IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 1, 2019, accepted October 26, 2019, date of publication October 29, 2019, date of current version November 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2950374

New Associative Classification Method Based on Rule Pruning for Classification of Datasets

KHAIRAN D. RAJAB
College of Computer Science and Information System, Najran University, Najran 55461, Saudi Arabia
e-mail: khairanr@gmail.com

This work was supported by the Najran University under Grant NU/ESCI/16/009.

ABSTRACT In data mining, a rule-based classification approach called Associative Classification (AC) normally builds accurate classifiers from supervised learning data sets. It extracts ''If-Then'' rules and associates each of the generated rules with two computed parameters; support and confidence. These two parameters are utilized to differentiate the rules' superiority during the building of a classifier's step. In current AC algorithms, whenever a rule is inserted into a classifier, all of its corresponding training data is discarded. However, the discarded data actually are used to compute support and confidence of other rules and will affect other lower ranked rules since rules normally have common training data examples. The use of static support and confidence may result in very large less-accurate classifiers.Thus, a procedure that amends other rules' support and confidence is important. This paper proposes a new procedure named Active Pruning Rules (APR) to overcome the above problem so then the classifiers' performance - especially predictive accuracy and reducing rule redundancy - will be further improved. The experimental results obtained from a number of University of California Irvine (UCI) data sets and real adult autism classification data set showed that APR is highly competitive to other AC and rule-based classifiers and often produces smaller yet more predictive classifiers.

INDEX TERMS Association rule, associative classification, data mining, prediction, rule induction, rule pruning.

I. INTRODUCTION

Nowadays, businesses have numerous data sets scattered internally over numerous functioning units within organizations and externally - most commonly online. Automated smart software tools can be vital for the analysis of such large data sets in the decision-making process. Data mining, which is a multi-disciplinary research field, can be at the core of many of such needed software. It is based on mathematics, databases and other computer science topics and can help decision makers find useful information from their data sets. It is also involved in several tasks such as clustering, association rules, classification and regression.

In the last decade, the classification and association rules have been integrated to form a new research topic named associative classification (AC), [1]. AC primarily utilizes association rule discovery methods to train on an input data set in order to discover rules and then it adds on steps involving constructing the classifier and predicting the test data. Recently, AC has been utilized in several businesses and security applications, i.e., website phishing detection [2], fault prediction [3], recommendation systems [4], and text mining [5], etc.

Two distinguishing characteristics associated with the AC approach which expands on its use in applications are:

- The high predictive classification accuracy of its classifiers.
- The interpretability of the rules within the classifier.

Most AC algorithms induce the knowledge (rules) from the input data, and then construct a predictive model that is named the classifier. The two main parameters connected with each rule are support and confidence (see Definitions 6 and 8). The process of choosing the rules during the building of the classifier is the concern of this article. In this context, the current AC methods evaluate the induced rules on the input data set one by one after sorting them based on a rule ranking procedure, i.e., (rules' confidence, support, length). Starting from the highest ranked rule downward, each training data will be evaluated and covered by a single rule only. When a training example is covered by a certain rule, it will be discarded and that rule will become part of the classifier. The evaluation process continues until all of the rules have been

The associate editor coordinating the review of this manuscript and approving it for publication was Vijay Mago.

evaluated or no more data can be removed. Only at this time, is the classifier generated. The classifier will contain only rules that were covered in the training examples and all other rules are removed since they are redundant.

This paper investigates a major shortcoming associated with AC algorithms during the process of rule evaluation. Specifically, when removing the training examples connected with a classifier's rules whenever a rule gets evaluated against a training data set. This problem was raised by [6] in which the authors proposed a multi-label rules mechanism. We argue that when training examples are discarded after a rule is evaluated, this removal affects other lower ranked rules since rules normally have common training data examples (items inside the data). Current AC algorithms in the literature assume that a training example can be used to learn multiple rules. So these algorithms do not eliminate any overlap of the training data amongst the generated rules, resulting in generating very large classifiers. The aim of this research is to overcome the data overlapping issue in AC mining by developing a new learning method to ensure that rules derived have no data overlapping thus reducing the size of the classifier and maintain predictive accuracy performance.

The above issue occurs during the process of inducing the rules and should be resolved when the algorithm chooses the classifier's rules. Consider for instance rule R_1: "If y_1 and x_2 then Class A", that has been evaluated against the training data set, assuming that all training data examples of R_1 are deleted. All other lower ranked rules having the values ("y_1" and "x_2") in their body are impacted because of R_1's data removal. This means that these rules' confidence and support and support counts should now be updated to reflect R_1's data removal. When the training data set is frequently updated because of rule example removal, the rule rank should change as well during the process of the rule evaluation. If this is implemented, a fairer classifier that does not rely on static rules rank will result.

A new AC method that we call Active Pruning Rules (APR) is developed to resolve the discussed issue. For each rule inserted into the classifier, the algorithm not only deletes all of its connected training data but also removes said data from any other unevaluated rules. This results in decrementing the confidence and support values for some of the rules and changing the ranking of these rules before the next in-line rule is evaluated. This can be viewed as a rule-pruning procedure that discards insufficient rules without having to evaluate them against the training data as with current AC algorithms, saving computing resources. Furthermore, amending the rule rank every time a rule is inserted into the classifier may generate classifiers with no rule redundancy. Most importantly, the algorithm significantly increases the predictive accuracy of the classifier as the results from the experiments section shows.

Section II explains the AC definitions besides the raised research issue. Section III reviews the known rules related to pruning and AC methods and Section IV proposes the new algorithm and its primary steps. Section V provides a comprehensive example on the phases of the proposed algorithm, and Section VI describes the UCI data, experiments and result analysis. Section VII is devoted to experimental results on a behavioural application related to autism, and finally, conclusions are given in Section VIII.

II. RESEARCH ISSUES AND TERMS DEFINITIONS

For a predictive task such as classification, an input data set such as T contains a number of different variables (attributes) $At_1, At_2, \ldots, At_m$ plus a target variable named the class label, i.e., cl. The cardinality of T size is given by $|T|$. A variable can be continuous or discrete. Discrete variables are mapped to positive integers set, and any continuous variable is partitioned (discretized) before data processing. The goal of AC is to construct a predictive model from T that is able to accurately forecast the target variable in an unseen data called the test data set. The definitions below summarized the primary terms of AC as follows:

Definition 1: An item plus its value in T is called *attribute* value and denoted as (At_i, v_i).

Definition 2: A row in T is called a *training example* and is represented as $(At_{j1}, vj_1) \ldots (At_{jv}, v_{jv}), cl_j$ where cl_j is the target class value.

Definition 3: A rule r consists of <*antecedent, class*>, where *antecedent* is disjoint conjunctive items in a *class* value.

Definition 4: r's frequency *in T (r_freq)* is how many data examples in T similar to r.

Definition 5: antecedent's frequency *(antecedent_freq)* denotes the number of examples in T that have similar *antecedent* to a rule r.

Definition 6: The minimum support *(minSupp)* is a predefined threshold that gets inserted by the end user.

Definition 7: r survives *minSupp* when r's $|r\_freq| / |T| \geq minSupp$. A surviving rule is considered a strong rule.

Definition 8: r survives *minConf* when r's confidence $= |r\_freq|/|antecedent\_freq|$.

Definition 9: r is formatted in the classifier as: $at_1 = v_1 \wedge at_2 = v_2 \wedge \ldots \wedge at_n = v_n \rightarrow class_1$.

