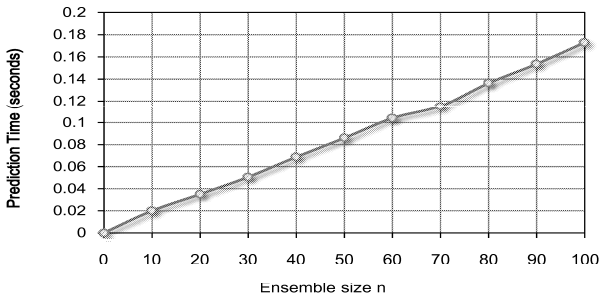


Figure 1: Prediction time vs. ensemble size

generality, suppose that the base classifiers 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 the spatial objects (base classifiers) in the spatial database (ensemble model).

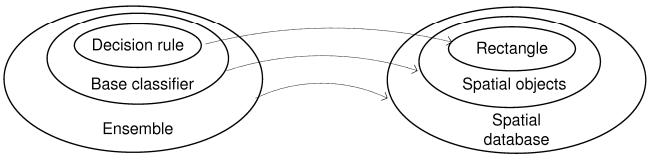


Figure 2: Mapping an ensemble model to a spatial database.

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 [11]. Examples include *R-tree* [12], R^**-tree* [25], R^+*-tree* [23], *M-tree* [6], to name a few [24]. Generally speaking, 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. Thus a decision region in ensemble models may be associated with different class labels.

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 the E-tree to classify an incoming stream record x; (2) Insertion: Integrate new classifier member into the E-tree; (3) Deletion: Remove the outdated classifier from the E-tree. As a result, E-tree approach not only guarantees a logarithmic time complexity for prediction, but also able to adapt to new trends and patterns in stream data.

Table 1: 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

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 the related work. We conclude the paper in Section 7.

2. PROBLEM DESCRIPTION

Formally, 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 invisible unless the sample is properly labeled. Suppose that we have built n base classifiers $C_1, C_2, \cdots, C_n$ 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 "$if...then...$"). 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 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 having two classes (normal and 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 $C_1, C_2, \cdots, C_5$ built along the stream are incorporated in the ensemble model E, with decision rules for the base classifiers listed in Table 1. 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.

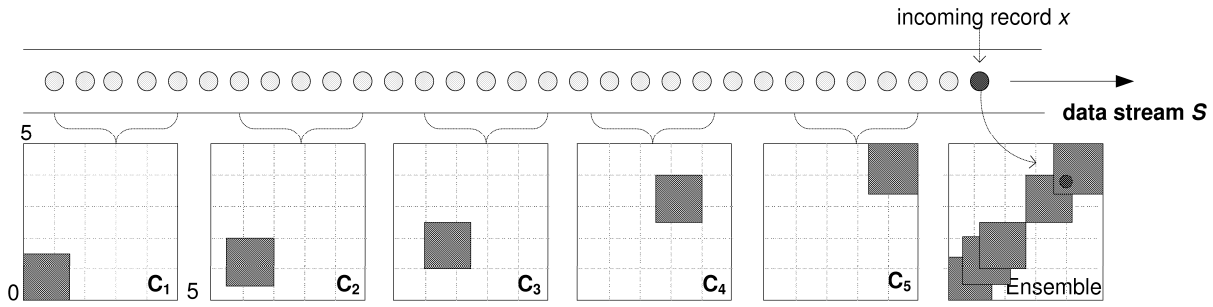


Figure 3: Mapping ensemble E in Example 1 to spatial database A_E.

3. THE E-TREE INDEXING STRUCTURE

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

3.1 The basic structure of E-trees

An E-tree, as shown in Fig. 4, is a height-balanced tree mainly consisting of two components: (1) an *R-tree* like structure T on the right-hand side storing all decision rules of the ensemble, and (2) a *table structure* on the left-hand side storing all the classifier-level information of the ensemble, such as IDs and weights of the classifiers. The two structures are connected together by linking each classifier in the table to its corresponding decision rules in the tree.

