The Thirty-Third International
FLAIRS Conference (FLAIRS-33)

Improving Classification Accuracy by Mining Deterministic and Frequent Rules

Yuxiao Huang
George Washington University
yuxiaohuang@gwu.edu

Abstract

Patterns underlying the data sometimes take the form of *IF* conditions *THEN* outcome. However, not all the classifiers can detect such rules, resulting in compromised classification accuracy. In this paper we proposed an Add-on Rule-based Classifier (ARC) that can be paired with any existing classifier (base). The idea of ARC is improving the accuracy of the base by 1) mining deterministic and frequent rules, and 2) using such rules to assist the base in classification. Key novelty includes 1) a greedy search algorithm that identifies the rules by alternating between adding the "best" condition and removing the "worst", and 2) new heuristics for selecting the best and worst conditions. We theoretically proved that rules detected by ARC are sound, complete, and minimal, indicating that ARC will almost never degrade the accuracy of the base, but instead, could often improve it. To experimentally verify this claim, we paired ARC with 9 leading classifiers and tested the ensembles on 12 UCI datasets. Empirical results show that, ARC never lowers the accuracy of the base and, more importantly, usually increases it (where some of the increases are statistically significant), echoing what we theoretically proved. The code, data, and full results are publicly available in our github repository: https://github.com/yuxiaohuang/research/tree/master/gwu/accepted/flairs_2020/arc. The results are reproducible by simply using one command.

Introduction

While some features can bring about the outcome when acting alone, others can only do so when acting with a set of features. This mechanism is common in drug interactions. Pravastatin (a lipid-lowering agent) and paroxetine (an antidepressant) are among the most widely prescribed drugs in the world. Although none of them associates with blood glucose, mixing the two drugs could result in high blood glucose level. A more general example is Simpson's paradox: when contingency tables are combined, the resulting table may show a relationship different from those shown by the original tables.

The above interactions (between drugs or between contingency tables) can be represented using a rule:

$$c_1 \wedge c_2 \wedge, \ldots, \wedge c_n \quad \rightarrow \quad y, \tag{1}$$

where $c_1 \wedge c_2 \wedge, \ldots, \wedge c_n$ is a conjunction of conditions (feature-value pairs, e.g., drug = used) and y the outcome (class label, e.g., high blood glucose level). The rule says that, *IF* all the conditions in the conjunction are met *THEN* y could be brought about. This rule can be either deterministic (y will absolutely be produced) or nondeterministic (y will be produced with a probability).

While rule-based classifiers (RCs hereafter) can capture the rules above, non-rule-based classifiers may not be able to do so. This could result in compromised classification accuracy when data were generated by such rules. Intuitively, this can be handled by using RCs to detect rules to assist non-rule-based classifiers (base hereafter) in classification. The problem is, most RCs were designed to find both "important" (i.e., deterministic and frequent) and "less important" (nondeterministic or rare) rules (we will give the formal definition of these rules later). This guarantees that for every new sample there is always a rule that fires (i.e., all the conditions in the rule are met), so that the class label can be predicted (by the fired rule). As a result, a RC will always propose its predicted class to the base. In simple words, a RC (fanatically) shouts "I know, I know!" all the time, leaving the base to decide whether the suggested class should be taken. However, it could be difficult for the base to decide, because it may not know whether the suggested class comes from an important rule or a less important one. While taking classes produced by important rules can improve the accuracy of the base, accepting ones obtained by less important rules may degrade the accuracy (since such results could be less accurate than the ones provided by the base itself).