APR relies on the two thresholds as mentioned earlier; *minSupp* and *minConf*. The *minSupp* is set by the end-user to discriminate rules. Each potential rule frequency in the training data set from the size of that data set denotes its support (Definition 6). Any rule with support stronger than the *minSupp* is kept for further evaluation. On the other hand, *minConf* is a threshold that distinguishes between the strong and weak rules (those that are statistically not fit). A rule confidence represents the frequency of the rule, i.e., <*antecedent, class*>, from the frequency of the antecedent in the training data set (Definition 5). A rule passes the *minConf* when it has a larger computed confidence value according to Definition 8.

III. LITERATURE REVIEW

Association rule mining and classification are related tasks as the former tries to discover concealed correlations amongst

the attributes' values in a data set while the latter uses the attributes' values to build a model which in turn is utilized to predict the response variable of a test case. Therefore, integrating both tasks together produces Associative Classification (AC) [1], which employs association rule methods in extracting classifiers that contains ''IF-THEN'' rules.

Generally, there are three main steps that an AC algorithm performs:

- Learning the rules (Training). The algorithm processes the data to generate the rules.
- Rule ranking and pruning. The extracted potential rules are sorted based on certain criteria like rule confidence, support and length and others are then evaluated on the training data set to identify the ones contained within the classifier. During this step, any duplicate rules are removed.
- Classification of test data. In this step, the rules in the classifier are utilized to forecast the class values of unseen data (test data). During this step, the predictive power of the classifier is measured using prediction accuracy or error rate.

In the last decade, multiple research studies, for example [4], [7] have reported the applicability of AC on various different applications including fraud detection, credit card scoring, bioinformatics, on-line security, medical diagnoses, text categorization and others. The wide spread of this classification approach is primarily due to the simplicity and interpretability of the rules in the classifier besides the high predictive accuracy of its classifiers.

In the literature, many algorithms utilize a variety of knowledge-reasoning methodologies, rule pruning, and class predictions for test data. Since AC methods suffer from the exponential growth of the rules [8], this article tackles this problem and proposes a novel pruning method hence this section focuses more on the rule pruning procedures rather than on general AC algorithms in the literature. Rule pruning is the key to success in AC and ensures that any classifiers derived are controllable and usable by the end users. In fact, without rule pruning, most generated classification systems will not be useful and might get discarded because of the large number of rules in the classifier [9]. This might lead into difficulties in managing the classification systems and hence the managers and decision makers will not put these systems in practise.

One of the first developed AC algorithms is CBA [1]. This algorithm utilizes the *Apriori* candidate generation function to discover class association rules from data sets. CBA introduces the database coverage pruning method to choose high predictive rules. This pruning method is similar to the way that classic greedy algorithms find the rules. The database coverage in relation to the pruning assumes that the rules are already discovered and ranked in order of the confidence and support values. Then starting with the top rule, this method evaluates the rule on the training data set for possible classification. If the current rule was able to classify at least a single training example, it will be put into the classifier and all of its classified data examples will be removed. The same test is repeated for all of the remaining potential rules until all of the data examples are classified or all rules have been tested. Any potential rule that was unable to classify at least one training example is discarded. A number of successive AC algorithms have adopted the CBA database coverage method like CBA (2) [10] and ACCF [11], (uCBA) [12], X-class [13], SPARK-CBA [7], and others.

An improvement over the database coverage pruning was done in the Multiclass Associative Classification (MAC) algorithm [14]. MAC induces the rules based on the concept of vertical AC mining. In pruning the rules, MAC tests each of them on the training data set to decide the most significant ones. Unlike CBA which requires equality between the candidate rule's class and the training example class so that the rule can be inserted into the classifier, MAC considers only the similarity between the rule's body and the training example's attributes values and omits the class equality. This increases the data coverage per rule and reduces over fitting. MAC was applied successfully on generic classification data sets and domain specific data sets (website phishing classification), and showed superiority over CBA with reference to its predictive accuracy and the number of rules in the classifier.

A new improved database coverage method was proposed by [15] in the classification phase of test data. When test data is about to be classified, this method considers each rule's position in the classifier and the number of rules that have similar items to the test data items. Therefore, each class label will be associated with a computed score based on the above conditions and the class that has the largest score will be assigned to the test data. Another closely related work by Hasanpour *et al.* [8] that employed binary harmony search method to choose the ideal class association rule showed competitive results in terms of accuracy on 17 data sets. The authors integrated CBA classifier building method with a harmony search method to cut down the number of chosen class association rules.

Another early developed association-rules based algorithm is Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [16]. It added optimization stage over Incremental Reduced Error Pruning (IREP) [17] that works by simplifying each of the rules as soon as it is built. Reduced Error Pruning (IREP) [18] is the base of both RIPPER and IREP. Unlike CBA and its improvements that use database coverage pruning method, REP and its successors use the concept of error reduction, as stated in the names. It was developed to simplify or prune complex decisions trees by separate-and conquer strategy that divides the data in two sets; growing set and pruning set. The growing set is created using the initial training set whole the pruning set is created by applying one of the pruning operators. When applying the pruning operator results in increased of the error, the process stops. The main disadvantages is the high running time complexity; $O(n^4)$. Unline REP which builds a full rile set before it prunes it, IREP and RIPPER prune each rule as soon as it is built; and the algorithms re faster in computation; $O(n \log n)$.

Reference [19] developed algorithm called Multiple Association Rules (CMAR). It works by constructing the rules in a data structure, which takes on the shape of a tree. This tree saves the rules in a ranked manner according to the rules' support values. Then CMAR performs rule pruning by discarding any rule with a large number of attribute values and keeps the smaller attribute value rules in order to minimize the chance of rule redundancy. The rule pruning method proposed in CMAR favours rules with a smaller body (left hand side) since these rules frequently classify larger portions of the training data examples and therefore the final classifier size gets minimized. This makes the classifier more usable. There are a few AC algorithms that have utilized CMAR pruning method including lazy AC approaches, i.e., Live and Let Live L^3 [20]. In 2012, a new enhancement over CMAR pruning was developed in [21]. The authors developed a pruning method called I-Prune that not only eliminates low frequency items in the preliminary stage but also eliminates items with low to no correlation with the class labels. The basis of item-class correlation is chi-square testing:

$$CHI(a,c) = \frac{N \times (XZ - YW)}{(X+W) \times (Y+Z) \times (X+Y) \times (W+Z)} \quad (1)$$

where X is the number of times feature a and class c occur together, Y is the number of times feature a occurs without class c, W is the number of times class c occurs without feature a, Z is the number of times neither a or c occur, and N is the total size of the training set.

A different learning methodology inherited from the FOIL rule induction technique [22] was introduced in Classification based on Predictive Association Rules (CPAR) [23]. Unlike the greedy approach, CPAR builds the rules simultaneously without having to remove their data examples from the training data set. Rather, when a rule is built, CPAR amends the weight of the data examples by decrementing them by a factor. This ensures the generation of additional rules since a training example can be covered by multiple rules instead of a single one. Nevertheless, this unfortunately may lead to large classifiers. CPAR reduces this problem by performing Laplace estimation pruning on each discovered rule as shown in Equation (2).

$$Laplace\ estimation = \frac{n_c + 1}{n_{tot} + k} \quad (2)$$

where k denotes number of classes, n is the examples and c is the predicted class of a rule.

This procedure involves computing the expected error per derived rule and comparing the computed error with the predefined threshold. When the expected error of the rule exceeds the threshold, the rule gets deleted. This pruning procedure favours rules with a higher expected accuracy.

A closely related pruning method based on the Pearson correlation measure (Equation 3) was proposed in an AC called Hierarchical Multi-label Associative Classification (HMAC) [24].

$$Pearson\ correlation\ measure = \frac{N \sum xy - (\sum x)(\sum y)}{\sqrt{[N \sum x^2 - (\sum x)^2][N \sum y^2 - (\sum y)^2]}} \quad (3)$$

where N is the member of pairs x and y variables to be tested for their relationship strength.

