Rules Based Classifiers: Rules

Eric Cañas Tarrasón

Course: 2019/2020

Contents

1 Introduction

The following report will present the implementation and the analysis of the ***RULES* algorithm**, proposed by *D. T. Pham* and *M. S. Aksoy* at 1995 [1]. *RULES* algorithm its framed on the field of ***Rule-Based Classifiers***.

Rule-Based Classifiers are a group of algorithm designed to fit a set of rules in the following form: ***IF*** *Attribute*$_i$ ***= A AND*** *Attribute*$_j$ ***= B THEN Class = C***. Concretely, *RULES* (*RULe Extraction System*), is presented by its authors as a simple heuristic **inductive learning** algorithm for automating the process of extracting these *IF THEN* rules for the system, from a set of training examples. Being each one of this rules a **conjunction**, in the form: $Condition_0$ AND $Condition_1$ AND ... $Condition_n$.

Moreover, some **optimizations** over the original algorithm will be proposed and analyzed in order to help it when dealing with the those cases where is highly difficult for the algorithm to **computationally** process.

2 Implementation

Basic algorithm could be summarized as code 1 presents:

Algorithm 1 RULES Implementation

Data: Set of categorical examples **X** with **C** attributes. Set of his expected outputs **Y**.
Result: Set of rules **R**.

```
$n_c \leftarrow 1$
while ($n_c < C$) OR (All instances are classified) do
    // Explore all combinations of $C$ attributes taking $n_c$ at a time
    foreach $Comb_i \in ({}_C C_{n_c})$ do
        $Selectors_{Comb_i} \leftarrow$ Each combination of Attribute-Value with attributes within $Comb_i$
        foreach $Selector_j \in Selectors_{Comb_i}$ do
            if All $X$ elements satisfying $Selector_j$ have the same $Y$ value (From now on $Y_j$) then
                // Create and append this rule to $R$
                R $\leftarrow R + \{IF$(selector$_j$ - Irrelevant Conditions) THEN $Y_j\}$
            end
        end
    end
    $n_c \leftarrow n_c + 1$
end
// If any instance remains unclassified
foreach $UnclassifiedInstance \in X$ do
    $R \leftarrow R$ + specificRules(UnclassifiedInstance)
end
return R
```

As seen, basically, rules explores the complete combinatory of attributtes and selectors, searching for the ones reaching the **100% of precision** to create a rule. Each time it defines a rule, verifies that it does not contain irrelevant conditions and add it to the **ruleset** *R*.

However, this **simplicity** could provoke, sometimes, catastrophic resources problems. This is the reason why the following optimization on the algorithm is proposed in code 2 and its impact will be analyzed below.

Algorithm 2 RULES Implementation with the proposed optimization flags

```
   Data: Set of categorical examples $\mathbf{X}$ with $\mathbf{C}$ attributes. Set of his expected outputs $\mathbf{Y}$.
   Result: Set of rules $\mathbf{R}$.
1  $n_c \leftarrow 1$
2  $(X_{unc}, Y_{unc}) \leftarrow (X, Y)$ // Instances which remains unclassified
   // This performance optimization will be discussed below
3  if Highly Invasive Detect Unclassifiable Optimization enabled $^{1a}$ then
       // Detect classifiable ($X_{unc}$) and unclassifiable ($X_{unclassifiable}$) samples
4      $(X_{unc}, Y_{unc}, X_{unclassifiable}, Y_{unclassifiable}) \leftarrow$ detectUnclassifiable($X_{unc}$, $Y_{unc}$)
5  end
6  while ($n_c < C$) OR (All instances are classified) do
       // Explore all combinations of $C$ attributes taking $n_c$ at a time
7      foreach $Comb_i \in ({}_{C}C_{n_c})$ do
8          $Selectors_{Comb_i} \leftarrow$ Possible combinations of values for the attributes $Comb_i$ $^2$
9          foreach $Selector_j \in Selectors_{Comb_i}$ do
10             if All $X_{unc}$ elements satisfying $Selector_j$ have the same $Y$ value (From now on $Y_j$) then
                   // Remove $X_{unc}$ elements satisfying $Selector_j$ from $X_{unc}$ and $Y_{unc}$
11                 $(X_{unc}, X_{unc}) \leftarrow (X_{unc}, Y_{unc})$ - $(X_{classified}, Y_{classified})$
                   // Create and append this rule to $R$
12                 R $\leftarrow R + \{IF($selector$_j$ - Irrelevant Conditions) THEN $Y_j\}$
13                 if All instances are classified OR (all instances unclassified are unclassifiable$^{1b}$) then
14                     go to 21
15                 end
16             end
17         end
18     end
19     $n_c \leftarrow n_c + 1$
20 end
21 if Highly Invasive Detect Unclassifiable Optimization enabled $^{1a}$ then
       // Unique possible unclassified instances are the first detected as unclassifiable
22     $(X_{unc}, Y_{unc}) \leftarrow (X_{unclassifiable}, Y_{unclassifiable})$
23 end
24 foreach UnclassifiedInstance $\in X_{unc}$ do
25     R $\leftarrow$ R + specificRules(UnclassifiedInstance)$^3$
26 end
27 return $\boldsymbol{R}$
```

The optimization flags proposed here attempts to **heuristically** avoid this cases **improving the performance**. If all of them are **disabled**, code 2 turns equal to code 1.