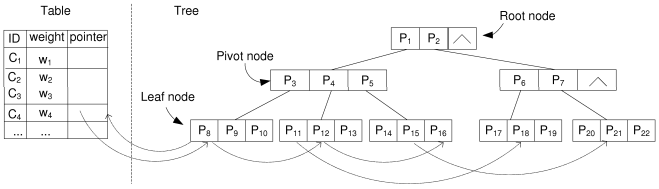


Figure 4: An illustration of E-tree (some links are omitted).

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

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

where I 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 $I = (I_0, I_1, \cdots, I_d)$ with each I_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

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

where I is the smallest rectangle covering all the decision rules in its child nodes, and *child_pointer* references its child node. Every non-root node contains between m and M entries. The root node has at least two entries unless it is a leaf.

The table structure contains information about all base classifiers in an ensemble model with each entry denoted as follows:

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

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.

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 constrains:

- All decision rules make 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*", 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 $(r_1 \le 1.5) \wedge (r_2 \le 1.5)$, then the lower bound $r_1 = 0, r_2 = 0$ will be added to make it a closed space $(0 \le r_1 \le 1.5) \wedge (0 \le r_2 \le 1.5)$. 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 as follows:

  $\mathcal{R} = (0 \le r_1 \le 1.5)$

  We will expand $\mathcal{R}$ to a closed space as

  $\mathcal{R}' = (0 \le r_1 \le 1.5) \wedge (0 \le r_2 \le 5)$.

- All decision rules are "hard" decisions. For example, a Web page has 100% 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. Moreover, we only index decision rules from one class (the minor class containing fewer decision rules). For multi-class classification, an intuitive method is to combine multiple E-trees together by using a one-against-one or 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.

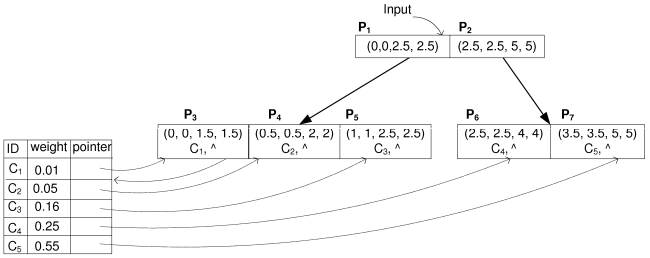


Figure 5: E-tree for the ensemble model in Example 1.

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 decision rules using Eq. (4),

$$y_x = sgn(\Sigma_{i=1}^{u} w_i P(C_{index}|x),\ \gamma), \tag{4}$$

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.

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). In the following, we use Example 3 derived from Example 1 to explain the search process.

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 can be calculated as follow:

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

Therefore, x is predicted as abnormal. In this example, at worst four comparisons will be used (P_1, P_2, P_6, and P_7). Compared to a linear scan that needs five comparisons, E-tree achieves a 20% 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.

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

Algorithm 1: Search

Input : E-tree T, stream record x, parameter γ
Output: x's class label y_x

```
Initialize(stack) ;    // initialize a stack for search
U ← ∅ ;                // U records all rules covering x
P ← T.tree.root ;      // get the root of the tree
foreach entry R ∈ T do
    push(stack, R) ;
while stack ≠ ∅ do
    e ← pop(stack) ;
    if e is an entry of a leaf then
        U ← U ∪ e ;
    else
        P ← e.child ;
        foreach entry e ∈ P do
            if x ∈ e then
                push (stack, e) ;
foreach entry e ∈ U do
    find its weights in the table structure ;
y_x ← Call Eq. (4) to calculate x's class label ;
Output y_x ;
```

M entries, then 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 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, by 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 can not 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,

$$< P_L, P_R >= argmin_{<X,Y>|X,Y \in R \cup P} (X.I + Y.I,) \tag{5}$$

where X and Y are variables, and P_L and P_R are the new nodes containing R and all entries in P. Obviously, solving Eq. (5) requires investigation of all possible groupings of all nodes in $R \cup P$, and thus is an NP-hard problem. In other words, when M is large, Eq. (5) is very hard to solve. Then, a greedy heuristic would first randomly select two entries in $R \cup P$ that has the largest distance, and then cluster all the remaining $m - 1$ entries in $R \cup P$ into the given 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 the page limit, we omit the table structure in the E-tree.