To address this problem, we propose an Add-on Rule-based Classifier (ARC) that aims to detect only important (deterministic and frequent) rules. When paired with a base, ARC only classifies samples where an important rule fires, and leaves other samples to the base. In simple words, ARC (calmly) says "I am confident, let me do it" to the base when an important rule fires, and "I do not know, it is up to you" when no important rule fires. In this case, the base can simply follow ARC's instruction, since it knows that ARC only proposes classes obtained by important rules (so that ARC

Algorithm 1: *predict*($\mathbf{x}$, *base*)

1 **if** there is a rule $C \rightarrow y$ in $\mathcal{R}$ (the set of detected rules) where conjunction C is met in sample $\mathbf{x}$ **then**
2 **return** class label y
3 **else**
4 **return** *base.predict*($\mathbf{x}$)

Algorithm 2: *fit*($\mathbf{X}$, $\mathbf{y}$, *base*)

1 *base.fit*($\mathbf{X}$, $\mathbf{y}$)
2 **for** each class label y of $\mathbf{y}$ **do**
3 $\mathcal{R}_y$ = *greedy_search*($\mathbf{X}$, $\mathbf{y}$, y, $\varnothing$)
4 $\mathcal{R} = \mathcal{R} \cup \mathcal{R}_y$

is confident about what it says). Intuitively, the ensemble (of ARC and the base) should be at least as accurate as the base, since classes produced by the important rules should be at least as accurate as those provided by the base. This claim will be experimentally demonstrated later.

While we can tweak existing RCs (rule-based classifiers) to find only important rules, the ensemble of a tweaked RC and a base may still not be as accurate as the ensemble of ARC and the base. The reason is that, important rules detected by ARC can be more accurate than those found by RCs (which will later be echoed by empirical results). This is because ARC uses a new approach to search for the rules, which can handle problems in methods used by most RCs (more on this in the Related Work section).

Method

ARC has two actions: *fit* the model and *predict* the class label. We explained how ARC predicts (with the base classifier) in Introduction. That is, if an important (deterministic and frequent) rule detected by ARC fires (all the conditions in the rule are met), the class will be predicted by the rule. Otherwise, it will be predicted by the base. The above steps are summarized in algorithm 1. Here, function *predict* takes as input a new sample $\mathbf{x}$ (discrete or discretized) and the base classifier *base*, and outputs the predicted class of the sample.

We now explain how ARC fits, that is, how ARC detects important rules ($\mathcal{R}$ in line 1 of algorithm 1). The steps for fitting are summarized in algorithm 2. Here, function *fit* takes as input the samples $\mathbf{X}$ (discrete or discretized), the target $\mathbf{y}$ (discrete or discretized), and the base classifier *base*. The algorithm begins with fitting the base, then for each class label of the target, y, it finds a set of important rules, $\mathcal{R}_y$, using a method named *greedy_search*.

The Definition of Important Rule and Necessary Condition

Before moving on to the greedy search, let us formally define the two concepts that will be used frequently hereafter: important rule and necessary condition. Let C be a conjunction of conditions and y a class label. We claim that $C \rightarrow y$ is an *important* rule if it is both deterministic and frequent.

Algorithm 3: *greedy_search*($\mathbf{X}$, $\mathbf{y}$, y, C)

1 **repeat**
2 **if** detected rule $C \rightarrow y$ is important (both eqs. (2) and (3) hold) **then**
3 Remove irrelevant conditions from C
4 $\mathcal{R}_y = \mathcal{R}_y \cup \{C \rightarrow y\}$
5 Remove samples where rule $C \rightarrow y$ fires
6 **for** each condition c in C **do**
7 *greedy_search*($\mathbf{X}$, $\mathbf{y}$, y, $C \setminus c$)
8 Delete conditions in C from conditions pool
9 $C = \varnothing$
10 **else if** some conditions can be added to C **then**
11 Add the best condition to C
12 **else**
13 Remove the worst condition from C
14 **until** step 13 removes the last condition from C;
15 **return** $\mathcal{R}_y$

That is, the rule must satisfy both of the constraints below:

$$P(y \mid C) = 1, \tag{2}$$

$$\#\big(C \rightarrow y\big) \geq \textit{min\_samples\_importance}. \tag{3}$$

The first constraint says that, for any sample where the rule $C \rightarrow y$ fires (all the conditions in C are met), the class of the sample must be y. That is, the rule is deterministic. The second constraint says that, the number of samples where the rule fires cannot be smaller than a threshold. That is, the rule is frequent (with respect to the threshold). On the other hand, we claim that a rule is *less important* if it is either nondeterministic (eq. (2) fails) or rare (eq. (3) fails).

