the tree. More recently, SURPASS [47] makes use of linear discriminants during the recursive partitioning process. The summary statistics (like AVC tables) are obtained incrementally. Rather than using summary statistics, [74] samples the training data, with confidence levels determined by PAC learning theory.

4.4.3 Parallel Decision Tree Construction

Several studies have sought to further speed up the decision tree construction using parallel machines. One of the first such studies is by Zaki et al. [88], who develop a shared memory parallelization of the SPRINT algorithm on disk-resident datasets. In parallelizing SPRINT, each attribute list is assigned to a separate processor. Also, Narlikar has used a fine-grained threaded library for parallelizing a decision tree algorithm [57], but the work is limited to memory-resident datasets. A shared memory parallelization has been proposed for the RF-read algorithm [38].

The SPIES approach has been parallelized [39] using a middleware system, FREERIDE (Framework for Rapid Implementation of Datamining Engines) [36, 37], which supports both distributed and shared memory parallelization on disk-resident datasets. FREERIDE was developed in early 2000 and can be considered an early prototype of the popular MapReduce/Hadoop system [16]. It is based on the observation that a number of popular data mining algorithms share a relatively similar structure. Their common processing structure is essentially that of *generalized reductions*. During each phase of the algorithm, the computation involves reading the data instances in an arbitrary order, processing each data instance (similar to *Map* in MapReduce), and updating elements of a *Reduction object* using associative and commutative operators (similar to *Reduce* in MapReduce).

In a distributed memory setting, such algorithms can be parallelized by dividing the data items among the processors and replicating the reduction object. Each node can process the data items it owns to perform a local reduction. After local reduction on all processors, a global reduction is performed. In a shared memory setting, parallelization can be done by assigning different data items to different threads. The main challenge in maintaining the correctness is avoiding race conditions when different threads may be trying to update the same element of the reduction object. FREERIDE has provided a number of techniques for avoiding such race conditions, particularly focusing on the memory hierarchy impact of the use of locking. However, if the size of the reduction object is relatively small, race conditions can be avoided by simply replicating the reduction object.

The key observation in parallelizing the SPIES-based algorithm is that construction of each type of AVC group, i.e., small, concise, and partial, essentially involves a reduction operation. Each data item is read, and the class histograms for appropriate AVC sets are updated. The order in which the data items are read and processed does not impact the final value of AVC groups. Moreover, if separate copies of the AVC groups are initialized and updated by processing different portions of the data set, a final copy can be created by simply adding the corresponding values from the class histograms. Therefore, this algorithm can be easily parallelized using the FREERIDE middleware system.

More recently, a general strategy was proposed in [11] to transform centralized algorithms into algorithms for learning from distributed data. Decision tree induction is demonstrated as an example, and the resulting decision tree learned from distributed data sets is identical to that obtained in the centralized setting. In [4] a distributed hierarchical decision tree algorithm is proposed for a group of computers, each having its own local data set. Similarly, this distributed algorithm induces the same decision tree that would come from a sequential algorithm with full data on each computer. Two univariate decision tree algorithms, C4.5 and univariate linear discriminant tree, are parallelized in [87] in three ways: feature-based, node-based, and data-based. Fisher's linear discriminant function is the basis for a method to generate a multivariate decision tree from distributed data [59]. In [61] MapReduce is employed for massively parallel learning of tree ensembles. Ye et al. [86] take

FIGURE 4.5: Illustration of streaming data.

on the challenging task of combining bootstrapping, which implies sequential improvement, with distributed processing.

4.5 Incremental Decision Tree Induction

Non-incremental decision tree learning methods assume that the training items can be accommodated simultaneously in the main memory or disk. This assumption grieviously limits the power of decision trees when dealing with the following situations: 1) training data sets are too large to fit into the main memory or disk, 2) the entire training data sets are not available at the time the tree is learned, and 3) the underlying distribution of training data sets is changed. Therefore, incremental decision tree learning methods have received much attention from the very beginning [68]. In this section, we examine the techniques for learning decision tree incrementally, especially in a streaming data setting.

Streaming data, represented by an endless sequence of data items, often arrive at high rates. Unlike traditional data available for batch (or off-line) prcessing, the labeled and unlabeled items are mixed together in the stream as shown in Figure 4.5.

In Figure 4.5, the shaded blocks are labeled records. We can see that labeled items can arrive unexpectedly. Therefore, this situation proposes new requirements for learning algorithms from streaming data, such as iterative, single pass, any-time learning [23].