Example 4. We insert the five classifier 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

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**
  $L \leftarrow searchLeaf(R, P)$; // leaf node 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' ;

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 in the table
$R \leftarrow searchClassifier(C, A)$; // get C from table A
$P \leftarrow R.pointer$;
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**
    $T' \leftarrow deleteNode(T, L')$; // recursive deletion
    foreach *entry* $e \in L'$ **do**
      $T' \leftarrow insertRule(e, T)$;
  $P \leftarrow q$;
Output T' ;

inserted into the right leaf node directly as the node size is smaller than M.

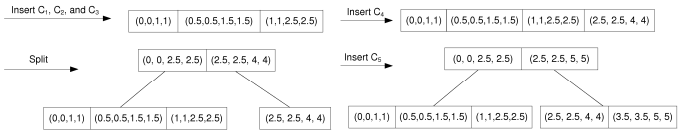


Figure 6: Inserting the five classifiers in Example 1 to E-tree.

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. (2) Re-insertion will incrementally refine the spatial structure of the tree. In addition, it has been shown [12] 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 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.

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 gets full, all the 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.

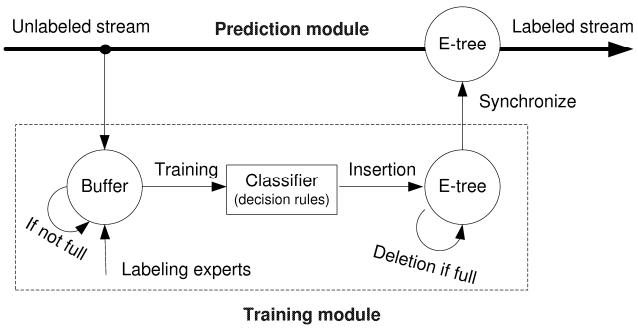


Figure 7: Architecture of Ensemble Learning with E-trees.

4. THEORETICAL ANALYSIS OF E-TREES

In this section, we provide theoretical study on the properties and performance of E-trees. In 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. On the other hand, in the worst scenario, if the target decision rules span all the leaf nodes, the search algorithm has to traverse all possible pathes 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 $d-$ dimensional space $S = S_1 \times S_2 \times \cdots \times S_d$, where $S_i \in [0, s_i]$, $1 \leq i \leq d$. Since not all rules cover all the d-dimensions, rules R_a and R_b may overlap on some dimensions. Without loss of generality, assume rule R_a is defined on r_a dimensions $S_1 \times S_2 \times \cdots \times S_{r_a}$, with the i^{th} $(1 \leq i \leq r_a)$ dimension S_i spanning in a_i region (out of the whole domain $[0, s_i]$), and rule R_b is defined on r_b dimensions $S_{1'} \times S_{2'} \times \cdots \times S_{r_b}$, with the j^{th} $(1 \leq j \leq r_b)$ dimension S_j spanning in b_j region. Due to the 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 1 as follows:

THEOREM 1. *Given two decision rules R_a and R_b, the probability that they are located in the same leaf node $\mathcal{P}$ is*

$$\mathcal{P} \propto \prod_{i=1}^{r} \frac{a_i \cdot b_i}{s_i \cdot s_i} \prod_{j=r+1}^{r_a} \frac{a_j}{s_j} \prod_{k=r+1}^{r_b} \frac{b_k}{s_k}. \tag{6}$$