We now formally define the necessary condition. For an important rule $C \rightarrow y$, we claim that condition c in conjunction C is *necessary*, if the rule is less important when removing c from C. That is, eq. (2) fails when replacing C with $C \setminus c$ (i.e., C without c). On the other hand, we claim that a condition is *irrelevant* if the rule is still important when removing the condition from the conjunction. It turns out that, the definition of necessary and irrelevant conditions are closely related to the heuristics for selecting the "best" and "worst" conditions, which are the key of the greedy search.

The Greedy Search Algorithm

The idea of the greedy search is that, we iteratively add the "best" condition to the conjunction until we 1) find an important rule, or 2) cannot add condition anymore (due to the sparseness of data in high dimensions). We then iteratively remove the "worst" condition from the conjunction until we 1) find an important rule where every condition is necessary, or 2) can add condition again. The removal step expels irrelevant conditions, which in turn, makes room for the necessary ones. This is one of the differences between ARC and many top-down approaches (more on this in Related Work).

The steps used in the greedy search are summarized in algorithm 3. Here, function *greedy_search* takes as input the samples $\mathbf{X}$ (discrete or discretized), the target $\mathbf{y}$ (discrete or

discretized), a class label y, and a conjunction C. Here C is initialized as empty, which is different from the bottom-up approaches that begin the search with a conjunction of all the conditions (more on this in Related Work).

There are three stages in algorithm 3: the stage where the detected rule is important (steps 2 to 9), the stage where we add the best condition to C (steps 10 and 11), and the stage where we remove the worst condition from C (step 13). We now explain each stage in detail.

When the rule is important, we first remove all the irrelevant conditions (step 3 in algorithm 3). This leads to an important rule comprised of only necessary conditions. We then remove the samples where all the conditions in C (the conjunction in the rule) are met. This is a standard step in many rule-based methods (to isolate the impact of detected rules and make better choice for best and worst conditions). Next we loop over each condition c in C, and recursively call the function (steps 6 and 7). However, instead of kicking off the search with an empty conjunction (which is the case the first time we call the function, as shown in step 3 of algorithm 2), we do so with conjunction $C \setminus c$ (i.e, C without c). This helps us find rules that have many conditions in common with the detected ones. Last (steps 8 and 9), we first delete all the conditions in C from the conditions pool (initialized as the set of all the conditions), so that they will not be selected again. We then empty the conjunction. This helps us find rules that have no condition in common with the detected ones.

When the rule is less important (defined below eq. (3)), we check if there is any condition c satisfying all of the following four constraints, so that it can be added to conjunction C (step 10 in algorithm 3). First, c has not been removed from C (step 13) or deleted from the conditions pool (step 8). Second, if we add c to C the new conjunction ($C \wedge c$) will not contain all the conditions in any conjunction of any detected rule. The two constraints above prevent us from repeatedly exploring the same search space, guaranteeing that the search will eventually terminate. Third, the number of samples where $C \wedge c$ is met (both C and c are met) cannot be smaller than *min_samples_importance* (the same parameter for determining whether a rule is frequent, as shown in eq. (3)). Last, the number of samples where $C \wedge \neg c$ is met (C is met but c is not) cannot be smaller than *min_samples_importance*. The last two constraints not only help us find rules that are frequent, but also help us choose more accurate "best" conditions (more on this later). If there are conditions satisfying all of the four constraints above, we add the best of them to C (step 11). Otherwise, we remove the worst condition from C (step 13). We stop the search after removing the last condition from C (step 14).

As mentioned earlier, the key of the greedy search is the heuristics for selecting the best condition (that should be added to conjunction C) and the worst condition (that should be removed from C). Intuitively the best condition is a necessary condition that, once added, increases the predictive power of the conjunction the most. The increase of predictive power (brought about by adding condition c to C) can be written as the following difference in probabilities:

$$P(y \mid C \wedge c) - P(y \mid C). \tag{4}$$

Here, the first item is the probability of class label being y when both C and c are met, and the second is the probability when C is met. Then we can think of the best condition as the one that maximizes the above difference.