This pruning is performed to measure the correlations between the class of the rule and the rule's items. Any rule that has a computed correlation below the minimum threshold will be discarded. HMAC further reduces the number of rules by invoking CMAR rule pruning to remove any specific rules that have many attribute values in their bodies (left hand side).

Reference [25] proposed a new pruning method called MCAC that makes use of the conflicting rules in an algorithm. Conflicting rules are rules with the same body but that predict different class labels. These conflicting rules were considered errors and therefore removed by the majority of the existing AC algorithms. Nevertheless,unlike the aforementioned AC algorithms,the MCAC algorithm merges these conflicting rules to derive a new multi-label rule during the rule evaluation phase. This rule pruning was based on an earlier rule discovery approach called recursive learning that was proposed by [26]. Experimental results on generic classification data sets from UCI repository - besides phishing data sets - revealed the advantages of the new multi-label rules that users may gain besides the higher predictive accuracy classifiers. Distributed versions of different AC pruning methods were implemented by [9] in an algorithm named MapReduce MCAR. The authors clearly pinpointed to the performance differences of pruning phase when it comes to big datasets.

Finally, [27] developed a modified pre-pruning method based on J-Pruning [28] to minimize overfitting the training data. J-Pruning adopts Entropy from the information theory to measure the worthiness of deleting a variable (attribute) from the rule's body. Experimental results against UCI datasets showed a decrement on the number of items per rule when J-prune was plugged in covering the algorithms.

The aforementioned AC approaches evaluate the rules to select the most effective ones. In doing so, unfortunately, these approaches do not consider the fact that the rules have common data examples. Not paying attention creates the serious problem addressed earlier in Section II. This problem can be reduced if a constant change of the rules' position is applied whenever a rule such as R_i is inserted into the classifier because of $R_i's$ training data removal. Hence, it is imperative to remove the overlapping data amongst all of the rules to end up with a more genuine classifier with minimal rule redundancy.

Padillo *et al.* [7] evaluated a number of associative classifiers on 30 classic data sets and 10 big data sets (datasets with very high dimensionality). These algorithms including CBA and CPAR among others have been extended and implemented In big data platform such as SPARK and FLINK. Results reported that CBA-SPARK and CBA-FLINK generated good results in terms of accuracy. However, sequential

learning methods are unable to process high dimensional data sets because of the extensive computations and the intermediate memory needed during the training phase.

IV. THE PROPOSED APR ALGORITHM

The proposed algorithm consists of three main phases: inducing the rules, constructing the classifier and classifying the test data. The algorithm in phase (1) scans the training data set to discover the rules based on two main thresholds: (*MinSupp* and *MinConf*). In phase (2), the discovered rules are ranked based on different criteria including the rule's confidence, support and number of attributes and it is then evaluated on the training data set to seek out the ones that are able to cover data examples. Any rule that is able to cover at least one training data example is inserted into the classifier and rules with no data coverage are deleted. During the rule evaluation, when a rule is tested on the training data set, the order of the remaining untested rules is constantly revised to reflect the action of removing the training examples per inserted rule. This procedure is discussed in Section 4.2. Phase (3) involves using the classifier to forecast the test data. In this phase, a group of rules classification method that allocates class labels to the test data based on weights is utilized. This method is explained in Section 4.3. The APR algorithm primary steps are displayed in Algorithm 1 and in the next subsequent sections details about each phase are elaborated.

Algorithm 1 APR Algorithm Main Steps

Input: Training data set T, MinSupp and *MinConf* thresholds
Output: A classifier that contains rules

Phase (1) Inducing the rules

1: **for** each attribute value (at_v, class) in T **do**
2: Build a vertical data structure to hold attributes values and their location in T (TidList)
3: Convert frequent 1- attribute values that pass the *MinConf* threshold into 1-rules (Rule_Temp← (at_v, class))
4: **for** each disjoint frequent n-attribute value (start with n=1) **do**
5: a. Perform intersection to produce candidate n+1-attribute values
6: b. Identify frequent n+1-attribute values
7: c. Produce n-rules from n+1- frequent attribute values (Rule_Temp← n+1 (at_v, class))
8: d. Repeat for next inline level candidate attribute values
9: e. Stop when no n+1-attribute value frequency survives the *MinSupp* threshold
10: **end for**
11: **end for**

Phase (2) Constructing the classifier (Algorithm 3, Section 4.2)
Phase (3) Predict the class of test data (Algorithm 4, Section 4.3)

A. RULE INDUCTION STEP

In the search for the rules, our algorithm performs two main steps. The first step involves discovering all of the frequent items. In the second step, the frequent items are converted into ''If-Then'' rules. An item is said to be frequent when its frequency in the training data set is larger than the user's *MinSupp* threshold. All infrequent items are discarded since they do not hold enough support.

In the process of discovering frequent items, unlike the majority of the existing AC algorithms which primarily use horizontal mining based on *Apriori* [29] and its successors, the APR algorithm utilizes a vertical mining approach introduced by [30]. For additional information on the differences between horizontal and vertical mining [2] with the references herewith are good review. The significant element in vertical mining is a data structure named TidList (Transaction IDs List) that holds all items plus class labels along with their appearances in the training data. Hence after the first training data scan, our algorithm creates a TidList that initially contains candidate 1-items and the class value. APR then performs simple intersections among the item TidLists to discover the frequent ones and subsequently, the rules. For example, if we have two frequent 1-items such as (<''Gender=Male''>, class=yes) and (<''Credit rate= Good''>, class=yes) with TidLists (Row IDs) (10, 20, 22,205, 255, 285) and (5, 6, 20, 22, 200, 205, 280, 285, 291) respectively. The new possible candidate 2-items, i.e. (<''Gender=Male, Credit rate= Good''>, class=yes), has a TidList = (20,22,205,285) because of the 1-items' intersection. This new item has a support count of 4 (the size of its new TidList).

The production of candidate items at any level (1,2,3, etc) is performed by intersecting the TidLists of any two disjoint frequent N-attribute values with similar class labels to create candidate N+1-items. So, frequent 1-items are used to generate candidate 2-items, which in turn, are used to derive frequent 2-items, and so forth. The fact that the item's TidListis used to locate its frequency is an advantage when calculating the item's support and confidence, which are the main criteria used to produce the rules. This is an efficient data processing task. The APR algorithm performs the below steps according to phase (1) in Algorithm 1 to find the rules:

1) Scan the training data set to build TidLists for frequent 1-items and their class labels.
2) Intersect disjoint frequent 1-items' TidLists to produce candidate 2-items. Once the entire frequent 2-items are derived, we repeat the same process of intersection to derive frequent 3-items and so forth.
3) When the complete frequent items are found, we convert any with a confidence greater than the *MinConf* threshold as a candidate rule.

B. RULE EVALUATION STEP

The process of building the classifier or finding the most significant rules is comprised of two sub-steps. Firstly, all of the extracted rules from phase (1) are sorted according to the

sorting method of Algorithm 2. This sorting method ensures that the rules with high confidence and support values are placed at the top so they have a higher chance of becoming part of the classifier. This is a vital step to ascertain high predictive rules, usually those that mathematically have a large confidence and support values. In addition, the rules in the classifier need to cover substantially large numbers of training examples (rows) to minimize the size of the classifier. The process of ranking precedes the step of the rule evaluation against the training data. The ultimate aim of the evaluation is to select good representative rules set as the classifier.