PROOF. Obviously, for each of the first r common dimensions, S_i, the common region between the two rules on this dimension is the overlapping region between a_i and b_i, which is equivalent to the expectation of observing a joint event a_i and b_i, as $E(a_i \wedge b_i)$. Because observing a_i out of the whole domain $[0, s_i]$ is a_i/s_i, and R_a and R_b are assumed to be independently generated, we have

$$E(a_i \wedge b_i) = E(a_i) \cdot E(b_i) = \frac{a_i \cdot b_i}{s_i \cdot s_i}. \tag{7}$$

For any of the remaining $r_a - r$ dimensions for rule R_a (*i.e.*, the j^{th}, $r \leq j \leq r_a$ dimension), it spans in a region a_j. For rule R_b, it spans over the whole s_j region (Recall in Section 3.1, a disclosed decision rule will be spanned over the whole undefined attribute space). As a result, the common region between the two rules in s_j is given as

$$E(a_j) = \frac{a_j}{s_j}. \tag{8}$$

Similarly, for decision rule R_b, the shared interval on the k^{th}, $1 \leq k \leq r_b$ dimension is b_k/s_k. Because dimensions and rules are assumed to be independent, the total overlapping area can be calculated by multiplying all the above three terms, as defined in Theorem 1. □

Assume that an incoming record has t target decision rules that are mutually independent, the probability that all these rules being located in the same leaf node is proportional to

$$\mathcal{P}' = (\prod_{i=1}^{r} \frac{a_i \cdot b_i}{s_i \cdot s_i} \prod_{j=r+1}^{r_a} \frac{a_j}{s_j} \prod_{k=r+1}^{r_b} \frac{b_k}{s_k})^{t-1}. \tag{9}$$

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 were conducted on a Linux machine with 3GHz CPU and 2GB memory. The E-tree source code can be downloaded from *http://streammining.org*.

Benchmark data streams. Three real-world streams and one synthetic steam from the UCI repository were used [2]. Table 2 lists the real streams after using the F-Score feature selection [5]. All of them are binary classification problems. Due to the page limit, please refer to [2] for detailed descriptions. The synthetic stream is similar to Example 1. It was generated as follows. First, we randomly generated a collection of stream data $\{\cdots, (x_i, y_i), \cdots\}$, 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 was 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 50% probability. If the upper or lower bound of the decision space is reached, u will change it value. For simplicity, we initially set $\alpha = 0.1$ and $b - a = 0.5$.

Table 2: Real-world data streams

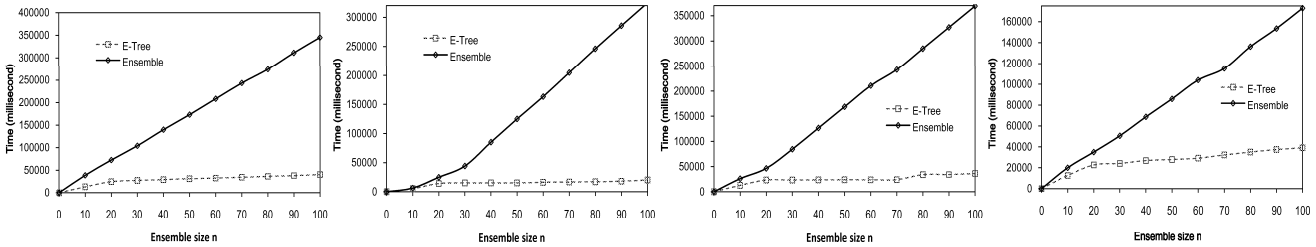
Name	Instances	Attributes (Feature selection)	Areas
intrusion detection	4000000	11	Security
spam detection	460001	15	Security
malicious url detection	488420	14	Security

Benchmark methods. We implemented four benchmark methods for comparison purposes. (1) Global E-tree (**GE-tree**). In this method, there is 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**). In this method, 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 was used to generate decision rules from data streams. All the decision rules are "hard" decisions.

Measures. Three measures were used. (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 typical ensemble models. (2) Memory cost. E-trees are expected to consume a larger yet affordable memory space. (3) Accuracy. E-trees are expected to achieve the same prediction accuracy as original ensemble models.