It is worth noting that the above idea for choosing the best condition has been widely used in most top-down approaches. For instance, Decision Tree defines the best condition as the one leading to the largest information gain. The difference is that, unlike eq. (4) that directly uses difference in probabilities, the measurement in Decision Tree uses difference in impurities (e.g., gini or entropy). Similar to eq. (4), the measurement also considers the increase of predictive power when a new condition is met. However, when C contains almost all of the necessary conditions, the increase (in either probabilities or impurities) could be too small to separate necessary conditions from irrelevant ones (the definition of the two kinds of conditions were given in the second paragraph below eq. (3)). As a result, an irrelevant condition could be wrongly chosen as the best one.

To address this problem, besides considering the increase of predictive power when a new condition is met, we also consider the decrease of predictive power when a new condition is not met. The decrease can be represented by replacing c in eq. (4) with its negation, $\neg c$:

$$P(y \mid C \wedge \neg c) - P(y \mid C). \tag{5}$$

Here the first item is the probability of class label being y when C is met but c not met, and the second item is the probability when C is met. The idea is that, (when C contains almost all of the necessary conditions) the first probability in the equation will be close to 0 when c is necessary, but close to 1 when c is irrelevant. Thus we can also think of the best condition as the one that minimizes eq. (5).

Since (as mentioned below eq. (4)) the best condition can also be thought of as the one that maximizes eq. (4), we combine the two measurements so that the best one can stand out. That is, in this paper the *best* condition is defined as the one that not only maximizes eq. (4) but also minimizes eq. (5) or, in other words, maximizes eq. (4) minus eq. (5), which can be written as

$$P(y \mid C \wedge c) - P(y \mid C \wedge \neg c). \tag{6}$$

This equation can also be thought of as the difference of predictive power when c works for C (the first probability in the equation) and when c works against C (the second probability). Then the best condition is the one that maximizes the difference. To the best of our knowledge, this is the first time such heuristic (combining the maximization of eq. (4) and minimization of eq. (5)) is used in rule search.

Now let us recall the last two constraints for deciding whether there is any condition c that can be added to conjunction C (step 10 in algorithm 3). That is, both the number of samples where $C \wedge c$ is met, and the number of samples where $C \wedge \neg c$ is met, cannot be smaller than *min_samples_importance* (the parameter in eq. (3)). The two constraints allow us to have enough number of samples when using eq. (6) to choose the best condition.

It turns out that, the worst condition can be defined by replacing C in eq. (6) with $C \setminus c$ (i.e., C without c). The idea

is that, unlike the best condition which is not in C (so that we can add it to C), the worst condition is actually in C (so that we can remove it from C). After replacing C with $C \setminus c$, eq. (6) can be written as

$$P(y \mid C \setminus c \wedge c) - P(y \mid C \setminus c \wedge \neg c). \quad (7)$$

Here, the first item is the probability of class label being y when C (which contains c) is met, and the second is the probability when $C \setminus c$ (which does not contain c) is met but c not met. Then similar to eq. (6), eq. (7) measures the difference of predictive power when c works for $C \setminus c$ (the first probability in the equation) and when c works against $C \setminus c$ (the second probability). However, unlike the best condition which maximizes eq. (6), the *worst* condition is defined as the one that minimizes eq. (7).

Soundness, Completeness, and Minimality

Theorem 1. *The rules detected by ARC (using algorithm 3) are sound, complete, and minimal.*

Here the soundness property ensures that ARC will only find important (deterministic and frequent) rules. The completeness ensures that (in theory) ARC will find all the important rules, and the minimality ensures that the rules are the most generalized. The last two properties guarantee that the rules will cover the most possible samples (where all the conditions in one of the rules are met). The three properties as a whole guarantee that ARC will make use of the most possible samples (covered by the important rules) to assist the base classifier in classification. Further, such assistance (in theory) will only improve (rather than degrade) accuracy.