To learn decision trees from streaming data, there are two main strategies: a greedy approach [14, 68, 78–80] and a statistical approach [19, 33]. In this section, we introduce both approaches, which are illustrated by two famous families of decision trees, respectively: *ID3* and *VFDT*.

4.5.1 ID3 Family

Incremental induction has been discussed almost from the start. Schlimmer [68] considers incremental concept induction in general, and develops an incremental algorithm named ID4 with a modification of Quinlan's top-down ID3 as a case study. The basic idea of ID4 is listed in Algorithm 4.3.

In Algorithm 4.3, A_v stands for all the attributes contained in tree node v, and A_v^* for the attribute with the lowest E-score. Meanwhile count $n_{ijy}(v)$ records the number of records observed by node v having value x_{ij} for attribute A_i and being in class y. In [68], the authors only consider positive and negative classes. That means $|\mathbf{Y}| = 2$. v_r stands for the immediate child of v containing item r.

Here, the E-score is the result of computing Quinlan's expected information function E of an attribute at any node. Specificially, at node v,

- n^p: # positive records;
- n^n: # negative records;
- n_{ij}^p: # positive records with value x_{ij} for attribute A_i;
- n_{ij}^n: # negative records with value x_{ij} for attribute A_i;

Algorithm 4.3 ID4(v, r)

Input: v: current decision tree node;
Input: r: data record $r = \langle x, y \rangle$;
1: **for** each $A_i \in \mathbf{A}_v$, where $\mathbf{A}_v \subseteq \mathbf{A}$ **do**
2: Increment count $n_{ijy}(v)$ of class y for A_i;
3: **end for**
4: **if** all items observed by v are from class y **then**
5: **return**;
6: **else**
7: **if** v is a leaf node **then**
8: Change v to be an internal node;
9: Update A_v^* with lowest E-score;
10: **else if** A_v^* does not have the lowest E-score **then**
11: Remove all children $Child(v)$ of v;
12: Update A_v^*;
13: **end if**
14: **if** $Child(v) = \emptyset$ **then**
15: Generate the set $Child(v)$ for all values of attribute A_v^*;
16: **end if**
17: **ID4**(v_r, r);
18: **end if**

Then

$$E(A_i) = \sum_{j=1}^{|A_i|} \frac{n_{ij}^p + n_{ij}^n}{n^p + n^n} I(n_{ij}^p, n_{ij}^n), \tag{4.20}$$

with

$$I(x,y) = \begin{cases} 0 & \text{if } x = 0 \text{ or } y = 0 \\ -\frac{x}{x+y}\log\frac{x}{x+y} - \frac{y}{x+y}\log\frac{y}{x+y} & \text{otherwise.} \end{cases}$$

In Algorithm 4.3, we can see that whenever an erroneous splitting attribute is found at v (Line 10), ID4 simply removes all the subtrees rooted at v's immediate children (Line 11), and computes the correct splitting attribute A_v^* (Line 12).

Clearly ID4 is not efficient because it removes the entire subtree when a new A_v^* is found, and this situation could render certain concepts unlearnable by ID4, which could be induced by ID3. Utgoff introduced two improved algorithms: ID5 [77] and ID5R [78]. In particular, ID5R guarantees it will produce the same decision tree that ID3 would have if presented with the same training items.

In Algorithm 4.4, when a splitting test is needed at node v, an arbitrary attribute $A_o \in \mathbf{A}_v$ is chosen; further, according to counts $n_{ijy}(v)$ the optimal splitting attribute A_v^* is calculated based on E-score. If $A_v^* \neq A_o$, the splitting attribute A_v^* is pulled up from all its subtrees (Line 10) to v, and all its subtrees are recursively updated similarly (Line 11 and 13).

The fundamental difference between ID4 (Algorithm 4.3) and ID5R (Algorithm 4.4) is that when ID5R finds a wrong subtree, it restructures the subtree (Line 11 and 13) instead of discarding it and replacing it with a leaf node for the current splitting attribute. The restructuring process in Algorithm 4.4 is called the *pull-up* procedure. The general pull-up procedure is as follows, and illustrated in Figure 4.6. In Figure 4.6, left branches satisfy the splitting tests, and right ones do not.