Clarifications about the optimizations included and controversial parts:

1. *(Line 3 and 21 or Line 13)* The purpose of the ***"Detect Unclassifiable Optimization"*** is to allow the algorithm to heuristically avoid the combinatorial explosion when is sure that it will occurs.

   RULES ensures to be 100% precise if there are no duplicated instances with different class labels (***unclassifiable* instances**). However, if they exists, does not propose to deal with this cases until last step. It implies that in some usual cases, as the one proposed in section 2.1 (where this optimization will be discussed) enable this modification could be the difference between waiting a **few seconds** or a few hours or even years for the results of a *dataset.* There are two possibilities of implementing this optimization:

(a) *(Line 3)* **Highly invasive modification of the algorithm:** This is the most invasive one with the original algorithm proposal.

This optimization search which instances are unclassifiable and **remove** them from the dataset before starting the **combinatorial search** and until it ends. This removing could lead to a different *ruleset* than original proposal. In following section will be analyzed if this change is in a positive or negative way.

(b) *(Line 13)* **Use it only as stop condition:** This modification computes, each time a rule is generated, whether or not all the instances which remains unclassified are **unclassifiable**. If it is the case, is possible to ensure that no more rules could be generated, so **breaks** the combinatorial exploration.

In consequence, the time spent will be slightly higher than last proposal, but it is insurable that the output *ruleset* will be exactly the same that in the original algorithm.

2. *(Line 8)* The decision of selectors proceeds as follow. For example: Supposes a case were we combine the two attributes shown in table 1.

Trees	**Temperature**
Yellow	Low
Leafless	Average
Yellow	Low

Table 1: Selectors decision example

In this case, algorithm will only try the combinations with **ensures** his appearance in the unclassified dataset (X_{unc}): These are {Yellow, Low} and {Leafless, Average}. (Not trying the non existent combinations as {Yellow, Average}).

3. *(Line 24)* Finally, it is important to clarify how specific rules has been implemented. When cases like the one proposed in table 2 appears:

Trees	**Temperature**	**Season (Class)**
Yellow	Low	Autumn
Yellow	Low	Winter
Yellow	Low	Autumn

Table 2: Selectors decision example

These are considered as *unclassifiable.* In these cases *specificRules()* means to generate a *rule* where uses **all the attributes** and generates as conclusion the most repeated Class. In this case, for example, rule generated would be: ***IF*** *Trees are Yellow AND Temperature is Low* ***THEN*** *Season is Autumn.*

2.1 Optimization against the combinatorial explosion (*Mushroom Dataset modified*)

Unclassifiable instances are them which have two or more **different Y** values for duplicated instances. Let's see in a **practical case** the implications of enabling or disabling this optional optimization flag, as well as how the algorithm manages when cases like it are present into a dataset.

Mushroom dataset is an usual dataset used for comparing the performance of different machine learning algorithms. It is considered as an easy case, but in order to simulate a most difficult one we will use as a objective variable one more inconsistent, the habitat. With this modification, it is composed by 8,125 examples (5,644 if rows with missing values are erased), with 22 attributes each one. Between them 2,632 are *unclassifiable*, and the rest of them could be classified with at most 2 attributes.

- If optimization is **enabled** (1a, or 1b), we could ensure that the maximum number of attributes to combine in worst case will remain being C ($\max|c| \leq C$), but in most of the practical cases we will never need to reach this value. Using this modification of *Mushroom Dataset* as example, we will only explore a total of **26 combinations** in 1a optimization or **116** in 1b, in order to classify all classifiable elements (never exploring beyond ${}_{22}C_2$), and taking the process less than **5 seconds** to compute in both cases.
- If optimization is **disabled** we would be in the same scenario, but without knowing that *unclassifiable* instances exists until exploring all combinations $[{}_{22}C_1,\ {}_{22}C_2 ... {}_{22}C_{22}]$. It is a total of **4,194,302** combinations, taking the same process more than **42 hours**. If the number of attributes would be for example 40 (which is not unusual in real cases, the same process would need to explore more than 10^{12} **combinations**, taking then more than **4865 years** to compute.

Table 3 summarizes the aforementioned performance affectation.