Proof. We first prove that the rules detected by ARC are sound. That is, they are important, satisfying eqs. (2) and (3). This is guaranteed by step 2 in algorithm 3, where we accept a rule only when both equations are met.

We next prove that the detected rules are complete. Assuming enough number of samples, algorithm 3 will find at least one important rule, if there is any. This is because in the worst case we will find a conjunction including all the possible conditions and all the irrelevant conditions will be removed by step 3 in the algorithm. For the same reason, all the rules that have conditions in common with the detected rule will be identified by steps 6 and 7. Similarly, all the rules that have no condition in common with the detected rule will be identified due to steps 8 and 9. The idea above was also discussed in the third paragraph above eq. (4).

Last we prove that the detected rules are minimal. That is, all the conditions in the conjunction of a rule are necessary. Similar to the proof for the completeness, this is guaranteed by step 3 in algorithm 3, where we remove all the irrelevant conditions from the conjunction of an important rule. □

Time Complexity

Let m be the number of samples and n the number of conditions. Then the time complexity of computing eqs. (6) and (7) is $O(mn)$. Then the complexity of updating a conjunction (by adding the best condition or removing the worst) each time is $O(mn^2)$, since it entails at most n times of calculating eq. (6) or (7). Then the complexity of detecting an important rule (that satisfies eqs. (2) and (3)) is $O(mn^3)$, since it entails at most $2n$ times of updating the conjunction. In turn, the complexity of the greedy search (algorithm 3), which is the most time consuming part in ARC, is $O(rmn^3)$, with r being the number of important rules. As a result, the complexity of ARC is also $O(rmn^3)$. In reality n is usually much higher than r, indicating that the average time complexity of ARC could be $O(mn^4)$.

Related Work

Discovering rules in the form of *IF* conditions *THEN* outcome (eq. (1)) has been widely studied in areas such as statistical relational learning, inductive logic programming, and association rule mining.

One kind of methods detect rules by starting with a conjunction of all the conditions and iteratively removing the "worst" condition from the conjunction. Such methods are sometimes referred to as the bottom-up (or generalization) approaches (Michalski 1983; Valiant 1984). However, due to the sparseness of data in high dimensions, the bottom-ups are infeasible when there are moderately large number of conditions in the conjunction.

Contrary to the bottom-ups (which identify rules by iteratively removing the "worst" condition from the conjunction), the top-down (or specialization) approaches detect rules by iteratively adding the "best" condition to the conjunction. Since the top-downs usually begin with an empty conjunction, they are more practical than the bottom-ups (which start with a conjunction of all the conditions). Along the line of top-downs (e.g., AQ15 (Michalski et al. 1986), FOIL (Quinlan 1990), nFOIL (Landwehr, Kersting, and De Raedt 2005), DL-FOIL (Fanizzi, d'Amato, and Esposito 2008), IREP (Fürnkranz and Widmer 1994)), the most widely used one is Decision Tree, with its early implementations such as CART (Breiman et al. 1984) and ID3 (Quinlan 1986). Later, ID3 was first extended to PRISM (Cendrowska 1987), which induces rules that are modular (rather than in the form of trees), and then to C4.5 (Quinlan 2014), which allows continuous features and pruning the trees. While methods such as Apriori (Agrawal and Srikant 1994) are also related to the top-down category, they focus on finding all the association rules (that satisfy some minimum support and confidence constraints), some of which may not be suitable for classification. This problem was later addressed by methods such as CARs (Liu, Hsu, and Ma 1998), CMAR (Li, Han, and J. 2001), and Supervised_Apriori_Gen (Ding et al. 2006) (most of which aim to identify a special subset of association rules that can be used for classification). Apriori was also improved by approaches such as (Wu, Zhang, and Zhang 2002), which allows both positive and negative association rules.

Unlike the top-down and bottom-up approaches that either add or remove conditions, ARC (the proposed method) combines the two by alternating between adding and removing conditions. Compared to the bottom-ups, ARC allows much larger datasets since it starts the search from an empty conjunction (rather than a conjunction of all the conditions, as in the bottom-ups). Compared to the top-downs, on the

Table 1: Statistics of 12 UCI datasets used in the experiment. Here # S and # F denote the number of samples and features.