1. if attribute A_v^* is already at the root, then stop;
2. otherwise,

Algorithm 4.4 ID5R(v, r)

Input: v: the current decision tree node;
Input: r: the data record $r = \langle x, y \rangle$;

```
If v = null, make v a leaf node; return;
If v is a leaf, and all items observed by v are from class y; return;
if v is a leaf node then
    Split v by choosing an arbitrary attribute;
end if
for A_i ∈ A_v, where A_v ⊆ A do
    Increment count n_ijy(v) of class y for A_i;
end for
if A_v^* does not have the lowest E-score then
    Update A_v^*, and restructure v;
    Recursively reestablish v_c ∈ Child(v) except the branch v_r for r;
end if
Recursively update subtree v_r along the branches with the value occuring in x;
```

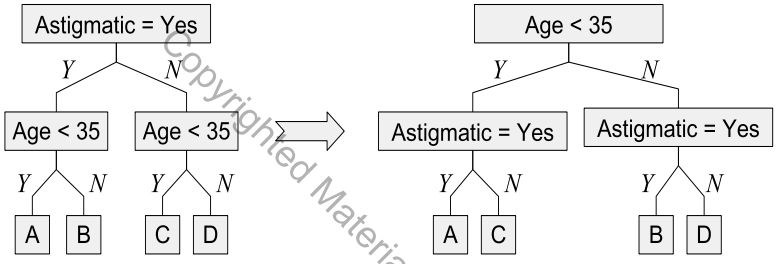

FIGURE 4.6: Subtree restructuring.

(a) Recursively pull the attribute A_v^* to the root of each immediate subtree of v. Convert any leaves to internal nodes as necessary, choosing A_v^* as splitting attribute.

(b) Transpose the subtree rooted at v, resulting in a new subtree with A_v^* at the root, and the old root attribute A_o at the root of each immediate subtree of v.

There are several other works that fall into the ID3 family. A variation for multivariate splits appears in [81], and an improvement of this work appears in [79], which is able to handle numerical attributes. Having achieved an arguably efficient technique for incrementally restructuring a tree, Utgoff applies this technique to develop Direct Metric Tree Induction (DMTI). DMTI leverages fast tree restructuring to fashion an algorithm that can explore more options than traditional greedy top-down induction [80]. Kalles [42] speeds up ID5R by estimating the minimum number of training items for a new attribute to be selected as the splitting attribute.

4.5.2 VFDT Family

In the Big Data era, applications that generate vast streams of data are ever-present. Large retail chain stores like Walmart produce millions of transaction records every day or even every hour, giant telecommunication companies connect millions of calls and text messages in the world, and large banks receive millions of ATM requests throughout the world. These applications need machine learning algorithms that can learn from extremely large (probably infinite) data sets, but spend only a small time with each record. The VFDT learning system was proposed by Domingos [19] to handle this very situation.

VFDT (Very Fast Decision Tree learner) is based on the *Hoeffding tree*, a decision tree learning method. The intuition of the Hoeffding tree is that to find the best splitting attribute it is sufficient to consider only a small portion of the training items available at a node. To acheive this goal, the *Hoeffding bound* is utilized. Basically, given a real-valued random variable r having range R, if we have observed n values for this random variable, and the sample mean is $\bar{r}$, then the Hoeffding bound states that, with probability $1-\delta$, the true mean of r is at least $\bar{r}-\varepsilon$, where

$$\varepsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}. \tag{4.21}$$

Based on the above analysis, if at one node we find that $\bar{F}(A_i) - \bar{F}(A_j) \geq \varepsilon$, where $\bar{F}$ is the splitting criterion, and A_i and A_j are the two attributes with the best and second best $\bar{F}$ respectively, then A_i is the correct choice with probability $1-\delta$. Using this novel observation, the Hoeffding tree algorithm is developed (Algorithm 4.5).

Algorithm 4.5 HoeffdingTree(S, $\mathbf{A}$, F, δ)

Input: S: the streaming data;
Input: $\mathbf{A}$: the set of attributes;
Input: F: the split function;
Input: δ: $1-\delta$ is the probability of choosing the correct attribute to split;
Output: T: decision tree

```
1:  Let T be a tree with a single leaf v_1;
2:  Let A_1 = A;
3:  Set count n_ijy(v_1) = 0 for each y ∈ Y, each value x_ij of each attribute A_i ∈ A
4:  for each training record ⟨x,y⟩ in S do
5:      Leaf v = classify(⟨x,y⟩, T);
6:      For each x_ij ∈ x: Increment count n_ijy(v);
7:      if n_ijy(v) does not satisfy any stop conditions then
8:          A_v^* = F(n_ijy(v), δ);
9:          Replace v by an internal node that splits on A_v^*;
10:         for each child v_m of v do
11:             Let A_m = A_v − {A_v^*};
12:             Initialize n_ijy(v_m);
13:         end for
14:     end if
15: end for
16: return T;
```


