Rule-based classifier: RULES

Albert Cardenete Massip

April 2019

1 Introduction

In this first practical delivery for the SEL course we are going to study the RULES algorithm, which is a simple rule-based classifier. First we present the algorithm itself, by explaining its main parts. Then, we are going to apply this model to three different datasets and we will compare the results of the predictions when we have different policies to choose the final prediction.

2 The RULES algorithm

The algorithm that has been implemented is the RULES algorithm [1]. It is a simple inductive algorithm for extracting if-then rules from a set of training examples. In this section it the development of the code will be shown.

First we will have to introduce some formalism. Let $\mathcal{C} = \{c_1, \ldots, c_k\}$ be the set of elements in the class attribute of a given domain, having k different classes, which is also called response variable. Let $\mathcal{O}$ be the set of all possible values for the explanatory variables $\mathcal{O} = \mathcal{O}_1 \times \ldots \times \mathcal{O}_m$, where m is the total number of explanatory variables in which every variable $\mathcal{O}_j = \{o_{j_1}, \ldots, o_{j_l}\}$ has j_l possible values. A dataset $\mathcal{D}$, is an a subset of the cartesian product of the response and observed variables, $\mathcal{D} \subset \mathcal{O} \times \mathcal{C}$, for a given domain.

The RULES algorithm developed can be seen in the algorithm 1. Now we are going to go through it explaining what it does. First of all, we will have a training dataset $\mathcal{D}_1$, and we will start by initializing all the relevant values (lines 1 to 4). We will also store a second dataset, which will contain all the unclassified training instances, which will begin being the training dataset as we do not have any rule at the beginning.

After the initialization, we will perform the main loop of the algorithm (lines 6 to 24). In this loop, we will be creating the rules with an increasing number of explanatory variables. At every run of the main loop, we will be generating all the candidate rules that can be extracted from the unclassified dataset $\mathcal{D}_2$ (line 8). These candidate rules with a number of conditions p, R_p will be created from all the remaining attribute values $R_p = \mathcal{O}_{2_{\sigma(1)}} \times \ldots \times \mathcal{O}_{2_{\sigma(p)}}$, $\forall \sigma \in S_m$, where σ is a permutation of the number of explainatory variables, m. This means that we will generate candidate rules for all the possible values of the permutations of length p of explanatory variables.

After that, we are going to check which of the candidate rules is a true rule for the training dataset (lines 9 to 15). This means that we will test if a given rule correctly separates the training dataset, by having all the instances that accomplish the rule the same class. We will only consider those rules.

Then, if the number of current conditions is more than one, we might have rules with irrelevant conditions, so we would like to discard those rules (lines 16 to 18). A rule is irrelevant if a simpler rule that we have already achieved already classifies all the instances. This is convenient as we will be generating the most general possible rules that completely separates our training dataset. This can be done easily by checking if a simple rule that we already have stored is a subset of a new rule. The new rule will be a more specific rule, that in general would have less coverage, and thus we will discard, as is also does not add more discriminatory power to our model.

After we have all the new generated instances, we will update the unclassified instances with our new rules and we will check if we have ended classifying (lines 19 to 23). Finally, if we have reached the maximum number of conditions for our rules and we still have unclassified instances, we will generate a rule for each of the individual unclassified instances and we will end the algorithm.

Algorithm 1 RULES Algorithm

```
    Input $\mathcal{D}_1$
 1: max_conditions ← $m$
 2: conditions ← 0
 3: all_examples_are_classified ← False
 4: rules ← {}                                          ▷ Final rules obtained
 5: $\mathcal{D}_2$ ← $\mathcal{D}_1$                   ▷ $\mathcal{D}_2$ is the dataset of the non classified instances
 6: while (NOT all_examples_are_classified) AND (conditions≤max_conditions) do
 7:     conditions ← conditions + 1
 8:     candidate_rules ← get_candidate_rules($\mathcal{D}_2$, conditions)
 9:     valid_rules ← {}
10:     for all rule ∈ candidate_rules do
11:         is_good_rule ← rule_tester(rule, $\mathcal{D}_1$)
12:         if is_good_rule then
13:             valid_rules ← [valid_rules, rule]
14:         end if
15:     end for
16:     if conditions > 1 then
17:         valid_rules ← relevant_conditions(valid_rules,rules)
18:     end if
19:     $\mathcal{D}_2$ ← get_unclassified_instances(valid_rules,$\mathcal{D}_2$)
20:     rules ← [valid_rules, rules]
21:     if instances($\mathcal{D}_2$) = 0 then
22:         all_examples_are_classified ← True
23:     end if
24: end while
25: rules ← [rules, generate_final_rules($\mathcal{D}_2$)]
```

3 Experimental Setup

In order to test our algorithm, we will perform a k-fold cross validation over different datasets. However, it will not be as straightforward as plugging the dataset into the RULES algorithm and extracting the rules.

First of all, we will perform a simple preprocessing to our training dataset in order to improve the results. Whenever we have more than one instance that has the same values for all the explanatory variables, we will get just one of them randomly, and we will discard the rest of the instances. Otherwise, if the duplicated instances had a different class value, we would not be able to classify any of those instances using RULES.

After having all the rules generated rules with the training set, we will have to compute the accuracies of the validation and test sets. To do so, we will have to be able to predict the class of a given instance from the rules. Although it might seem easy, by checking if we have a rule that correctly classifies that instance, it is not that easy. By following that idea, we might have unreal accuracies. For instance, we could have a rule that correctly classifies a given instance, although we could have a different rule, which could be applied to the explanatory variables of that instance, but leading to a different predicted class. In that case, We would have to consider that as incorrectly classified instance, as in the reality we would not know which rule to trust. This situation might be overcame by giving a weight to each rule, although we will not do it in these experiments.