No.	Name	# S	# F
1	balloon (adult-stretch)	20	4
2	balloon (adult+stretch)	20	4
3	balloon (yellow-small)	20	4
4	balloon (yellow-small+adult-stretch)	16	4
5	breast-cancer-wisconsin	699	10
6	king-rook-vs-king-pawn	3196	36
7	monks-problems (1)	556	7
8	monks-problems (2)	601	7
9	monks-problems (3)	554	7
10	mushroom	8124	22
11	tic-tac-toe endgame	958	9
12	voting-records	435	16

other hand, ARC enables to remove irrelevant conditions not only after the search (as in the top-downs) but also during the search, which can make room for necessary conditions. Thus the rules detected by ARC could be more accurate. This claim will later be experimentally demonstrated by comparing the rules detected by ARC and Decision Tree.

It is worth noting that there are other methods that, like ARC, also allow searching for rules in both directions (by alternating between adding and removing conditions). One such work is JoJo (Fensel and Wiese 1993), which searches for rules using both specialization (adding the best condition) and generalization (removing the worst). However, unlike ARC (but similar to most of the existing methods), the heuristics used by JoJo (named s-preference and g-preference) for determining the best and worst conditions do not consider the decrease of predictive power when a new condition is not met. That is, the second probability in eqs. (6) and (7), which can help us separate necessary conditions from irrelevant ones (see below eq. (5)) are ignored.

Empirical Results

The main goal here is to experimentally demonstrate what we theoretically proved. That is, when paired with a base classifier, ARC should almost never degrade the accuracy of the base and, more importantly, usually improve the accuracy. This was illustrated by the empirical results of the ensemble of ARC and 9 leading classifiers on 12 UCI datasets. The code, data, and full results are publicly available in our github repository[1]. The results are reproducible by simply using one command.

Data and Method

Since ARC is a classifier that requires categorical data, we used 12 UCI datasets under category *Classification* + *Categorical*, summarized in table 1.

We paired ARC with 9 leading *sklearn* classifiers: AdaBoost (AB for short), Decision Tree (DT), Gaussian Naive Bayes (GNB), Gaussian Process (GP), K-Nearest Neighbors (KNN), Logistic Regression (LR), Multi-layer Perceptron (MLP), Random Forest (RF), and Support Vector Machine (SVC). The default settings (in *sklearn*) were used for each classifier.

[1]https://github.com/yuxiaohuang/research/tree/master/gwu/accepted/flairs_2020/arc

Results

The average classification accuracy (obtained by cross-validation) on each of the 12 UCI datasets are summarized in table 2. The first column is the identifier of the datasets (whose name can be seen in column *Name* in table 1). The remaining columns are the accuracy. The results show that, ARC never degraded the accuracy of any base classifier in any dataset. Further, symbols $\triangle$ (denoting increase of accuracy) and $\blacktriangle$ (denoting significant increase, with p-value < 0.05) show that, ARC improved the accuracy of each of the 9 bases in many datasets, where some of the increases are significant. For example, when paired with GNB, ARC improved the accuracy in 8 out of 12 datasets (with 4 increases being significant). This is also the case for KNN (8 increases with 3 being significant), AB (6 increases with 2 being significant), GP (5 increases with 2 being significant), DT (4 increases with 2 being significant), LR and SVC (4 increases with 1 being significant), MLP (3 increases with 1 being significant), and RF (5 increases). These results experimentally demonstrated what we theoretically proved. That is, when using the detected important rules to assist the base classifier (in classification), ARC should almost never degrade the accuracy of the base and, more importantly, often improve the accuracy.

It is worth noting that the claim above not only applies to base classifiers that are not rule-based, but also to ones that are actually rule-based. This can be seen from the number of accuracy increase (4 with 2 being significant) and decrease (0) for DT (Decision Tree), as reported above. Such results unveil an interesting insight: the important rules detected by ARC are not simply a subset of the rules detected by DT. This is because if that were the case, classes predicted by important rules of ARC would be the same as those predicted by DT, and we would not see any change in DT's accuracy.