Case	**Combinations**	**Selectors**	**Rules**	**Time Spent**	**Accuracy**
Mushrooms with Opt. 1a	26	74	24+1,100	0.92 seconds	62.7%
Mushrooms with Opt. 1b	116	836	35+1,100	4.97 seconds	72.9%
Mushrooms without Opt.	4,194,302	416,073,565	35+1100	42:27 hours	72.9%
40 attr dataset without Opt.	$> 10^{12}$	-	-	$\approx$ 4.865 years	-

Table 3: How much enabling the **heuristic optimization** (1a or 1b) improves the efficiency of the algorithm when there is at least a case of an *unclassifiable* instance. *Combinations* defines the amount of combinations of attributes computed by the algorithm and *selectors* the quantity of different selectors explored. *Rules* are given in format $A+B$ where A is the amount of rules generated during the *combinatorial exploration*, and B, the *specificRules* generated for those instances that could not been classified during it (*Line 25* on code 2). In cases of optimization 1b and without it, rules generated are (as expected) the **same** for both. The precision of the rules generated has been taken from the training dataset, in order to observe how the learning qualities of the original algorithm are maintained or not in one or another optimization. *Rulesets* generated can be reviewed at the attached *MushroomGeneratedRules.txt* attached files.

As has been shown, ensuring the algorithm is not *wasting resources* searching for new rules that is known that **can't be found** during the combinatorial search could, heuristically, make the difference between obtaining the results in an **acceptable time** or never obtaining it in 48 consecutive human lives.

Between proposed optimizations 1a and 1b, erasing the data from the searching space (1a) can lead to worsen the final results, given that simpler rules can be taken as valid by the lower search space. So we can conclude that optimization 1b, where results are the same than without it but can heuristically being found in reasonable time, should be **always activated**.

2.1.1 A brief comment about *Mushroom Dataset modified*

Given his more difficult readability, this modification of the *Mushroom dataset* will be not deeply analyzed in following sections. However, given the **special results** that it draws, it is interesting to make a brief analysis.

This is the unique one among the analyzed datasets which create a **non 100% precise rules**. It is because, due the new objective variable, is the unique analyzed dataset containing a lot of *unclassifiable* instances. This is the main reason why, in spite of having the advantage of being analyzed over the ***training dataset***, its results are the worst ones with a 72.9% of *accuracy*.

Rules generated by it can be seen at the *MushroomGeneratedRules.txt* attached files. As seen there, for example in case 1b, only the first 35 rules have a 100% precision while the 1,100 following ones are generated about *unclassifiable* instances having a **lower precision** (but never lower than $\frac{1}{classes}$). This extremely large amount of imprecise rules is the responsible of the low accuracy results obtained by the algorithm (table 3).

3 Results

In this section we will **analyze** the result of the implemented algorithm over **3 different datasets**. In order to define this section, there is important to clarify how the prediction will work. Let's imagine the following case:

Rules				
IF Weather is Sunny **THEN** Season is Summer				
IF Trees are Yellow **AND** Temperature is Low **THEN** Season is Autumm				
Instances				
Case	**Weather**	**Trees**	**Temperature**	**Season**
1	Sunny	Yellow	Low	Autumm

Table 4: Example of rules conflict

In case shown at table 4, there is a rule conflict. Instance 1 satisfies **both proposed rules** and both have a different output. For this cases, a **priority rule** has been implemented, in order to decide if simpler rules prevails over the complex ones or not. Following the **original proposal** of the algorithm this rule will be always configured as **simpler ones prevails**, and in fact, it has been also proven empirically during the testing phase of the implementation that is the best configuration.

All analysis will be analyzed by ***k-fold*** **methodology**, with a previous **shuffling** of the dataset in order to avoid implicit pre-orders. In order to train as most accurate as possible, all rows containing **missing values** will be **erased** from the training set but not from the validation set of its fold.

For the sake of simplicity, original datasets has been modified, including a row with the **names of columns** at first line of the *CSV*. This modification allows to inspect in a more understandable way the rules generated without making any modification at code.

The 3 used datasets, has been selected due his human understandability and all of them have not *unclassifiable* instances. It implies that they all will generate **100% precise** rules in opposition of the example shown before at subsection 2.1.

3.1 Short Dataset - Votes

Congressional Voting Records Dataset includes the votes of each one of the **U.S. House of Representatives Congressmen** on the **16 key votes** identified by the ***CQA***, and its objective variable records if they are from the **democrat or republican** party.

This dataset is the shortest one analyzed including a total of **435 instances** with **16 binary attributes**. The fact of the attributes are binary compensates the high dimensionality of the dataset in terms of required **computational resources**.