In Algorithm 4.5, the n_{ijy} counts are sufficient to calculate $\bar{F}$. Initially decision tree T only contains a leaf node v_1(Line 1), and v_1 is labeled by predicting the most frequent class. For each item $\langle x,y\rangle$, it is first classified into a leaf node v through T (Line 5). If the items in v are from more than one class, then v is split according to the *Hoeffding* bound (Line 8). The key property of the Hoeffding tree is that under realistic assumptions (see [19] for details), it is possible to guarantee that the generated tree is asymptotically close to the one produced by a batch learner.

When dealing with streaming data, one practical problem that needs considerable attention is concept drift, which does not satisfy the assumption of VFDT: that the sequential data is a random sample drawn from a *stationary* distribution. For example, the behavior of customers of online shopping may change from weekdays to weekends, from season to season. CVFDT [33] has been developed to deal with concept drift.

CVFDT utilizes two strategies: a sliding window W of training items, and alternate subtrees $ALT(v)$ for each internal node v. The decision tree records the statistics for the $|W|$ most recent

unique training items. More specifically, instead of learning a new model from scratch when a new training item $\langle x,y\rangle$ comes, CVFDT increments the sufficient statistics n_{ijy} at corresponding nodes for the new item and decrements the counts for the oldest records $\langle x_o,y_o\rangle$ in the window. Periodically, CVFDT reevaluates the classification quality and replaces a subtree with one of the alternate subtrees if needed.

Algorithm 4.6 CVFDT(S, $\mathbf{A}$, F,δ, w, f)

Input: S: the streaming data;
Input: $\mathbf{A}$: the set of attributes;
Input: F: the split function;
Input: δ: $1-\delta$ is the probability of choosing the correct attribute to split;
Input: w: the size of window;
Input: f: # training records between checks for drift;
Output: T: decision tree;

1: let T be a tree with a single leaf v_1;
2: let $ALT(v_1)$ be an initially empty set of alternate trees for v_1;
3: let $\mathbf{A}_1 = \mathbf{A}$;
4: Set count $n_{ijy}(v_1) = 0$ for each $y \in \mathbf{Y}$ and each value x_{ij} for $A_i \in \mathbf{A}$;
5: Set record window $W = \emptyset$;
6: **for** each training record $\langle x,y\rangle$ in S **do**
7: $L = classify(\langle x,y\rangle, T)$, where L contains all nodes that $\langle x,y\rangle$ passes through using T and all trees in ALT;
8: $W = W \cup \{\langle x,y\rangle\}$;
9: **if** $|W| > w$ **then**
10: let $\langle x_o,y_o\rangle$ be the oldest element of W;
11: ForgetExample(T,ALT,$\langle x_o,y_o\rangle$);
12: let $W = W \setminus \{\langle x_o,y_o\rangle\}$;
13: **end if**
14: CVFDTGrow(T,L,$\langle x,y\rangle$,δ);
15: if there have been f examples since the last checking of alternate trees
16: CheckSplitValidity(T,δ);
17: **end for**
18: **return** T;

An outline of CVFDT is shown in Algorithm 4.6. When a new record $\langle x,y\rangle$ is received, we classify it according to the current tree. We record in a structure L every node in the tree T and in the alternate subtrees ALT that are encountered by $\langle x,y\rangle$ (Line 7). Lines 8 to 14 keep the sliding window up to date. If the tree's number of data items has now exceeded the maximum window size (Line 9), we remove the oldest data item from the statistics (Line 11) and from W (Line 12). *ForgetExample* traverses the decision tree and decrements the corresponding counts n_{ijy} for $\langle x_o,y_o\rangle$ in any node of T or ALT. We then add $\langle x,y\rangle$ to the tree, increasing n_{ijy} statistics according to L (Line 14). Finally, once every f items, we invoke Procedure *CheckSplitValidity*, which scans T and ALT looking for better splitting attributes for each internal node. It revises T and ALT as necessary.

Of course, more recent works can be found following this family. Both VFDT and CVFDT only consider discrete attributes; the VFDTc [26] system extends VFDT in two major directions: 1) VFDTc is equipped with the ability to deal with numerical attributes; and 2) a naïve Bayesian classifier is utilized in each leaf. Jin [40] presents a numerical interval pruning (NIP) approach to efficiently handle numerical attributes, and speeds up the algorithm by reducing the sample size. Further, Bifet [6] proposes a more efficient decision tree learning method than [26] by replacing naïve Bayes with perceptron classifiers, while maintaining competitive accuracy. Hashemi [32] de-

velops a flexible decision tree (FlexDT) based on fuzzy logic to deal with noise and missing values in streaming data. Liang [48] builds a decision tree for uncertain streaming data.