Algorithm 2 Rule Ranking Procedure of APR Algorithm

Input: The set of rule; *R'*
Output: A set of ranked rules *R*.
For two rules; r_a and r_b, $r_a > r_b$ iff :

1: Confidence (r_a) > Confidence (r_b)
2: Confidence (r_a) = Confidence (r_b) but Support (r_a) > Support (r_b)
3: Confidence (r_a) = Confidence (r_b) and Support (r_a) = Support (r_b) but $|r_a| > |r_b|$
4: When all 1-3 are similar for r_a and r_b then the choice is arbitrary

In the rule evaluation (Algorithm 3), and for each training data example, we iterate over the set of discovered candidate rules starting with the highest sorted one. When the rule's body matches the training example, it will cover the training example and will then be inserted into the classifier. Then, the training data examples connected with the evaluated rules are deleted. The process is repeated until no more data remains or the complete set of candidate rules has been evaluated. When this happens, the largest frequent class label in the remaining uncovered data examples becomes the default class rule.

The novelty of the proposed algorithm lies in the process of rule pruning. In particular, once a R_i is highlighted as a classifier rule and its training data is deleted, we can argue that this deletion impacts other not yet evaluated rules (often with a lower rank). The removal of $R_i's$ training data must be reflected on those rules and therefore the APR algorithm updates the confidence and support frequencies of these rules. The proposed procedure changes the rank for some rules and may determine which rules have the chance to be inserted into the classifier. Hence identifying on a continuous basis which rules have higher rank is crucial since they have a better chance of being selected first during the rule evaluation and thus can be used later in the test data classification step. The proposed procedure leaves no chance for lower rank candidate rules to become part of the classifier and discards them once their frequency drops below *MinSupp*.

Lastly, running the APR rule pruning procedure ensures keeping the right rules that are mathematically fit (higher rank) and discarding others that are weaker (lower rank) in real time. This is a new process in which the useless rules are identified without going back to count them in the training data set, hence enhancing the classifier's overall performance. Overall, when a rule covers a training example, APR performs the following in phase (2):

1) The rule will be inserted into the classifier
2) All covered training data examples are removed.
3) The rules' rank gets changed

Table 1 below shows the primary differences between the proposed pruning method and existing ones in AC research.

Algorithm 3 APR Rule Evaluation Procedure

Input: The set of extracted Rules_set and the training data set *(T)*
Output: classifier *(C)*

1: *Temp'* = rank (Rules_set)
2: $\theta \leftarrow$ *Classifier*
3: $\theta \leftarrow$ *Data Structure (D_S)*
4: **for** each training example t in T **do**
5: find the first ranked rule $r_i \in$ *Temp'* that its attributes is inside t
6: **if** no rules match t **then**
7: keep t uncovered;
8: **else**
9: begin
10: $D\_S \leftarrow D\_S \cup r_i$
11: remove t
12: update the rank of rules in *Temp'*
13: end
14: **end if**
15: **end for**
16: discard all rules in *Temp'*
17: C $\leftarrow C \cup D\_S$
18: **if** $|T| > 0$ – size of the training data set **then**
19: create a default rule from unclassified examples T
20: **else**
21: create a default rule from the current C (most class appearing with rules) and add it to C
22: **end if**

C. TEST DATA CLASSIFICATION STEP

The last and most vital step in the life cycle of any classification algorithm is test data classification. In this step, the AC algorithm normally uses one or more rules to assign the class label to a test data. Generally, there are two main approaches in AC classification to accomplish test data classification. The first relies on assigning a single rule class, usually the first rule similar to the test data items. This approach has been highly criticized by several scholars primarily because there could be multiple rules that possibly match the test data and therefore utilizing all of these rules seems to be more realistic and less questionable. The second approach utilizes

TABLE 1. Primary differences between the proposed pruning method and existing pruning methods in AC mining.

APR Pruning	Other AC Pruning
The proposed pruning method considers pruning during the learning step in a dynamic manner	The database coverage pruning considers pruning the rule once all rules are derived
The proposed pruning method considers the last support and confidence per rule as the bases for pruning	The majority of AC pruning like database coverage, specific rules and party matching pruning methods utilize the original support and confidence as the bases for rule ranking.
The proposed pruning method tries to cover as many data instances as possible per rule and thus smaller number of rules are derived	The database covering method and its successors require do not consider the number of instances covered by the rule which may result in rules covering just limited number of training instances

a group of rules to make the classification decision based on voting or weights.

The APR algorithm utilizes a multiple rule test data classification method (see Algorithm 4) whereupon when a test data (t_{si}) is about to be classified, our algorithm goes over the classifier rules and identifies those whose body (items) is contained within the test data. Then the identified rules are divided into clusters according to the class labels and the class which belongs to the largest cluster (the cluster containing more number of rules) is then allocated to the test data. In cases where there is more than one cluster with similar rules' count, the choice will depend on the rule rank in the classifier. By applying this procedure, we fulfil the concept of group-based rule decisions and eliminate the biased decision of favouring a single rule over others. Consequently, the class allocation decision of test data becomes less questionable and unbiased. Lastly, when no rules in the classifier are similar to the test data, the default class rule will be used.

Algorithm 4 Classification Procedure of APR

Input: classifier (C) / set of ranked rules and a test data instance (t_s)
Output: classifier (C)

```
for each class cl_j in C do
    for for each rule (r_i) do
        if t_s ∈ r_i then
            cl_j_count ++;
        end if
    end for
    t_s (class) = cl_j (max (cl_j_count))
    if cl_j_count = 0 then
        t_s (class) = default rule
    end if
end for
```

D. MERITS OF APR ALGORITHM AND DISTINGUISHING FEATURES

1) APR proposed a new pruning method that ensures each rule is tested during the learning phase and not after all rules have to be generated. This removes an entire phase in AC which is rule evaluation.
2) The search space of items in APR algorithm is reduced since weak items are eliminated on the fly and therefore minimizing the time required to build the classifier.
3) APR algorithm ensures a true frequency (support) and strength (confidence) are linked with each rule rather a static support and confidence. Therefore, each rule represents the true state of the data it was derived from.
4) The classification accuracy is improved by the APR algorithm because more than one rule is taken into account when allocating the class of test data. In doing so, two metrics are utilized; the rule position in the cluster and the number of rules in each cluster. This method reduces random prediction particularly when more than one rule is applicable to classify a test data.

V. APR EXAMPLE

A detailed example is given in this section to demonstrate the pros and cons of the APR algorithm. It will show the algorithm's main phases using a sample of historical data (Table 2). Let's assume that the user's *MinSupp* and *MinConf* are set to 20% (2 out 10) and 66.67% respectively for presentation purposes. The support count is used in the tables instead of percentage to indicate the *MinSupp*.

A. RULE GENERATION

The frequent 1-items are given in Table 3. All infrequent 1-items are deleted after the initial training data set scan. When frequent 1-items are derived, they are used to produce frequent 2-items based on testing their support counts against the *MinSupp* threshold.

One feature imposed within APR is that it only intersects candidate items that have a common class label which reduces the number of TidLists intersections. This may summarize the number of candidate items at any given iteration. Once all candidate items are identified, the APR computes their confidence in order to generate the rules as shown in Table 4. All rows that are not bold in Table 4 correspond to rules that have been removed since they fail to survive the *MinConf* threshold. Before the candidate rules are evaluated against the training data set, they must be ranked according to the ranking

TABLE 2. A sample historical data set.