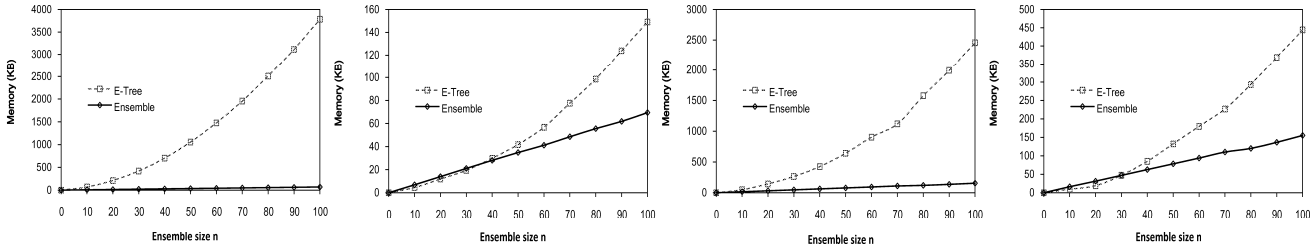
Experimental results. We compared the four methods under different parameter settings on number of classifiers n, parameter M, and the target indexing class. By default the chunk size was set to 10000, parameter γ was set to 0.5, the ensemble size was set to 30 for the local methods and 100 for the global methods. Besides, all base classifiers were weighted by an averaging weighting scheme.

Parameter M. This parameter controls the node size and is the most important one 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. 8 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



(a) Syn-10 (b) Intrusion detection (c) Spam detection (d) Malicious URL detection

Figure 9: E-trees vs. original ensemble models with respect to prediction time. We can observe that E-trees significantly reduced the prediction time.



(a) Syn-10 (b) Intrusion detection (c) Spam detection (d) Malicious URL detection

Figure 10: E-trees vs. original ensemble models with respect to memory consumption. We can observe that E-trees consume larger yet affordable memory space.

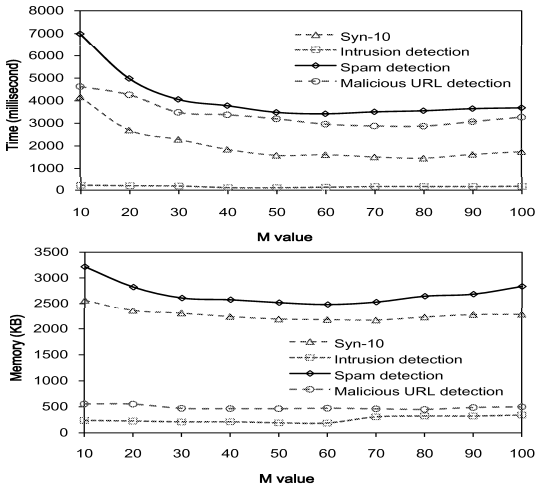


Figure 8: Comparisons with respect to different M values.

benefit becomes marginal and then turns negative with increasing M. For example, in the spam data set, when M increases from 10 to 40, the prediction time drops sharply from 6990ms to 3780ms, meanwhile the memory cost also drops quickly from 3218KB to 2520KB. On the other hand, when M increases from 60 to 100, both the prediction and memory costs increase.

To see why, increasing M 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 3: Indexing different classes

Data stream	Class	Rules	LE-tree		GE-tree	
			Time (ms)	Mem. (KB)	Time (ms)	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 on predicting efficiency, we compared the LE-tree and GE-tree methods by varying ensemble size n. From the results in Fig. 9 and Fig. 10, 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 39166ms to classify a record, about four times faster than the original ensemble model, which took 172937ms. (2) As far as the memory cost is considered, E-trees consumes larger yet affordable memory space comparing to the original ensemble models.