Notice that there are some general works on handling concept drifting for streaming data. Gama [27, 28] detects drifts by tracing the classification errors for the training items based on PAC framework.

4.5.3 Ensemble Method for Streaming Data

An ensemble classifier is a collection of several base classifiers combined together for greater prediction accuracy. There are two well-known ensemble learning approaches: *Bagging* and *Boosting*.

Online bagging and boosting algorithms are introduced in [60]. They rely on the following observation: The probability for each base model to contain each of the original training items k times follows the binomial distribution. As the number of training items goes to infinity, k follows the *Possion(1)* distribution, and this assumption is suitable for streaming data.

In [73], Street et al. propose an ensemble method to read training items sequentially as blocks. When a new training block D comes, a new classifier C_i is learned, and C_i will be evaluated by the next training block. If the ensemble committee E is not full, C_i is inserted into E; otherwise, C_i could replace some member classifier C_j in E if the quality of C_i is better than that of C_j. However, both [73] and [60] fail to explicitly take into consideration the concept drift problem.

Based on Tumer's work [76], Wang et al. [84] prove that ensemble classifier E produces a smaller error than a single classifier $G \in E$, if all the classifiers in E have weights based on their expected classification accuracy on the test data. Accordingly they propose a new ensemble classification method that handles concept drift as follows: When a new chunk D of training items arrives, not only is a new classifier C trained, but also the weights of the previously trained classifiers are recomputed.

They propose the following training method for classifiers on streaming chunks of data, shown in Algorithm 4.7.

Algorithm 4.7 EnsembleTraining(S, K, $\mathcal{C}$)

Input: S: a new chunk of data;
Input: K: the number of classifiers;
Input: $\mathcal{C}$: the set of previously trained classifiers;
1: Train a new classifier $C^{'}$ based on S;
2: Compute the weight $w^{'}$ for $C^{'}$;
3: **for** each $C_i \in \mathcal{C}$ **do**
4: Recompute the weight w_i for C_i based on S;
5: **end for**
6: $\mathcal{C} \leftarrow$ top K weighted classifiers from $\mathcal{C} \cup \{C^{'}\}$;

In Algorithm 4.7, we can see that when a new chunk S arrives, not only a new classifier $C^{'}$ is trained, but also the weights of the previous trained classifiers are recomputed in this way to handle the concept drifting.

Kolter et al. [45] propose another ensemble classifier to detect concept drift in streaming data. Similar to [84], their method dynamically adjusts the weight of each base classifier according to its accuracy. In contrast, their method has a weight parameter threshold to remove bad classifiers and trains a new classifier for the new item if the existing ensemble classifier fails to identity the correct class.

Fan [21] notices that the previous works did not answer the following questions: When would the old data help detect concept drift and which old data would help? To answer these questions, the

author develops a method to sift the old data and proposes a simple cross-validation decision tree ensemble method.

Gama [29] extends the Hoeffding-based Ultra Fast Forest of Trees (UFFT) [25] system to handle concept drifting in streaming data. In a similar vein, Abulsalam [1] extends the random forests ensemble method to run in amortized $O(1)$ time, handles concept drift, and judges whether a sufficient quantity of labeled data has been received to make reasonable predictions. This algorithm also handles multiple class values. Bifet [7] provides a new experimental framework for detecting concept drift and two new variants of bagging methods: ADWIN Bagging and Adaptive-Size Hoeffding Tree (ASHT). In [5], Bifet et al. combine Hoeffding trees using stacking to classify streaming data, in which each Hoeffding tree is built using a subset of item attributes, and ADWIN is utilized both for the perceptron meta-classifier for resetting learning rate and for the ensemble members to detect concept drifting.

4.6 Summary

Compared to other classification methods [46], the following stand out as advantages of decision trees:

- Easy to interpret. A small decision tree can be visualized, used, and understood by a layperson.
- Handling both numerical and categorical attributes. Classification methods that rely on weights or distances (neural networks, k-nearest neighbor, and support vector machines) do not directly handle categorical data.
- Fast. Training time is competitive with other classification methods.
- Tolerant of missing values and irrelevant attributes.
- Can learn incrementally.