Line #	$Attribute_1$	$Attribute_2$	$Attribute_3$	Class
1	x_1	y_1	z_1	$class_2$
2	x_1	y_1	z_2	$class_2$
3	x_1	y_1	z_2	$class_1$
4	x_1	y_2	z_2	$class_1$
5	x_3	y_1	z_1	$class_1$
6	x_1	y_1	z_1	$class_2$
7	x_1	y_2	z_3	$class_1$
8	x_1	y_2	z_3	$class_1$
9	x_1	y_2	z_2	$class_1$
10	x_1	y_2	z_2	$class_2$

TABLE 3. Frequent 1-rule items obtained from Table 2.

Frequent 1-**Ruleitem**		**Support Count**	**Confidence**
Attribute	**Class**		
Attribute$_1$			
x_1	$class_2$	4	57.14% (4/7)
x_2	**$class_1$**	**2**	**100.00% (2/2)**
x_1	$class_1$	3	42.85% (3/7)
Attribute$_2$			
y_1	$class_2$	3	60.00% (3/5)
y_1	$class_1$	2	40.00% (2/5)
y_2	**$class_1$**	**4**	**80.00% (4/5)**
Attribute$_3$			
z_1	**$class_2$**	**2**	**66.67% (2/3)**
z_2	$class_2$	2	40.00% (2/5)
z_2	$class_1$	3	60.00% (3/5)
z_3	**$class_1$**	**2**	**100.00% (2/2)**

TABLE 4. Frequent 2 & 3-rule items.

Frequent 2-**Ruleitem**		**Support Count**	**Confidence**
Attribute	**Class**		
(x_1, y_1)	**$class_2$**	**3**	**100.00% (3/3)**
(x_1, y_2)	$class_1$	**3**	**75.00% (3/3)**
(x_1, z_1)	**$class_2$**	**2**	**100.00% (2/2)**
(x_1, z_2)	$class_2$	2	50.00% (2/4)
(x_1, z_2)	$class_1$	2	50.00% (2/4)
(y_1, z_1)	**$class_2$**	**2**	**66.67% (2/3)**
(y_2, z_2)	**$class_1$**	**2**	**66.67% (2/3)**
(y_2, z_3)	**$class_1$**	**2**	**100.00% (2/2)**
Frequent 3-**Ruleitem**		**Support Count**	**Confidence**
Attribute	Class		
(x_1, y_2, z_2)	**$class_1$**	**2**	**66.67% (2/3)**
(x_1, y_1, z_1)	**$class_2$**	**2**	**100.00% (2/2)**

procedure of Algorithm 2. Thus, a column titled "Original Candidate Rule Rank" (First column)is added to Table 5. In Table 5 the survived candidate rules are sorted with respect to confidence and support values.

B. RULES EVALUATION

The rule evaluation method of APR (Algorithm 3) is applied on the candidate rules derived (see Table 5) where each training example can be classified by one rule. Starting with the highest ranked rule ($x_1 \wedge y_1 \rightarrow class_2$), which covers line #'s (1,2,6) in the training data set, it will be inserted into the classifier and its training examples will be discarded. All unevaluated rules that have all of its items inside (Lines 1,2,6), i.e., rules' rank #4, #6,#9 and #10 are amended and pruned since they no longer cover any more data. The algorithm then moves to evaluate the next in line rule according to rank (Rule: $x_2 \rightarrow class_1$) after updating the rule rank when Rule #1 was inserted into the classifier. This rule was originally ranked #2 but after evaluating Rule #1, its rank has improved to #1. This rule covers line #'s (3,7) so it gets inserted into the classifier and all its classified data is removed. The removal of this rule's data instances from the training data set has decremented the support of three potential rules $z_3 \rightarrow class_1$, $y_2 \rightarrow class_1$ and $y_2 \wedge z_3 \rightarrow class_1$. Two of these three rules are discarded since their support values are now below the minimum support threshold and one impacted rule has stayed, i.e., $y_2 \rightarrow class_1$.

The next in line rule to be evaluated is $x_1 \wedge y_2 \rightarrow class_1$. This rule covers four training instances (4, 8,9,10) and thus these instances are removed leaving a single uncovered data example (Line #5). From this data example, APR creates a default class rule since none of the candidate rules are able to classify this particular data example. This means that all training data examples are now classified (covered) by only three rules, as well as the default class rule (Class 1 - majority class). The remaining candidate rule, i.e., (#12) is redundant and so it is discarded. The classifier generated from the training data set of Table 2 consists of three rules plus a default rule as shown in Table 6. The default class represents the most frequent class in the unclassified training instances.

VI. DATA AND ANALYSIS OF RESULTS

In this section, different data sets from the University of California Irvine (UCI) repository have been used to evaluate the proposed algorithm's performance. The data sets are displayed in Table 7 where all numerical attributes have been discretised. We have chosen different types and sizes of data sets for a fair comparison and to measure scalability. A number of known classification algorithms that generate

TABLE 5. Candidate rules set produced before rules evaluation.

Original Candidate Rule Rank	Attribute values		Original Support count	Original Confidence	Rule Rank After Evaluating $x_1 \wedge y_1 \rightarrow class_2$	Rule Rank After Evaluating $x_2 \rightarrow class_1$	Rule Rank After Evaluating $x_1 \wedge y_2 \rightarrow class_1$
	ATT	Class					No rules with acceptable frequency remains
2	$\mathbf{x_2}$	$\mathbf{class_1}$	**2**	**100.00%**	**(1)**	**evaluated**	
7	$\mathbf{y_2}$	$\mathbf{class_1}$	**4**	**80.00%**	**(4)**	**75.00% (keep) (2)**	**0 < min support (discard)**
9	$\mathbf{z_1}$	$\mathbf{class_2}$	**2**	**66.67%**	**0< min support (discard)**	**–**	**–**
3	$\mathbf{z_3}$	$\mathbf{class_1}$	**2**	**100.00 %**	**(2)**	**1 < min support (discard)**	**–**
1	$\mathbf{(x_1, y_1)}$	$\mathbf{class_2}$	**3**	**100.00%**	**evaluated**		
8	$\mathbf{(x_1, y_2)}$	$\mathbf{class_1}$	**3**	**75.00 %**	**(5)**	**(1)**	**evaluated**
4	$\mathbf{(x_1, z_1)}$	$class_2$	**2**	**100.00 %**	**0< min support (discard)**	**–**	**–**
5	$\mathbf{(y_2, z_3)}$	$\mathbf{class_1}$	**2**	**100.00 %**	**(3)**	**1 < min support (discard)**	**–**
10	$\mathbf{(y_1, z_1)}$	$\mathbf{class_2}$	**2**	**66.67%**	**0< min support (discard)**	**–**	**–**
11	$\mathbf{(y_2, z_2)}$	$\mathbf{class_1}$	**2**	**66.67%**	**(6)**	**(3)**	**0< min support (discard)**
6	$\mathbf{(x_1,y_1,z_1)}$	$\mathbf{class_2}$	**2**	**100.00 %**	**0< min support (discard)**	**–**	**–**
12	$\mathbf{(x_1,y_2,z_1)}$	$\mathbf{class_2}$	**2**	**66.67%**	**(7)**	**(4)**	**Discarded (no data coverage)**

rule-based classifiers have been utilized to conduct the experiments. In particular, decision trees [22], CBA [1] and RIPPER [16] are the selected algorithms. The selection of these algorithms was based on three primary reasons:

a) All produce rules in the format of "If-Then" similar to the proposed algorithm
b) They adopt different rule induction mechanisms
c) All are known algorithms that have been evaluated by previous researchers in the data mining community and applied to different types of data

The experiments on the UCI data sets using the considered classification algorithms were conducted on the Waikato Environment for Knowledge Analysis (Weka) tool [31]. This tool is an open source based on a Java platform that contains implementation for different data mining methods including

TABLE 6. Classifier produced from Table 2.