Classes Distribution:

- **democrat** $\rightarrow$ 61.38%
- **republican** $\rightarrow$ 38.62%

3.1.1 Set of rules, coverage and precision

In order to allow an easier analysis, rules presented here are the ones generated with the **shortest** training set used during cross-validation. This selection criterion will also be applied at the **rest of analysis**, because shorter training sets will be more tending to generate **shorter *rulesets***.

In this case these are the rules generated at the last fold of the *2-fold* cross-validation:

RULE 0. Coverage: 0.48408. Precision: 1.

IF physician-fee-freeze $\rightarrow$ n THEN class $\longrightarrow$ democrat

RULE 1. Coverage: 0.02548. Precision: 1.

IF el-salvador-aid $\rightarrow$ n THEN class $\longrightarrow$ republican

RULE 2. Coverage: 0.03822. Precision: 1.

IF religious-groups-in-schools $\rightarrow$ n THEN class $\longrightarrow$ republican

RULE 3. Coverage: 0.10191. Precision: 1.

IF anti-satellite-test-ban $\rightarrow$ y THEN class $\longrightarrow$ republican

RULE 4. Coverage: 0.01274. Precision: 1.

IF mx-missile $\rightarrow$ y THEN class $\longrightarrow$ republican

RULE 5. Coverage: 0.26115. Precision: 1.

IF synfuels-corporation-cutback $\rightarrow$ n THEN class $\longrightarrow$ republican

RULE 6. Coverage: 0.01274. Precision: 1.

IF crime $\rightarrow$ n THEN class $\longrightarrow$ republican

RULE 7. Coverage: 0.03185. Precision: 1.

IF export-administration-act-south-africa $\rightarrow$ y THEN class $\longrightarrow$ republican

RULE 8. Coverage: 0.00637. Precision: 1.

IF handicapped-infants $\rightarrow$ n AND water-project-cost-sharing $\rightarrow$ n THEN class $\longrightarrow$ democrat

RULE 9. Coverage: 0.00637. Precision: 1.

IF handicapped-infants $\rightarrow$ y AND water-project-cost-sharing $\rightarrow$ n THEN class $\longrightarrow$ republican

RULE 10. Coverage: 0.00637. Precision: 1.

IF handicapped-infants $\rightarrow$ n AND adoption-of-the-budget-resolution $\rightarrow$ y THEN class $\longrightarrow$ democrat

RULE 11. Coverage: 0.00637. Precision: 1.

IF handicapped-infants $\rightarrow$ n AND immigration $\rightarrow$ n THEN class $\longrightarrow$ democrat

RULE 12. Coverage: 0.00637. Precision: 1.

IF handicapped-infants $\rightarrow$ n AND immigration $\rightarrow$ y THEN class $\longrightarrow$ republican

3.1.2 Interpretation of rules

Rules generated have a **pretty good description** of the dataset nature. First, it is important to note that most complex rules generated are composed by only **2 of the 16 attributes**. Moreover, 7 of them are composed by only **one attribute**.

When looking over the rules with **more coverage**, the conclusions are that RULE 0 and RULE 5 are the **most important** ones. Rule 0 have a 48.4% of coverage over the dataset, so noting that his consequent (*democrat*) covers a 61.38% of the dataset, there are **extremely high** possibilities of being *democrat* if the politician voted *No* to *Physician Fee Freeze*, concretely a $\frac{48.41\%}{61.38\%} = 78,88\%$. Also, if we look for example at Rule 5 we found that it have a 26.12% of coverage, while his class covers a 38.62% of the dataset, so we can stimate that there are a $\frac{26.12\%}{38.62\%} = 67,62\%$ of possibilities of being *republican* if he or she voted *No* to *Synfuels Corporation Cutback*. This kind of analysis is **so important** when we try to extract some **keypoints** about the analysis of the resulting ruleset.

In terms of the algorithm all precision values are 1., as is expected when *Rules Algorithm* runs through a dataset where there are no unclassifiable instances. Finally, is important to note that coverage of rules composed by **one condition** are always considerably higher than rules composed by **more conditions**, denoting that it is easy to sense which is the **ideology of the politician** by his or her response for determinate questions.

3.1.3 Accuracy results

Table 5 shows the different **accuracy** and **not prediction** measures taken from each k-fold cross-validation analysis done. This kind of analysis aims to deduce if reductions of the *training dataset* would seriously **damage the performance** of the method or not.

Votes Dataset		
K-fold	**Accuracy**	**Not Predicted**
2	91.0%	0.7%
10	93.8%	2.8%
50	92.8%	3.0%
100	92.9%	3.0%
Leave One Out	93.3%	2.8%

Table 5: *Accuracy* and *Not Predicted Ratio* for the different K-Fold validations performed over the *Votes Dataset*

As seen, results obtained at this dataset has been **pretty good**, obtaining in the most reliable case (Leave One Out) a **93.3% of accuracy** and only a **2.8% of not predicted instances ratio**.