The claim above is echoed in the tic-tac-toe endgame dataset (No. 11 in table 2). There are 9 features in the dataset, representing the 9 cells on the game board. Each feature has 3 values, x, o, b, where x means the cell is taken by player x, o means the cell is taken by player o, and b means the cell is blank. The goal here is to predict whether player x will win (class label *Positive*) or not (*Negative*). In one training set, ARC found 8 rules with respect to class Positive, where each rule has 3 conditions. These rules represent 8 cases (3 rows, 3 columns, and 2 diagonals) where player x has a "three-in-a-row" so that player o will definitely lose (class Positive). In the same training set, ARC found 6 rules representing 6 cases (3 rows, 1 column, and 2 diagonals) where player o has a "three-in-a-row" so that player x will definitely lose (class Negative). However, from the same training set DT found rules containing up to 11 conditions, and some of these rules lead to wrong classification. This is the reason why the accuracy of DT in the corresponding testing set is 0.72, whereas the accuracy of the ensemble (of DT and ARC) is 0.98. This is also part of the reason why on this dataset the improvement of accuracy for DT (brought about by ARC) is signif-

Table 2: The average classification accuracy (obtained by cross-validation) on each of the 12 UCI datasets shown in table 1. For each entry, the number on the left-hand side of + sign is the average accuracy of the base classifier (denoted by the column name) in the dataset (denoted by the row number), and the number on the right-hand side of + is the average increase of accuracy when pairing the base with ARC (no decrease was found for any base in any dataset). Further, symbol △ means that ARC increased the accuracy of the base but the change is not significant, and ▲ denotes significant increase (p-value < 0.05).

No.	AB	DT	GNB	GP	KNN	LR	MLP	RF	SVC
1	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00
2	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00
3	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00	1.00 + 0.00
4	0.67 + 0.33 ▲	0.78 + 0.22 ▲	0.56 + 0.33 ▲	0.72 + 0.28 ▲	0.72 + 0.28 ▲	0.67 + 0.33 ▲	0.67 + 0.33 ▲	0.72 + 0.17 △	0.56 + 0.33 ▲
5	0.97 + 0.00	0.96 + 0.01 △	0.96 + 0.01 △	0.98 + 0.00	0.96 + 0.01 △	0.97 + 0.00	0.98 + 0.00	0.98 + 0.00	0.97 + 0.00
6	0.95 + 0.01 △	0.99 + 0.00	0.59 + 0.33 ▲	0.88 + 0.07 ▲	0.83 + 0.11 ▲	0.95 + 0.01 △	0.95 + 0.02 △	0.96 + 0.01 △	0.92 + 0.03 △
7	0.67 + 0.12 △	1.00 + 0.00	0.75 + 0.11 △	0.86 + 0.03 △	0.79 + 0.08 △	0.73 + 0.11 △	1.00 + 0.00	0.95 + 0.01 △	0.75 + 0.11 △
8	0.40 + 0.01 △	0.93 + 0.00	0.40 + 0.01 △	0.70 + 0.00	0.60 + 0.01 △	0.41 + 0.00	1.00 + 0.00	0.88 + 0.00	0.51 + 0.00
9	0.94 + 0.01 △	0.96 + 0.00	0.96 + 0.01 △	0.97 + 0.00	0.85 + 0.06 △	0.97 + 0.00	0.98 + 0.00	0.98 + 0.00	0.97 + 0.00
10	1.00 + 0.00	0.99 + 0.00	0.94 + 0.05 ▲	1.00 + 0.00	0.99 + 0.00	0.98 + 0.00	1.00 + 0.00	1.00 + 0.00	0.96 + 0.00
11	0.77 + 0.15 ▲	0.86 + 0.09 ▲	0.56 + 0.31 ▲	0.90 + 0.06 △	0.70 + 0.22 ▲	0.97 + 0.01 △	0.97 + 0.02 △	0.89 + 0.06 △	0.89 + 0.05 △
12	0.96 + 0.00	0.96 + 0.01 △	0.94 + 0.00	0.95 + 0.01 △	0.93 + 0.02 △	0.96 + 0.00	0.97 + 0.00	0.97 + 0.01 △	0.96 + 0.00

icant (as indicated by the ▲ symbol in entry (row No. 11, column DT) of table 2).