Target indexing class. As discussed in Section 3.1, for a binary classification problem, E-trees only index decision rules from the target class. Which class should be selected as the target class? To answer this question, we conducted a series of experiments presented in Table 3. 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 2308 for "abnormal". It is obvious that indexing "normal" will significantly reduce the prediction time. Thus, for

Table 4: Comparisons among 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

binary classification the minor class should be chosen as the target class.

Comparisons among benchmark methods. We compared the four benchmark methods on 10 data sets as shown in Table 4. 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 with L-Ensemble. For example, in the URL data stream, LE-tree only takes 3% prediction time of L-Ensemble, but achieves the same prediction error rate 5.60%. 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 with respect to prediction time and memory cost. It is obvious that E-tree, which indexes the most recent classifiers, will perform better than G-Ensemble, which linearly scans all historical classifiers during prediction.

6. RELATED WORK

Stream classification. Existing data stream classification models can be categorized into two groups: online / incremental models [8, 7] and ensemble learning [14, 26, 31, 16, 10, 36, 33, 4]. 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 [4]. 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* [3] that aims to obtain statistical results from data streams, and *boolean expression query* [28, 20] where queries appear as DNF or CNF expressions [28]. 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 [17] or micro-clusters [21] on data streams. However, none of them considers the problem of indexing classifiers for anytime data stream classification.

Spatial indexing. The E-tree originates from the R-tree. The R-tree has been extensively studied in the past several decades, and many variants have been proposed to enhance its performance, such as the R^**-tree* [25], R^+*-tree* [23], *Hilbert R-tree* [15], and *SS-tree* [29]. The techniques in these works can be employed to enhance E-tree's performance.

Ensemble pruning. For the purpose of reducing computational and memory costs, ensemble pruning [34, 18] searches for a good subset of all ensemble members that performs as well as the original ensemble. A basic assumption in ensemble pruning is that all the base models in the ensemble share a unique global probability distribution. Thus, some base models can be discarded from the ensemble to reduce prediction costs. However, in dynamic data streams, concepts change continuously, and it is very difficult to predict which base model(s) can be discarded from the ensemble.

7. CONCLUSIONS

Ensemble learning has become a common tool for data stream classification. However, existing ensemble models perform a linear scan of all base classifiers during prediction, which is increasingly inadequate for many real world applications that involve large volumes of data steams with concept drifting and require timely responses. To address this problem, we proposed a novel E-tree indexing structure to organize the base classifiers, achieving $O(log(N))$ time prediction for incoming stream records. Comprehensive experiments on both synthetic and real-world data streams demonstrated the performance of E-trees for stream classification.

In this study, base classifiers are trained using decision trees and decision rules can be represented as spatial objects in the decision space. In case that base classifiers are trained using other non-axis-parallel classification models such as support vector machines, k nearest neighbors or neural networks, rule extraction can be applied to extract if-then rules from such black-box models [9, 13]. Then, these models can also be converted to spatial objects and indexed using E-trees.

The main contributions of this study are twofold: (1) We are the first to formulate and address the prediction efficiency problem for ensemble models on data streams, which is a legitimate research problem well motivated by increasing real-world 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.

8. ACKNOWLEDGMENTS

This research was partially supported by the National Science Foundation of China (NSFC) under Grant No.61003167, Basic Research Program of China (973 Program) under Grant No.2007CB311100, the Texas Norman Hackerman Advanced Research Program under Grant No.003656-0035-2009, and Australian Research Council (ARC) Future Fellowship under Grant No.FT100100971.

9. REFERENCES

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

[2] A. Asuncion and D. Newman. *UCI Machine Learning Repository*. Irvine, CA, 2007.

[3] S. Babu and J. Widom. Streamon: An adaptive engine for stream query processing. In *Proc. of SIGMOD 2004*.

[4] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavalda. New ensemble methods for evolving data streams. In *Proc. of KDD 2009*.

[5] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines, Software available at: http://www.csie.ntu.edu.tw/ cjlin/libsvm*, 2001.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. of VLDB 1997*.