The most difficult case, *2-fold*, the prediction has continued being pretty good with a 91.0% of Accuracy and a 0.7% of not predicted instances ratio.

These results leads to consider that the dataset was **easy to predict** due its nature.

3.2 Medium Dataset - Cars

Car Evaluation Dataset is one of the **most commonly used** when testing classification algorithms, especially for constructive induction and structure discovery methods. It is because it was derived from a simple **hierarchical decision model** originally developed for the demonstration of *DEX* [2], where some parts of the structure have been hide.

It have a total of **1,728 instances**. The purpose of the dataset is to predict how much a car is **acceptable or not** in function of **6 different properties**: *buying price*, *maintaining price*, *doors*, *persons capacity*, *luggage boot size* and *safety.*

Classes distribution:

- **unacceptable** → 70.02%
- **acceptable** → 22.22%
- **good** → 4%
- **very good** → 3.76%

3.2.1 Set of rules, coverage and precision

In order to **not overwhelm** the reader, only a summary of the rules generated will be included here. The complete version of the rules can be found at the ***CarsRules.txt*** file attached to this document.

RULE 0. Coverage: 0.33681. Precision: 1.

IF persons → 2 THEN class ⟶ unacc

RULE 1. Coverage: 0.21296. Precision: 1.

IF safety → low THEN class ⟶ unacc

RULE 2. Coverage: 0.03472. Precision: 1.

IF buying → high AND maint → vhigh THEN class ⟶ unacc

RULE 3. Coverage: 0.02199. Precision: 1.

IF buying → vhigh AND maint → high THEN class ⟶ unacc

RULE 4. Coverage: 0.02546. Precision: 1.

IF buying → vhigh AND maint → vhigh THEN class ⟶ unacc

RULE 5. Coverage: 0.0162. Precision: 1.

IF buying → vhigh AND doors → 5more THEN class ⟶ acc

RULE 6. Coverage: 0.02546. Precision: 1.

IF buying → high AND lug-boot → big THEN class ⟶ acc

RULE 7. Coverage: 0.01273. Precision: 1.

IF buying → vhigh AND lug-boot → big THEN class ⟶ acc

RULE 8. Coverage: 0.01157. Precision: 1.

IF buying → vhigh AND safety → high THEN class ⟶ acc

...

RULE 104. Coverage: 0.00116. Precision: 1.

IF buying → low AND maint → med AND doors → 4 AND persons → 4 THEN class ⟶ good

RULE 105. Coverage: 0.00116. Precision: 1.

IF buying → med AND maint → low AND doors → 2 AND persons → 4 THEN class ⟶ good

RULE 106. Coverage: 0.00116. Precision: 1.

IF buying $\rightarrow$ med AND maint $\rightarrow$ med AND doors $\rightarrow$ 2 AND persons $\rightarrow$ 4 THEN class $\longrightarrow$ acc

RULE 107. Coverage: 0.00116. Precision: 1.

IF buying $\rightarrow$ med AND maint $\rightarrow$ med AND doors $\rightarrow$ 4 AND persons $\rightarrow$ 4 THEN class $\longrightarrow$ acc

3.2.2 Interpretation of rules

As could been seen here, now, the amount of rules needed is **considerably higher** than before (8.9 times higher). Now the problem looks more complex to solve. It shows only **2 rules** which have only one attribute in account, but these 2 have an **extremely high coverage**. Cars with low safety or prepared for only 2 persons are never a good business. First of these rules coverage the 33.68% of the dataset, while the second one covers the 21.29% of it. It seems to prove that most acceptable cars are the ones with **bigger space inside** and **higher buying prices**.

It is also important to note that last five rules (*102 to 107*) are composed by 4 of the 6 attributes, and this is the **higher complexity** that can we found into the ruleset. In the most of cases (*Rules 10 to 101*) only **3 attributes are needed** (half of the total number of attributes). The order of importance how each rule is presented in the ruleset is: *Buying price*, *maintaining price*, *quantity of doors*, *safety*, *luggage bot* and *number of persons*. It is expected, given that prices of the cars usually have a high correlation with its quality. In the middle terms, *rules* usually classify as 'good' those cars with: low or medium prices, low maintaining prices and enough space inside.

3.2.3 Accuracy results

Table 6 shows the different **accuracy** and **not prediction ratio** measures for different *k-fold* cross-validation analysis performed. Aiming, one more time, to deduce whether or not reductions of the training dataset would seriously **damage the performance** of the method.

Cars Dataset		
K-fold	**Accuracy**	**Not Predicted**
2	89.1%	0.4%
10	95.2%	0.2%
50	95.0%	0.1%
100	94.9%	0.0%
Leave One Out	95.3%	0.0%

Table 6: *Accuracy* and *Not Predicted Ratio* for the different *k-fold* validations over the *Cars Dataset*.