Rules
$x_1 \wedge y_1 \rightarrow class_2$
$x_2 \rightarrow class_1$
$x_1 \wedge y_2 \rightarrow class_2$
Default $Class_1$

TABLE 7. The UCI data sets characteristics.

Data-set	Size	# of Classes
Glass	214	7
Iris	150	3
Tic-tac	958	2
Breast	699	2
Zoo	101	7
Heart	270	2
Diabetes	768	2
Pima	768	2
Vote	435	2
Balance-scale	625	3
Mushroom	8124	3
Led	3200	3

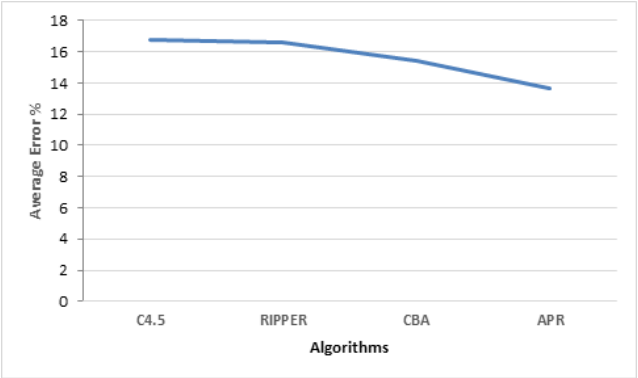

FIGURE 1. Average 1-error rate % for the considered algorithms on the UCI data sets.

filtering, classification, clustering, evaluation, visualisation, etc. This tool was designed and implemented at The University of Waikato in New Zealand to help students, researchers, and academic staff in conducting quantitative research. The proposed algorithm has been coded in Java programming language within the Weka environment. All experiments have been run on a computer machine with a 2.0 GHz processor.

The *minSupp* and *minConf* for APR and CBA have been set in all experiments to 2% and 30% respectively. The *minSupp* plays a critical role in controlling the number of candidate rules that may be generated and therefore we have followed other scholars in AC literature [1], [25] and set it to 2% accordingly. On the other hand, the *minConf* has low impact on the performance and was set to 30%. The stratified ten-folds cross validation method has been used for testing all of the considered classification algorithms. This method is widely used in data mining for fair testing. In this testing method and prior rule discovery, the data set gets partitioned into ten folds and the algorithm is trained on nine folds and then tested on the remaining fold. This process is repeated for ten runs whereupon in each run, an error rate is produced. Then the error rates derived from the ten runs are averaged to produce an overall error rate on the classifier.

A number of evaluation measures are utilized to show the pros and cons of the proposed algorithm when compared with other classification algorithms. The measures below have been used to evaluate APR:

- One error rate (%) (Equation 5)
- Number of rules derived

One of the common metrics of evaluation in classification is Accuracy (Equation 4). It represents the number of correct classifications made from of the total classifications made by the classifier. Error rate is the total number of misclassifications made by the classifier from the total number of classifications.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{4}$$

$$ErrorRate = 1 - Accuracy \tag{5}$$

Figure 1 depicts the average one-error rategenerated by C4.5, CBA, RIPPER and APR on the UCI data sets. The figures show that the APR algorithm outperformed the considered algorithms on average and particularly a higher average error than C4.5, CBA and RIPPER by 3.12%, 2.92%, and 1.71% respectively. The new rule pruning that ensures constant updates on the confidence and support values during building the classifier that guarantees fair and high quality rules. These rules play a significant role in decreasing the error rate of the classifiers produced by the APR algorithm when compared with the other static learning methods such as CBA. Furthermore, the APR algorithm utilizes natural pruning by deleting any weak rules that have been induced and are associated with low support or confidence. This assures that the only rules kept for predicting test data cases have acceptable statistical representation.

In Figure 2, the error rate per data set has been generated for all of the considered algorithms to further evaluate the predictive power of the proposed algorithm. The figures clearly show consistent domination for APR when compared to the remaining algorithms. In particular, APR's won-lost-tie record against C4.5, CBA and RIPPER is 7-3-0, 7-3-0 and 9-1-0 respectively. The way that APR constructs the classifier

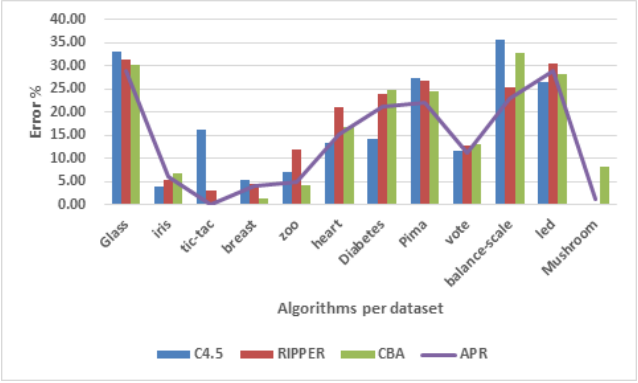

FIGURE 2. Error rate (%) for the considered algorithms on the UCI data sets.

by removing the training data overlapping among the rules safeguards that:

a) Each training example is covered by only a single rule and is used only once during the training phase. Therefore, not allowing a training example to be used multiple times in rule learning as in the association rule.

b) Rules' frequencies which are the primary measure for the rule strength (confidence and support) are constantly amended to achieve high quality classifiers.

Dealing with the exponential growth of the rule problem contributed to the decrease of the one-error rate in the classifiers derived from APR. This is since allowing only those rules that cover training data to be part of the classifier ensures that good predictive rules are used to assign class labels for the test data during the classification step. The classifier sizes (average numbers of rules) generated by C4.5, RIPPER, CBA and APR algorithms are shown in Figure 3. The purpose is to compare the proposed algorithm with rule induction, AC and decision tree approaches to seek the effectiveness of the new pruning method and its impact on the final classifiers.

Figure 3 illustrates that the proposed algorithm was able to reduce the classifier size of the AC algorithms yet did not negatively influence the predictive accuracy. This can be attributed to the new pruning performed by APR during the rule evaluation. APR always sustains strong rules by pruning any weak rules that do not maintain an acceptable support and confidence value during the rule evaluation step. In other words, removing training data instances from other rules whenever a rule becomes part of the classifier surely minimizes rule redundancy, and therefore has resulted in a more concise set of rules that maintain high predictive accuracy during the test data classification phase. It is obvious from the number of rule results that the CBA algorithm produced the largest classifiers despite performing multiple pruning runs followed by C4.5. The algorithm that derived the least numbers of rules is RIPPER since it follows a separate-and-conquer strategy while inducing the rules beside multiple levels of rule pruning. RIPPER normally employs validation set pruning and testing set pruning to only keep perfect rules (rules with low data coverage yet high expected accuracy). However, despite the fewer number of rules generated by RIPPER, this algorithm's classifiers often suffer from low predictive accuracy when compared to AC and decision tree approaches.

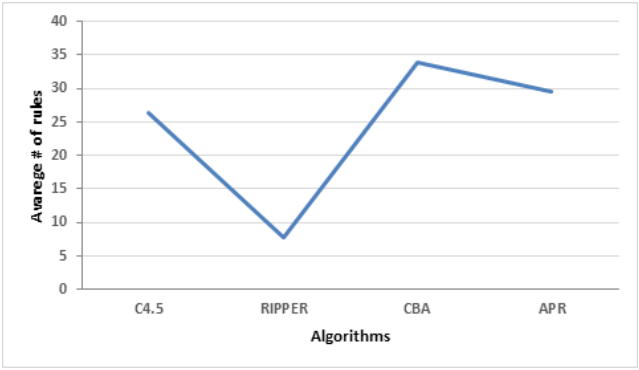

FIGURE 3. Average number of rules generated by the considered algorithms on the UCI data sets.