The shortcomings tend to be less obvious and require a little more explanation. The following are some weaknesses of decision trees:

- Not well-suited for multivariate partitions. Support vector machines and neural networks are particularly good at making discriminations based on a weighted sum of all the attributes. However, this very feature makes them harder to interpret.
- Not sensitive to relative spacing of numerical values. Earlier, we cited decision trees' ability to work with either categorical or numerical data as an advantage. However, most split criteria do not use the numerical values directly to measure a split's goodness. Instead, they use the values to sort the items, which produces an ordered sequence. The ordering then determines the candidate splits; a set of n ordered items has $n-1$ splits.
- Greedy approach may focus too strongly on the training data, leading to overfit.
- Sensitivity of induction time to data diversity. To determine the next split, decision tree induction needs to compare every possible split. As the number of different attribute values increases, so does the number of possible splits.

Despite some shortcomings, the decision tree continues to be an attractive choice among classification methods. Improvements continue to be made: more accurate and robust split criteria, ensemble methods for even greater accuracy, incremental methods that handle streaming data and concept drift, and scalability features to handle larger and distributed datasets. A simple concept that began well before the invention of the computer, the decision tree remains a valuable tool in the machine learning toolkit.

Bibliography

[1] Hanady Abdulsalam, David B. Skillicorn, and Patrick Martin. Classification using streaming random forests. *IEEE Transactions on Knowledge and Data Engineering*, 23(1):22–36, 2011.

[2] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. Clouds: A decision tree classifier for large datasets. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, pages 2–8. AAAI, 1998.

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

[4] Amir Bar-Or, Assaf Schuster, Ran Wolff, and Daniel Keren. Decision tree induction in high dimensional, hierarchically distributed databases. In *Proceedings of the Fifth SIAM International Conference on Data Mining*, SDM'05, pages 466–470. SIAM, 2005.

[5] Albert Bifet, Eibe Frank, Geoffrey Holmes, and Bernhard Pfahringer. Accurate ensembles for data streams: Combining restricted hoeffding trees using stacking. *Journal of Machine Learning Research: Workshop and Conference Proceedings*, 13:225–240, 2010.

[6] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Eibe Frank. Fast perceptron decision tree learning from evolving data streams. *Advances in Knowledge Discovery and Data Mining*, pages 299–310, 2010.

[7] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'09, pages 139–148. ACM, 2009.

[8] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[9] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, 19(1):45–77, 1995.

[10] Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[11] Doina Caragea, Adrian Silvescu, and Vasant Honavar. A framework for learning from distributed data using sufficient statistics and its application to learning decision trees. *International Journal of Hybrid Intelligent Systems*, 1(1):80–89, 2004.

[12] Xiang Chen, Minghui Wang, and Heping Zhang. The use of classification trees for bioinformatics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):55–63, 2011.

[13] David A. Cieslak and Nitesh V. Chawla. Learning decision trees for unbalanced data. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, ECML PKDD'08, pages 241–256. Springer, 2008.

[14] S. L. Crawford. Extensions to the cart algorithm. *International Journal of Man-Machine Studies*, 31(2):197–217, September 1989.

[15] Barry De Ville. *Decision Trees for Business Intelligence and Data Mining: Using SAS Enterprise Miner*. SAS Institute Inc., 2006.

[16] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. Originally presented at OSDI '04: 6th Symposium on Operating Systems Design and Implementation.

[17] Tom Dietterich, Michael Kearns, and Yishay Mansour. Applying the weak learning framework to understand and improve C4.5. In *Proceedings of the Thirteenth International Conference on Machine Learning*, ICML'96, pages 96–104. Morgan Kaufmann, 1996.

[18] Yufeng Ding and Jeffrey S. Simonoff. An investigation of missing data methods for classification trees applied to binary response data. *The Journal of Machine Learning Research*, 11:131–170, 2010.

[19] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'00, pages 71–80. ACM, 2000.

[20] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and J. Kay. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

[21] Wei Fan. Systematic data selection to mine concept-drifting data streams. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 128–137. ACM, 2004.

[22] Usama M. Fayyad and Keki B. Irani. The attribute selection problem in decision tree generation. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 104–110. AAAI Press, 1992.

[23] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. The MIT Press, February 1996.

[24] Jerome H. Friedman. A recursive partitioning decision rule for nonparametric classification. *IEEE Transactions on Computers*, 100(4):404–408, 1977.

[25] João Gama, Pedro Medas, and Ricardo Rocha. Forest trees for on-line data. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC'04, pages 632–636. ACM, 2004.