[7] C. Domeniconi and D. Gunopulos. Incremental support vector machine construction. In *Proc. of ICDM 2001*.

[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. of KDD 2000*.

[9] G. Fung, S. Sandilya, and R. Rao. Rule extraction from linear support vector machines. In *Proc. of KDD 2005*.

[10] J. Gao, W. Fan, and J. Han. On appropriate assumptions to mine data streams: analysis and practice. In *Proc. of ICDM 2007*.

[11] R. Guting. An introduction to spatial database systems. *VLDB Journal*, 3(4), 1994.

[12] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. of SIGMOD 1984*.

[13] B. Huysmans and J. Vanthienen. Using rule extraction to improve the comprehensibility of predictive models. *Technical report, FETEW Research Report KBI-0612, K.U.Leuven*, 2006.

[14] H.Wang, W. Fan, P. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. of KDD 2003*.

[15] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proc. of VLDB 1994*, pages 500–509.

[16] J. Kolter and M. Maloof. Using additive expert ensembles to cope with concept drift. In *Proc. of ICML 2005*.

[17] Z. Liu, S. Parthasarathy, A. Ranganathan, and H. Yang. Near-optimal algorithms for shared filter evaluation in data stream systems. In *Proc. of SIGMOD 2008*, pages 133–145.

[18] Z. Lu, X. Wu, X. Zhu, and J. Bongard. Ensemble pruning via individual contribution ordering. In *Proc. of KDD 2010*, pages 871–880.

[19] J. Ma, L. Saul, S. Savage, and G. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *Proc. of ICML 2009*.

[20] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram. Scalable ranked publish/subscribe. In *Proc. of VLDB 2008*.

[21] C. B. Philipp Kranen, Ira Assent and T. Seidl. The clustree: indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems*, 2010.

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

[23] T. Sellis, C. N. Roussopoulos, and Faloutsos. The r $^+$-tree: A dynamic index for multi-dimensional objects. In *Proc. of VLDB 1997*.

[24] T. Sellis, N. Roussopoulos, and C. Faloutsos. Multi-dimensional access methods: Trees have grown everywhere. In *Proc. of VLDB 1997*.

[25] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r*-tree: An efficient and robust access method for points and rectangles. In *Proc. of SIGMOD 1990*.

[26] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proc. of KDD 2001*.

[27] A. Tsymbal. The problem of concept drift: definitions and related work. *Available online: http://scss.tcd.ie/publications/tech-reports/reports.04/TCD-CS-2004-15.pdf*.

[28] S. Whang and H. Molina. Indexing boolean expressions. In *Proc. of VLDB 2009*.

[29] D. White and R. Jain. Similarity indexing with the ss-tree. In *Proc. of IEEE ICDE 1996*.

[30] P. Winston. Available online: www2.cs.uregina.ca/ dbd/cs831/notes/ml/dtrees/c4.5/tutorial.html. 1992.

[31] P. Zhang, X.Zhu, Y. Shi, L.Guo, and X. Wu. Robust ensemble learning for mining noisy data streams. *Decision Support Systems*, 50(2):469–479, 2011.

[32] P. Zhang, X. Zhu, and Y. Shi. Categorizing and mining concept drifting data streams. In *Proc. of KDD 2008*.

[33] P. Zhang, X. Zhu, J. Tan, and L. Guo. Classifier and cluster ensembles for mining concept drifting data streams. In *Proc. of IEEE ICDM 2010*.

[34] Y. Zhang, S. Burer, and W. Street. Ensemble pruning via semi-definite programming. *Journal of Machine Learning Research*, 7(2006):1315–1338, 2006.

[35] X. Zhu, P. Zhang, X. Lin, and Y. Shi. Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on System, Man, Cybernetics, Part B*, 40(4):1–15, 2010.

[36] X. Zhu, P. Zhang, X. Wu, D. He, C. Zhang, and Y. Shi. Cleansing noisy data streams. In *Proc. of ICDM 2008*, pages 1139 – 1144.