Lastly, the time taken to construct the classifier from the UCI datasets of the AC algorithms (CBA, APR) is shown in Figure 4. The proposed algorithm consistently required less training time to generate the classifiers when compared to CBA. We limited the experiments of the classifiers' training time to the AC algorithms (CBA, APR) for a fair test. The won-lost-tie record of APR with reference to processing time when compared to CBA is 9-0-1. This is clear evidence of the superiority of the APR pruning procedure in which it discards rules early on without having to check the data coverage against the training data set. This is unlike CBA, which assumes that all candidate rules have a fixed confidence and hence they will be checked during the building of the classifier until the training dataset becomes empty. The fact that our proposed algorithm removes rules when they become weak (their frequency dropped) during rule evaluation step is advantageous. APR scales well in terms of the number of instances in the dataset. This is obvious on the "Mushroom" and "Led" datasets that have large numbers of data examples if contrasted with the other datasets.

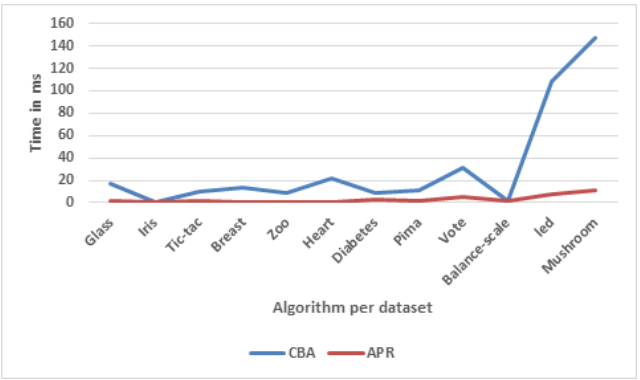

FIGURE 4. Processing time required to construct a classifier on the UCI data sets.

VII. AUTISM DATASET RESULTS ANALYSIS

A real Autism Spectrum Disorder (ASD) dataset related to adults have been used to validate APR performance. This dataset was retrieved from Thabtah [32], [33] using a mobile screening application called ASDTests. The dataset contains 24 varaibles and 1118 data instances of adults with and without ASD. The first ten variables in the dataset are questions based on a classic medical questionnaire for ASD called Autism Quotient - short version (AQ-10) (See Table 8) [34]. The values of the questions are converted into 0 and 1 based on respondents' replies Thabtah *et al.* [35]. The main variable that denotes exhibiting ASD traits is called the score. This variable represents the total score computed after summing up

TABLE 8. Description of the first ten variables of the ASD adult dataset.

Variable	Description based on AQ-10	Variable Type
A1	Does your child look at you when you call his/her name?	Boolean
A2	How easy is it for you to get eye contact with your child	Boolean
A3	Does your child point to indicate that he/she wants something?	Boolean
A4	Does your child point to share interest with you?	Boolean
A5	Does your child pretend?	Boolean
A6	Does your child follow where you're looking?	Boolean
A7	If you or someone else in the family is visibly upset, does your child show signs of wanting to comfort them?	Boolean
A8	How would you describe your child's first words?	Boolean
A9	Does your child use simple gestures?	Boolean
A10	Does your child stare at nothing with no apparent purpose?	Boolean

all 1s for variables A1–A10. When the score is larger than 6, then the response variable (target class) is given 'Yes' depicting the adult has ASD traits, and when the score is smaller than 6 the class is assigned 'No' showing that the adult has no ASD. There are additional attributes such as Family History, Age, Gender, Ethnicity and Resident Country among others. For complete details on the variables and their description for the adult ASD dataset refer to Thabtah *et al.* [35].

The adult ASD dataset is imbalanced with the low frequency class 'NO' (adult without ASD) of 358 instances, and majority class being 'YES' (adult without ASD) with 760 instances. To ensure we end up with fair classifiers during the process of training, we discarded the score variable from the dataset, as the class variable is derived from the total score.

The predictive accuracy generated by the considered classification algorithms from the adult ASD dataset is depicted in Figure 5. In that figure, it is clearly that APR consistently derived higher competitive classifiers with regards to accuracy than C4.5, RIPPER and CBA algorithms. It seems that automatically amending the rules and keeping the most influential ones (those that cover training instances without any overlapping) have direct impact on the predictive accuracy at least on the ASD dataset we consider. APR not only keeps rules with high confidence as conventional associative classifiers but also ensures that each rule covers at least one instance thus cutting down the search space and ensuring that a rule may classify more instances within the training dataset. Furthermore, only 25 instances that are actually not on the spectrum has been misclassified by APR algorithm based on the classifier derived from the ASD dataset. Whereas, 38 and 30, 30 instances that are actually without ASD have been misclassified by RIPPER and C4.5 and CBA algorithms respectively increasing the false positives. The number of rules derived by APR was 32 compared with 99 rules by CBA associative classifier and this indeed shows that not only APR reduced the classifier size but also slightly increased its predictive power.

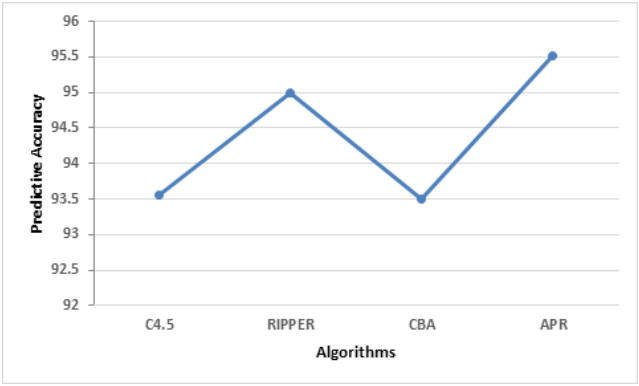

FIGURE 5. Predictive accuracy of APR when compared with the considered algorithms on the ASD dataset.

VIII. CONCLUSION

AC methods suffer from a large number of rules, which is a problem inherited from association rule mining. This problem limits the use of AC in application domains due to the very large classifiers that might be derived, which is hard to control by the end users. This article proposed a novel rule pruning procedure in AC; APR that amends the rules' position in the classifier every time that a rule is inserted rather than relying on the initial rules' frequency computed from the training data set. This results in only keeping rules with actual data coverage in the classifier and discards any rule whose data frequency has dropped below the acceptable minimum frequency. The influence of this new procedure was apparent on the classifiers size and on the predictive accuracy of the APR classifiers. The results have been obtained from experimenting on a number of data sets from the University of Irvine's (UCI) data repository & real adult ASD dataset and using known AC, covering and decision tree algorithms. The experimental results showed that APR is very competitive with reference to the predictive accuracy in relation to the C4.5, RIPPER, and CBA algorithms. In fact, the new classifiers procedure of APR showed an improvement in the classification accuracy when compared to those of C4.5, RIPPER, and CBA. Furthermore, APR regularly generated fewer rules yet had a more efficient training procedure than CBA due to the new pruning that discards any weak rules while the classifier is progressing. In the near future, we intend to apply APR on unstructured application data such as text mining to seek any possible challenges or improvements.

In near future, the proposed AC will be extended into two directions. The first issue on how to deal with unstructured

features related to medical diagnosis particularly in ASD and other behavioural applications. APR will need particular extension in dealing with variables with sparse possible values or variables that are need transformation before utilizing them during the training step. The second possible area of extending the current work is to integrate APR with ensembles learning approach in [36]–[39] in which multiple classifiers can be used to come up with a collective decision for assigning the class table.

ACKNOWLEDGMENT

The author declares that there is no conflict of interest regarding the publication of this paper).