Although that, we are going to implement three different voting mechanisms. The first of all, will be based on unanimity. Only if all the rules that might apply to a given instance predicts correctly the class we will count that as a correctly classified instance. The second voting mechanism will be based on the majority of votes, the predicted class will be the one which have more rules that predict that class. If there is a draw, we will choose randomly the final prediction. Finally, the last voting mechanism will be based on the majority of rules weighted by the coverage (equation (1)) of each rule. As before, in case of draw, we will choose randomly one of the most voted predictions.

$$\text{Coverage} = \frac{\#\text{Instances predicted by the rule in training set}}{\#\text{Training Set}} \tag{1}$$

In order to get an estimation of the performance of our resulting model in every dataset studied, we will perform a k-fold cross validation by splitting the data. Then we are going to apply the different voting mechanisms proposed in order to achieve better results. Finally, we will merge all the dataset again to extract all the rules that the best model can generate (as while having more data, we expect our model to be better), and we will also compute the coverage and precision for each of the rules.

The datasets studied in this work will be extracted from the UCI ML repository[2], which are:

- **Breast Cancer:** Containing 286 instances with 9 categorical attributes and 2 classes.
- **Nursery:** Containing 12960 instances with 8 categorical attributes and 5 classes.
- **Voting records:** Containing 435 instances with 16 categorical attributes and 2 classes.

4 Results and discussion

The results of the 3-fold cross validation test can be seen in the table 1. In these results we clearly see the importance of the chosen method of the prediction. Although it is not strictly part of the RULES algorithm, it will hardly happen that a rule that might apply to a given instance gives us the correct prediction.

Dataset	Prediction Method	Accuracy	Time [s]
Breast Cancer	Unanimity	0.293 ± 0.052	10.6 ± 1.7
	Majority	0.689 ± 0.033	
	Most Coverage	0.703 ± 0.047	
Voting	Unanimity	0.605 ± 0.092	18.5 ± 7.5
	Majority	0.9517 ± 0.0056	
	Most Coverage	0.947 ± 0.012	
Nursery	Unanimity	0.531 ± 0.026	168.8 ± 3.5
	Majority	0.765 ± 0.014	
	Most Coverage	0.9755 ± 0.0011	

Table 1: Results of the evaluation of the 3-fold cross validation for the RULES algorithm.

As we can see, by applying the policy of majority of votes, the results improve significantly. The weighted approach to the prediction also gives us better performances, although it is only significantly better in the case of the nursery dataset. more statistical tests would be needed in order to decide of the weighted method is better for any database than the majority method.

We can also see the time results. Here we can see the large amount of time that takes to train the nursery dataset in comparison with the other ones. This comes from the larger amount of instances that this last dataset has. Despite the fact that the algorithm has a factorial complexity [1], it scales almost linearly.

Another thing that we can take a look into is the number of generated rules per dataset when considered all the instance. This can be seen in the table 2. There we can see that sometimes we can have a larger number of rules than instances, which is the case of the Breast cancer and Voting datasets. This might be due to the large amount of different variables across all the explanatory variables. In the case of the nursery dataset we had 27 total attribute values, while the breast cancer and voting datasets have 53 and 47 respectively.

Dataset	No. Instances	No. Variables	No. Rules
Breast Cancer	286	9	1282
Voting	435	16	1473
Nursery	12960	8	638

Table 2: Results of the evaluation of the 3-fold cross validation for the RULES algorithm.

5 How to run

First of all, we will set up our python environment with the required packages. Those are pandas (≥0.23.0), scikit-learn (≥0.20.3) and numpy (≥1.15.4).

In order to run the code, first we will have to enter into the `Practical` folder in a console. Then, without entering into the Source folder, the command to run the code is simply:

```
python Source
```

This command has some flags, which can be seen using the help command:

```
python Source -h
```

The list of all the flags are:

`--dataset`: The name of the file inside the `Practical/Data` folder ended with `.csv`.

`--classname`: The name of the target class in the previous csv file. The default is `"Class"`

`--k`: (in lower letters) The number of folds for the validation process. The deault is 3.

`--no-validate`: If written, we will only compute the rules for the whole dataset, without validating.

So for example, if we have the dataset "`DS.csv`" inside the Data folder with the target class named "`Cls_attr`", if we would like to perform the validation and extraction of all the rules with a 3-fold cross calidation, we would write:

```
python Source --dataset DS.csv --classname Cls_attr
```

6 Conclusions

To conclude, we see that the RULES method to generate IF-THEN rules is a very simple, yet effective model for small datasets. Due to its computational complexity, it is unfeasible to be applied to large datasets, but we have seen that by applying this method to a dataset with more than ten thousand instances the time taken to train has been almost linear.

Although that, this algorithm needs some type of weighting method in order to generate the final results, as most of the times several extracted rules might be applied to a given test instance. Finally, we have seen that with a majority voting method or a weighted method by the coverage of the rules might yield to good results.

References

[1] Pham, D. T., Aksoy, M. S. (1995). RULES: A simple rule extraction system. Expert Systems with Applications, 8(1), 59-65.

[2] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [`http://archive.ics.uci.edu/ml`]. Irvine, CA: University of California, School of Information and Computer Science.