As can be seen here, results are even **better** than the ones obtained by the last dataset. In the **Leave One Out** validation method was able to **predict all instances** (0.0% of *Not Predicted Rate*), giving a pretty high value of **95.3% of accuracy**. However, in the case where training sets were the shortest ones (*2-fold*) there was a more notable **decrement of the performance**, but even so arriving at **89.1% of accuracy**.

In conclusion, it was possible to see here how the high **increment of the generated ruleset** has led to an **increment of the performance**.

3.3 Large Dataset - Nursery

As his **official description** relates: *Nursery Database* is derived from a **hierarchical decision model** developed to **rank applications** for nursery schools. It was used during 1980's to give explanations about the rejected applications. Final decision of the model depended on three sub-problems: *occupation of parents and child's nursery*, *family structure and financial standing*, and *social and health picture of the family*.

It have a total of **12,960 instances** with **8 attributes** and **between 2 and 4 possible values** for each attributes. It is the **largest dataset** analyzed.

Classes distribution:

- **not recommended** → 33.33%
- **recommended** → 0.02%
- **very recommended** → 2.53%
- **priority** → 32.91%
- **special priority** → 31.20%

3.3.1 Set of rules, coverage and precision

In order to **not overwhelm** the reader, only a **summary** of the rules generated is included here. The complete version of the rules can be found at the ***NurseryRules.txt*** file attached to this document.

RULE 0. Coverage: 0.33364. Precision: 1.

IF health → not-recom THEN class ⟶ not-recom

RULE 1. Coverage: 0.01528. Precision: 1.

IF parents → usual AND has-nurs → improper AND housing → critical THEN class ⟶ priority

RULE 2. Coverage: 0.01373. Precision: 1.

IF parents → great-pret AND has-nurs → improper AND social → problematic THEN class ⟶ spec-prior

RULE 3. Coverage: 0.01343. Precision: 1.

IF parents → pretentious AND has-nurs → less-proper AND social → problematic THEN class ⟶ priority

RULE 4. Coverage: 0.01512. Precision: 1.

IF parents → pretentious AND has-nurs → proper AND social → problematic THEN class ⟶ priority

RULE 5. Coverage: 0.0108. Precision: 1.

IF parents → usual AND has-nurs → improper AND social → problematic THEN class ⟶ priority

RULE 6. Coverage: 0.01373. Precision: 1.

IF parents → usual AND has-nurs → less-proper AND social → problematic THEN class ⟶ priority

RULE 7. Coverage: 0.01343. Precision: 1.

IF parents → usual AND has-nurs → proper AND social → problematic THEN class ⟶ priority

RULE 8. Coverage: 0.01466. Precision: 1.

IF parents → usual AND has-nurs → very-crit AND social → problematic THEN class ⟶ spec-prior

...

RULE 409. Coverage: 0.00031. Precision: 1.

IF parents → pretentious AND form → incomplete AND children → 2 AND housing → less-conv THEN class ⟶ priority

RULE 410. Coverage: 0.00015. Precision: 1.

IF parents → usual AND form → complete AND children → 2 AND housing → convenient THEN class ⟶ very-recom

RULE 411. Coverage: 0.00015. Precision: 1.

IF parents → usual AND form → completed AND children → 2 AND housing → convenient THEN class ⟶ very-recom

RULE 412. Coverage: 0.00031. Precision: 1.

IF parents → usual AND form → incomplete AND children → 2 AND housing → convenient THEN class ⟶ priority

3.3.2 Interpretation of rules

In this case, the ruleset generated is considerable higher than in last cases with a total of **412 rules**. On following section will be analyzed if it also entails an increment over the performance metrics.

Now, there is only a rule with a condition formed by one attribute, which says that is **not recommended** to accept nursery students with **not recommendable** health. However, this rule having a 33.33% of coverage denotes that, not only it was always a reason of direct rejection (Probabilities are $\frac{33.33\%}{33.33\%} = 100\%$), so it was the **unique reason** for direct rejections present on the dataset. This kind of **conclusions** that *rule based* algorithms (as well as tree classifiers) allows to mine from the datasets is one of the **most important strengths** of them.

It is also important to see that there are no rules formed by two attributes, which can lighten about the complexity of the dataset. Most complex rules, however, are formed by only 4 of the 8 attributes, and in fact more than 350 of the 412 rules generated **uses the attribute 'parents'** which denotes that it was the most important factor to have in account when accepting or rejecting an application.

3.3.3 Accuracy results

Table 7 shows, one more time, the different **Accuracy** and **Not Prediction Rate** measures for different *k-fold* analysis performed. For deducing if reductions of the training dataset would now have higher **damage over the performance** of the method than before.