REFERENCES

[1] B. L. W. H. Y. Ma and B. Liu, "Integrating classification and association rule mining," in *Proc. 4th KDD*, 1998, pp. 80–86.
[2] K. D. Rajab, "New hybrid features selection method: A case study on websites phishing," *Secur. Commun. Netw.*, vol. 2017, Mar. 2017, pp. 9838169:1–9838169:10.
[3] B. Ma, H. Zhang, G. Chen, Y. Zhao, and B. Baesens, "Investigating associative classification for software fault prediction: An experimental perspective," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 24, no. 1, pp. 61–90, 2014.
[4] J. P. Lucas, A. Laurent, M. N. Moreno, and M. Teisseire, "A fuzzy associative classification approach for recommender systems," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 20, no. 4, pp. 579–617, 2012.
[5] F. Thabtah, O. Gharaibeh, and R. Al-Zubaidy, "Arabic text mining using rule based classification," *J. Inf. Knowl. Manage.*, vol. 11, no. 1, 2012, Art. no. 1250006.
[6] F. Thabtah, "Ranked multi-label rules associative classifier," in *Research and Development in Intelligent Systems XXIII*. London, U.K.: Springer, 2007, pp. 87–100.
[7] F. Padillo, J. M. Luna, and S. Ventura, "Evaluating associative classification algorithms for big data," *Big Data Anal.*, vol. 4, no. 1, p. 2, 2019.
[8] H. Hasanpour, R. G. Meibodi, and K. Navi, "Improving rule based classification using harmony search," *PeerJ*, vol. 7, 2019, Art. no. e27634v1, doi: 10.7287/peerj.preprints.27634v1.
[9] F. Thabtah, S. Hammoud, and H. Abdel-Jaber, "Parallel associative classification data mining frameworks based mapreduce," *Parallel Process. Lett.*, vol. 25, no. 2, 2015, Art. no. 1550002.
[10] B. Liu, Y. Ma, and C.-K. Wong, "Classification using association rules: Weaknesses and enhancements," in *Data Mining for Scientific and Engineering Applications*, vol. 591. Boston, MA, USA: Springer, 2001.
[11] X. Li, D. Qin, and C. Yu, "ACCF: Associative classification based on closed frequent itemsets," in *Proc. 5th Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*, vol. 2, Oct. 2008, pp. 380–384.
[12] X. Qin, Y. Zhang, X. Li, and Y. Wang, "Associative classifier for uncertain data," in *Proc. Int. Conf. Web-Age Inf. Manage.* Berlin, Germany: Springer, 2010, pp. 692–703.
[13] G. Costa, R. Ortale, and E. Ritacco, "X-class: Associative classification of XML documents by structure," *ACM Trans. Inf. Syst.*, vol. 31, no. 1, p. 3, 2013.
[14] N. Abdelhamid, A. Ayesh, F. Thabtah, S. Ahmadi, and W. Hadi, "MAC: A multiclass associative classification algorithm," *J. Inf. Knowl. Manage.*, vol. 11, no. 2, 2012, Art. no. 1250011.
[15] S. Ayyat, J. Lu, and F. Thabtah, "Class strength prediction method for associative classification," in *Proc. ARIA Int. Acad., Res. Ind. Assoc.*, 2014, pp. 5–10.
[16] W. W. Cohen, "Fast effective rule induction," in *Proc. 12th Int. Conf. Mach. Learn.*, 1995, pp. 115–123.
[17] J. Fürnkranz and G. Widmer, "Incremental reduced error pruning," in *Proc. 11th Int. Conf. Mach. Learn. (ML)*, 1994, pp. 70–77.
[18] W. W. Cohen, "Efficient pruning methods for separate-and-conquer rule learning systems," in *Proc. 13th Int. Joint Conf. Artif. Intel. (IJCAI)*, 1993, pp. 988–994.
[19] W. Li, J. Han, and J. Pei, "CMAR: Accurate and efficient classification based on multiple class-association rules," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2001, pp. 369–376.
[20] E. Baralis, S. Chiusano, and P. Garza, "A lazy approach to associative classification," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 2, pp. 156–171, Feb. 2008.
[21] E. Baralis and P. Garza, "I-prune: Item selection for associative classification," *Int. J. Intell. Syst.*, vol. 27, no. 3, pp. 279–299, 2012.
[22] J. R. Quinlan and R. M. Cameron-Jones, "FOIL: A midterm report," in *Proc. Eur. Conf. Mach. Learn.* Berlin, Germany: Springer, 1993, pp. 1–20.
[23] X. Yin and J. Han, "CPAR: Classification based on predictive association rules," in *Proc. SIAM Int. Conf. Data Mining*, 2003, pp. 331–335.
[24] S. Sangsuriyun, S. Marukatat, and K. Waiyamai, "Hierarchical multi-label associative classification (HMAC) using negative rules," in *Proc. 9th IEEE Int. Conf. Cogn. Inform. (ICCI)*, Jul. 2010, pp. 919–924.
[25] N. Abdelhamid, "Multi-label rules for phishing classification," *Appl. Comput. Inform.*, vol. 11, no. 1, pp. 29–46, 2015.
[26] F. Thabtah, P. Cowling, and Y. Peng, "Multiple label classification rules approach," *J. Knowl. Inf. Syst.*, vol. 9, no. 1, pp. 109–129, 2006.
[27] F. Stahl and M. Bramer, "Computationally efficient induction of classification rules with the PMCRI and J-PMCRI frameworks," *Knowl.-Based Syst.*, vol. 35, pp. 49–63, Nov. 2012.
[28] M. Bramer, "Using J-pruning to reduce overfitting in classification trees," *Knowl.-Based Syst.*, vol. 15, nos. 5–6, pp. 301–308, 2002.
[29] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, vol. 1215, 1994, pp. 487–499.
[30] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules," in *Proc. KDD*, 1997, pp. 283–286.
[31] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 2005.
[32] F. Thabtah, "Machine learning in autistic spectrum disorder behavioral research: A review and ways forward," *Inform. Health Social Care*, vol. 44, no. 3, pp. 278–297, 2018.
[33] F. Thabtah, "An accessible and efficient autism screening method for behavioural data and predictive analyses," *Health Inform. J.*, vol. 25, no. 4, pp. 1739–1755, 2018.
[34] S. Baron-Cohen, S. Wheelwright, R. Skinner, J. Martin, and E. Clubley, "The autism-spectrum quotient (AQ): Evidence from asperger syndrome/high-functioning autism, malesand females, scientists and mathematicians," *J. Autism Develop. Disorders*, vol. 31, no. 1, pp. 5–17, 2001.
[35] F. Thabtah, F. Kamalov, and K. Rajab, "A new computational intelligence approach to detect autistic features for autism screening," *Int. J. Med. Inform.*, vol. 117, pp. 112–124, Sep. 2018.
[36] J. Alzubi, A. Nayyar, and A. Kumar, "Machine learning from theory to algorithms: An overview," *J. Phys., Conf. Ser.*, vol. 1142, no. 1, 2018, Art. no. 012012.
[37] J. A. Alzubi, "Diversity based improved bagging algorithm," in *Proc. Int. Conf. Eng. (MIS)*, 2015, p. 35.
[38] J. A. Alzubi, "Research article optimal classifier ensemble design based on cooperative game theory," *Res. J. Appl. Sci., Eng. Technol.*, vol. 11, no. 12, pp. 1336–1343, 2015.
[39] O. A. Alzubi, J. A. Alzubi, S. Tedmori, H. Rashaideh, and O. Almomani, "Consensus-based combining method for classifier ensembles," *Int. Arab J. Inf. Technol.*, vol. 15, no. 1, pp. 76–86, Jan. 2018.