[26] João Gama, Ricardo Rocha, and Pedro Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'03, pages 523–528. ACM, 2003.

[27] João Gama and Gladys Castillo. Learning with local drift detection. In *Advanced Data Mining and Applications*, volume 4093, pages 42–55. Springer-Verlag, 2006.

[28] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. *Advances in Artificial Intelligence–SBIA 2004*, pages 66–112, 2004.

[29] João Gama, Pedro Medas, and Pedro Rodrigues. Learning decision trees from dynamic data streams. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC'05, pages 573–577. ACM, 2005.

[30] Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh. Boat– optimistic decision tree construction. *ACM SIGMOD Record*, 28(2):169–180, 1999.

[31] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. Rainforest — A framework for fast decision tree construction of large datasets. In *Proceedings of the International Conference on Very Large Data Bases*, VLDB'98, pages 127–162, 1998.

[32] Sattar Hashemi and Ying Yang. Flexible decision tree for data stream classification in the presence of concept change, noise and missing values. *Data Mining and Knowledge Discovery*, 19(1):95–131, 2009.

[33] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'01, pages 97–106. ACM, 2001.

[34] Earl Busby Hunt, Janet Marin, and Philip J. Stone. *Experiments in Induction*. Academic Press, New York, London, 1966.

[35] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[36] Ruoming Jin and Gagan Agrawal. A middleware for developing parallel data mining implementations. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, SDM'01, April 2001.

[37] Ruoming Jin and Gagan Agrawal. Shared memory parallelization of data mining agorithms: Techniques, programming interface, and performance. In *Proceedings of the Second SIAM International Conference on Data Mining*, SDM'02, pages 71–89, April 2002.

[38] Ruoming Jin and Gagan Agrawal. Shared memory parallelization of decision tree construction using a general middleware. In *Proceedings of the 8th International Euro-Par Parallel Processing Conference*, Euro-Par'02, pages 346–354, Aug 2002.

[39] Ruoming Jin and Gagan Agrawal. Communication and memory efficient parallel decision tree construction. In *Proceedings of the Third SIAM International Conference on Data Mining*, SDM'03, pages 119–129, May 2003.

[40] Ruoming Jin and Gagan Agrawal. Efficient decision tree construction on streaming data. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'03, pages 571–576. ACM, 2003.

[41] Mahesh V. Joshi, George Karypis, and Vipin Kumar. Scalparc: A new scalable and efficient parallel classification algorithm for mining large datasets. In *First Merged Symp. IPPS/SPDP 1998: 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing*, pages 573–579. IEEE, 1998.

[42] Dimitrios Kalles and Tim Morris. Efficient incremental induction of decision trees. *Machine Learning*, 24(3):231–242, 1996.

[43] Michael Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, STOC'96, pages 459–468. ACM, 1996.

[44] Michael Kearns and Yishay Mansour. A fast, bottom-up decision tree pruning algorithm with near-optimal generalization. In *Proceedings of the 15th International Conference on Machine Learning*, pages 269–277, 1998.

[45] Jeremy Z. Kolter and Marcus A. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proceedings of the Third IEEE International Conference on Data Mining, 2003.*, ICDM'03, pages 123–130. IEEE, 2003.

[46] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 Conference on Emerging Artifical Intelligence Applications in Computer Engineering*, pages 3–24. IOS Press, 2007.

[47] Xiao-Bai Li. A scalable decision tree system and its application in pattern recognition and intrusion detection. *Decision Support Systems*, 41(1):112–130, 2005.

[48] Chunquan Liang, Yang Zhang, and Qun Song. Decision tree for dynamic and uncertain data streams. In *2nd Asian Conference on Machine Learning*, volume 3, pages 209–224, 2010.

[49] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228, 2000.

[50] Christiane Ferreira Lemos Lima, Francisco Marcos de Assis, and Cleonilson Protásio de Souza. Decision tree based on shannon, rényi and tsallis entropies for intrusion tolerant systems. In *Proceedings of the Fifth International Conference on Internet Monitoring and Protection*, ICIMP'10, pages 117–122. IEEE, 2010.

[51] R. López de Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6(1):81–92, 1991.

[52] Yishay Mansour. Pessimistic decision tree pruning based on tree size. In *Proceedings of the 14th International Conference on Machine Learning*, pages 195–201, 1997.

[53] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. Sliq: A fast scalable classifier for data mining. In *Proceedings of the Fifth International Conference on Extending Database Technology*, pages 18–32, Avignon, France, 1996.