Nursery Dataset		
K-fold	**Accuracy**	**Not Predicted**
2	98.0%	0.0%
10	99.3%	0.0%
50	99.5%	0.0%
100	99.5%	0.0%
500	99.5%	0.0%

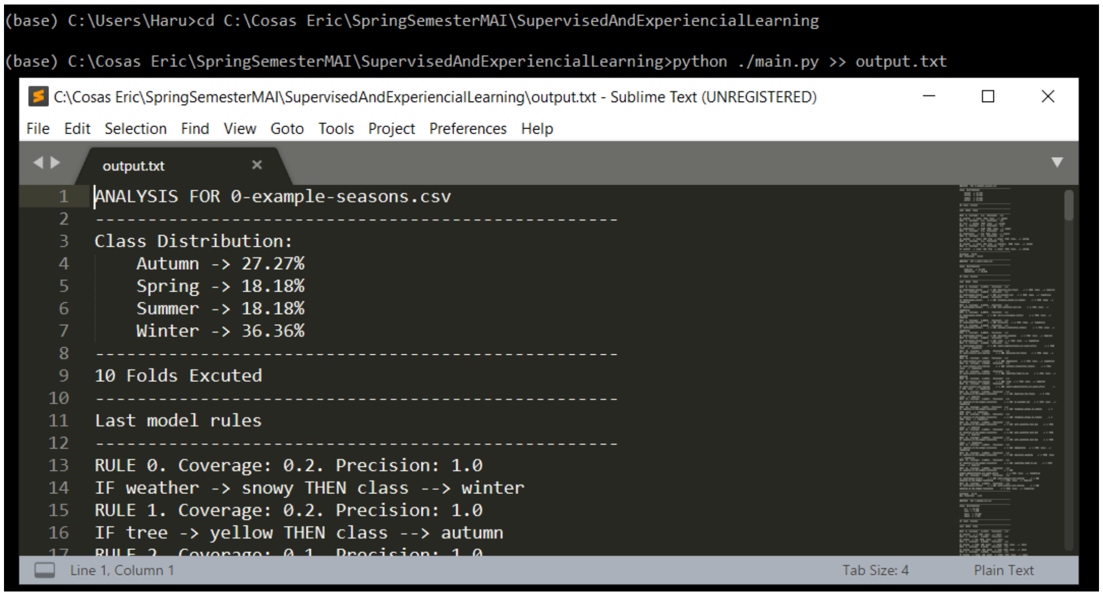
Table 7: *Accuracy* and *Not Predicted Ratio* for the different *k-fold* validations performed over the *Nursery Dataset*

As seen here, one more time, the large **increment in the generated ruleset** has entailed also an **increment in the performance** of the algorithm. From cases of *50-fold* and on, it has arrived until **99.5% of *Accuracy***, **predicting always all the instances** tested (0.0% of *Not Predicted Rate*). Even in the case where only the half of the dataset was used for training, *2-fold*, it has arrived until 98% of accuracy, which is a tremendously **great result**.

4 How to execute the code

Project has been developed using Python, and in order to execute it you must have a ***Python 3*** version installed at your computer as well as ***Numpy*** library.

It is executable through the main file, executing *python ./main.py >> output.txt &* on your command prompt **from the project folder**, as figure 1 shows.



```
(base) C:\Users\Haru>cd C:\Cosas Eric\SpringSemesterMAI\SupervisedAndExperiencialLearning

(base) C:\Cosas Eric\SpringSemesterMAI\SupervisedAndExperiencialLearning>python ./main.py >> output.txt
```

C:\Cosas Eric\SpringSemesterMAI\SupervisedAndExperiencialLearning\output.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
output.txt

```
ANALYSIS FOR 0-example-seasons.csv
--------------------------------------------------
Class Distribution:
    Autumn -> 27.27%
    Spring -> 18.18%
    Summer -> 18.18%
    Winter -> 36.36%
--------------------------------------------------
10 Folds Excuted
--------------------------------------------------
Last model rules
--------------------------------------------------
RULE 0. Coverage: 0.2. Precision: 1.0
IF weather -> snowy THEN class --> winter
RULE 1. Coverage: 0.2. Precision: 1.0
IF tree -> yellow THEN class --> autumn
```

Figure 1: Example of code execution and its expected output.