Besides DT, ARC also improved the accuracy of all the other 8 base classifiers in the tic-tac-toe endgame dataset, where the improvement for AB, GNB, and KNN are significant. The increase of accuracy is due to the fact that the rules detected by ARC are more accurate than the patterns identified by others. This is also the reason for the improvement of accuracy in the balloon dataset (No. 4 in table 2), with 9 increases (8 being significant), the king-rook-vs-king-pawn dataset (No. 6), with 8 increases (3 being significant), and monks-problems datasets, with 7 increases for monks 1 (No. 7), and 3 increases for monks 2 and 3 (Nos. 8 and 9).

It is worth noting that ARC did not improve the accuracy of any base classifier in the first 3 datasets in table 2. This is because these data were generated by simply rules which can be well captured by each of the 9 bases.

Conclusion

We proposed an Add-on Rule-based Classifier, ARC, which can be paired with any existing classifier (base). The idea of ARC is improving the base's accuracy by detecting deterministic and frequent rules. We theoretically proved that ARC will almost never degrade the accuracy of the base, and more importantly, usually improve the accuracy. This claim was demonstrated by empirical results of 9 leading classifiers on 12 UCI datasets, where ARC never lowers the accuracy of the base, but instead, usually increases it (with some of the increases being statistically significant). In the future we will extend ARC to allow both continuous features and nondeterministic rules.

References

Agrawal, R., and Srikant, R. 1994. Fast Algorithms for Mining Association Rules. In *VLDB*.

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks.

Cendrowska, J. 1987. PRISM: An Algorithm for Inducing Modular Rules. *International Journal of Man-Machine Studies* 27(4):349–370.

Ding, W.; Eick, C. F.; Wang, J.; and Yuan, X. 2006. A Framework for Regional Association Rule Mining in Spatial Datasets. In *ICDM*.

Fanizzi, N.; d'Amato, C.; and Esposito, F. 2008. DL-FOIL Concept Learning in Description Logics. In *ILP*.

Fensel, D., and Wiese, M. 1993. Refinement of Rule Sets with JoJo. In *ECML*.

Fürnkranz, J., and Widmer, G. 1994. Incremental Reduced Error Pruning. In *Machine Learning Proceedings 1994*. Elsevier. 70–77.

Landwehr, N.; Kersting, K.; and De Raedt, L. 2005. nFOIL: Integrating Naıve Bayes and FOIL. In *AAAI*.

Li, W.; Han, J.; and J., P. 2001. CMAR: Accurate and Efficient Classification based on Multiple Class-Association Rules. In *ICDM*.

Liu, B.; Hsu, W.; and Ma, Y. 1998. Integrating Classification and Association Rule Mining. In *KDD*.

Michalski, R. S.; Mozetic, I.; Hong, J.; and Lavrac, N. 1986. The Multi-purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. In *AAAI*.

Michalski, R. S. 1983. A Theory and Methodology of Inductive Learning. *Artificial Intelligence* 20(2):111–161.

Quinlan, J. R. 1986. Induction of Decision Trees. *Machine Learning* 1(1):81–106.

Quinlan, J. R. 1990. Learning Logical Definitions from Relations. *Machine Learning* 5(3):239–266.

Quinlan, J. R. 2014. *C4. 5: Programs for Machine Learning*. Elsevier.

Valiant, L. G. 1984. A Theory of the Learnable. *Communications of the ACM* 27(11):1134–1142.

Wu, X.; Zhang, C.; and Zhang, S. 2002. Mining both Positive and Negative Association Rules. In *ICML*.