[54] John Mingers. Expert systems—Rule induction with statistical data. *Journal of the Operational Research Society*, 38(1): 39–47, 1987.

[55] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227–243, 1989.

[56] James N. Morgan and John A. Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58(302):415–434, 1963.

[57] G. J. Narlikar. A parallel, multithreaded decision tree builder. Technical Report CMU-CS-98-184, School of Computer Science, Carnegie Mellon University, 1998.

[58] Tim Niblett. Constructing decision trees in noisy domains. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning*. Sigma, 1987.

[59] Jie Ouyang, Nilesh Patel, and Ishwar Sethi. Induction of multiclass multifeature split decision trees from distributed data. *Pattern Recognition*, 42(9):1786–1794, 2009.

[60] Nikunj C. Oza and Stuart Russell. Online bagging and boosting. In *Eighth International Workshop on Artificial Intelligence and Statistics*, pages 105–112. Morgan Kaufmann, 2001.

[61] Biswanath Panda, Joshua S. Herbach, Sugato Basu, and Roberto J. Bayardo. Planet: Massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2(2):1426–1437, 2009.

[62] J. Ross Quinlan. Learning efficient classification procedures and their application to chess end-games. In *Machine Learrning: An Artificial Intelligence Approach*. Tioga Publishing Company, 1983.

[63] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.

[64] J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[65] J. Ross Quinlan and Ronald L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computing*, 80:227–248, 1989.

[66] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[67] Maytal Saar-Tsechansky and Foster Provost. Handling missing values when applying classification models. *Journal of Machine Learning Research*, 8:1623–1657, 2007.

[68] Jeffrey C. Schlimmer and Douglas Fisher. A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 495–501. Morgan Kaufmann, 1986.

[69] John Shafer, Rakeeh Agrawal, and Manish Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB)*, pages 544–555, September 1996.

[70] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, July/October 1948.

[71] Harold C. Sox and Michael C. Higgins. *Medical Decision Making*. Royal Society of Medicine, 1988.

[72] Anurag Srivastava, Eui-Hong Han, Vipin Kumar, and Vineet Singh. Parallel formulations of decision-tree classification algorithms. In *Proceedings of the 1998 International Conference on Parallel Processing*, ICPP'98, pages 237–261, 1998.

[73] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'01, pages 377–382. ACM, 2001.

[74] Hyontai Sug. A comprehensively sized decision tree generation method for interactive data mining of very large databases. In *Advanced Data Mining and Applications*, pages 141–148. Springer, 2005.

[75] Umar Syed and Golan Yona. Using a mixture of probabilistic decision trees for direct prediction of protein function. In *Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology*, RECOMB'03, pages 289–300. ACM, 2003.

[76] Kagan Tumer and Joydeep Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3-4):385–404, 1996.

[77] Paul E. Utgoff. Id5: An incremental id3. In *Proceedings of the Fifth International Conference on Machine Learning*, ICML'88, pages 107–120, 1988.

[78] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.

[79] Paul E. Utgoff. An improved algorithm for incremental induction of decision trees. In *Proceedings of the Eleventh International Conference on Machine Learning*, ICML'94, pages 318–325, 1994.

[80] Paul E Utgoff, Neil C Berkman, and Jeffery A Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

[81] Paul E. Utgoff and Carla E. Brodley. An incremental method for finding multivariate splits for decision trees. In *Proceedings of the Seventh International Conference on Machine Learning*, ICML'90, pages 58–65, 1990.

[82] Paul A. J. Volf and Frans M.J. Willems. Context maximizing: Finding mdl decision trees. In *Symposium on Information Theory in the Benelux*, volume 15, pages 192–200, 1994.

[83] Chris S. Wallace and J. D. Patrick. Coding decision trees. *Machine Learning*, 11(1):7–22, 1993.

[84] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235. ACM, 2003.

[85] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[86] Jerry Ye, Jyh-Herng Chow, Jiang Chen, and Zhaohui Zheng. Stochastic gradient boosted distributed decision trees. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM'09, pages 2061–2064. ACM, 2009.

[87] Olcay Taner Yıldız and Onur Dikmen. Parallel univariate decision trees. *Pattern Recognition Letters*, 28(7):825–832, 2007.

[88] M. J. Zaki, Ching-Tien Ho, and Rakesh Agrawal. Parallel classification for data mining on shared-memory multiprocessors. In *Proceedings of the Fifteenth International Conference on Data Engineering*, ICDE'99, pages 198–205, May 1999.