4.1 Code Structure

Project is divided in **two sections**:

- **Data folder:** It contains all the datasets that we want to analyze, with *CSV* format. In order to make new visualizations simpler **without modifying** any line of code, those *CSV* should contain at first row the **names** of each column, being the **objective attribute** named "*class*". All of them will be executed in **alphabetical order** when main is executed. By default it contains **5 datasets**:
  - **0-example-seasons.csv:** It is the toy dataset used as example on the **original paper**. Can be used as a **test** for compare with the expected results.
  - **1-short-votes.csv, 2-medium-cars.csv and 3-large-nursery.csv:** The **analyzed datasets** presented at sections 3.1, 3.2 and 3.3
  - **4-special-case-mushroom.csv:** *Mushroom dataset* for the analysis about how algorithm works when there are **inconsistent instances**, done at section 2.1.
- **Code:** is divided in the following 4 files:

- **Rules.py:** It is the **core** file of the project. Contains a class named *RULES* which implements the algorithm. It is implemented following the *scikit-learn* standards, with a **fit and predict** as its principal functions. This class is also in charge of **showing the ruleset** in a human readable way and **getting some statistics** about the fitting such as the amount of *combinations* or *selectors* explored.
- **Reader.py:** It only contains an **iterable function**. It is in charge of reading *CSV* file, **dividing** it dynamically in k training/validation folds, and returning at each step one of these **folds**. It also analyzes the **class distribution** over the read dataset.
- **Analyze.py:** It executes the **complete analysis** of a dataset, printing as a result its class distribution, its generated rules and its results of **Validation Accuracy** and *Non Predicted* instances *Rate*.
- **main.py:** It executes the analysis for **all the datasets** in the Data folder, and is the one which must be executed when running the project.

5 Conclusions

Through this work, *RULES* algorithm has been **implemented** and **analyzed**.

Implementation part has drew the following results:

- *RULES* algorithm is defined as a simple *inductive learning* algorithm for extracting *IF-THEN* rules from a training set. However, this simplicity could easily lead to a **combinatorial explosion** making the algorithm never ends as was seen at table 3. It is important to have it in account when selecting dataset and add some **stop conditions** as optimization 1b for heuristically **avoiding** it.
- The most **damaging factor** in computational time for *RULES* algorithm are then: **Possible Values of each Attribute** and **Amount of Attributes**. In cases where it turns unmanageable it could be important to make some **preprocessing** of the data trying to reduce this 2 factors. Given that nature of the algorithm, **most interesting** preprocessings could be:
  - **Data Inconsistency Solving:** Removing from those instances which presents a different class label for the same combination of attributes (table 2) the ones which have the minority class label. Optimization 1b implemented avoids this risk, but in original algorithm case where it is not implemented, removing those instances could have the **same impact** in the performance.
  - **Feature Selection:** Removing those features with a low relevance could decrease significantly the **combinatorial explosion risk**.

However, the most relevant apprenticeships comes from the **analysis part**:

- **Rule based algorithms** are able to obtain not only high accuracy metrics in **categorical dataset** classifications, but they also draw a nice **human understandable description** of the prediction process, using the *ruleset*. As in case of *tree classification algorithms*, it is one of the most valuable characteristics of the method, given that it allows a posterior **easy human analysis** of the dataset (in opposition with the majority of machine learning algorithms).
- When analyzing the results, it is important to see accurately at **coverage** and **precision** metrics:
  - **Coverage:** While we are extracting conclusions from the generated *ruleset*, it is important to look first at higher coverage rules. As **close** a rule coverage is from his consequent occurrence at the dataset, **most important** is the rule for defining the data.

    For example, if we look at first rule generated for the *Nursery* dataset (section 3.3) which same percentage as *not recommended class*, we can discover that it was the **unique** factor of direct rejections. Another example could be found at first rule of *Votes* dataset (section 3.1) which practically covers the 80% of *democrats* ($\frac{48\% of coverage}{61\% of democrats} = 78\%$). This kind of **knowledge extraction** from the rules coverage could be one of the most useful when summarizing **keypoints**.

  – **Precision: Precision metrics** can determine how **consistent** is a rule. In the case of the 3 principal datasets analyzed, it is always 1., given that they have not inconsistent instances. However, this is not the same case for, in example, the modified *Mushrooms Dataset* tried (section 2.1), which had this kind of inconsistent instances. When they appear, it is useful to have in account the precision metric of a rule in order to measure its **reliability**.

- Usually, as larger is the *ruleset* generated, more able is the method to make more complex predictions. In tests done, it was reflected as an increment of the **Validation Accuracy** metric as longer was the *ruleset* generated. However, as was shown in section 2.1.1 this is more related with the **complexity** and **consistency** of the dataset than with the generated *ruleset*. Moreover, more important than the total amount of rules generated, is the **amount of 'simple' rules**, which are the ones which are more able to **generalize**.

- Looking at *k-fold Accuracy* tables, (tables 5, 6 and 7), there are performance implications involving the amount of training data used, but they are **not really significant** in most of cases (differences between 10-fold (90/10 partition) and leave one out, was always **not significant**).

References

[1] D. T. Pham and M. S. Aksoy, RULES: A Simple Rule Extraction System. At Expert Systems With Applications, Vol. 8, No. 1, pp. 59-65, 1995.

[2] M. Bohanec and V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.)