UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Practical Work 1:
Rule-Based Classifier RULES

Supervised and Experiential Learning

João Valério

joao.agostinho@estudiantat.upc.edu

31/03/2023

Index

1. INTRODUCTION

The main goal of this work is to implement, validate and analyze the RULES algorithm, a rule-based classifier, proposed by D. T. Pham and M. S. Aksoy in the paper *RULES: A Simple Rule Extraction System*.

Following this perspective, the descriptions presented in [1] are considered the base for the algorithm's development and analyses. Its implementation is performed in Python's programming language, through which it is expected to apply a set of classification rules to a validation/test dataset and obtain the global classification accuracy of the model.

Thus, from the UC Irvine Machine Learning[1] and DataHub[2] repositories, the three datasets with the specifications required were selected. To amplify the scope of study of the present algorithm, it was intended to choose diverse datasets. Therefore, these are Hepatitis, Tic-Tac-Toe Endgame, both from 1, and Phoneme, from 2, which have, accordingly, the following specifications: small dataset with numerical and, predominantly, categorical data, the medium dataset only with categorical data and the large dataset only with numerical data. Precisely, besides the size restrictions, it was also considered characteristics in the selection process, such as the deviation of the class distribution, the percentage of instances belonging to the majority and minority classes, the percentage of missing values, the number of numerical and categorical features and, finally, the number of classes.

Additionally, the three mentioned datasets are properly preprocessed in terms of missing values, normalization on numerical data and discretization, by the order exposed. Specifically, the last step is an inherent requirement of RULES, which only executes rule classification on qualitative data. Moreover, the data is split into the train, 80%, where the set of rules is obtained, and the test, 20%, where the algorithm is evaluated.

Besides the overall accuracy of the model, it is also obtained the number of rules, the train/test time and memory usage. Regarding each rule, it is acquired the coverage, recall and precision. Through these metrics, it is possible to reach more complete and grounded conclusions concerning the implementation.

Finally, the instructions for the code execution are provided, as well as complementary material sources, such that the replication of the current paper is conceivable.

2. DATA

a. Characteristics

According to both repositories, the datasets selected and the respective characteristics are:

- **Hepatitis**: small dataset with numerical and, predominantly, categorical data.
- **Tic-Tac-Toe Endgame**: medium dataset only with categorical data.
- **Phoneme**: large dataset only with numerical data.

[1] The datasets Hepatitis and Tic-Tac-Toe Endgame acquired from this repository can be found in the following link: https://archive-beta.ics.uci.edu/.
[2] The dataset Phoneme acquired from this repository can be found in the following link: https://datahub.io/.

Furthermore, besides the size restrictions, it was intended to obtain diversity between the data sets on the number of features and their properties. The exception is the number of classes, since all of the datasets concern binary classification problems, due to computational resource availability. Finally, the imbalance between class representations pretends to test the generalization capacity of the RULES algorithm.

The attributes mentioned in each dataset are presented in table 2.a.1.

Table 2.a.1 – Characteristics of the datasets.

	Characteristics of the Datasets							
	Number of Instances	**Numerical Features**	**Nominal Features**	**Number of Classes**	**Deviation of Class Distribution**	**Majority Class**	**Minority Class**	**Missing Values**
Hepatitis	155	6	13	2	29.35%	79.35%	20.65%	6.01% (42.23%*)
Tic-Tac-Toe Endgame	958	-	9	2	16.65%	66.34%	34.66%	-
Phoneme	5404	5	-	2	20.60%	70.65%	29.35%	-

* The value in parenthesis indicates that the feature with the highest rate of missing values has a portion of 43 % missing values. This is considered to make a large impact on the result, especially since the data is relatively small.

b. Preprocessing

In order to obtain a proper dataset to feed a supervised machine learning algorithm, it is important to preprocess the data. This domain is separated into five central categories: Data Upload (i.), Missing Values (ii.), Different Ranges (iii.), Discretization (iv.) and Data Splitting (v.), in which the sequential order referred to agrees with the respective code.

i. Data Upload

Regarding the data upload, the preprocessing code is able to read either .csv or .arff files, for code generalisation purposes. Thus, in the folder ‘Data’, Hepatitis is a .arff file, while the remaining are .csv ones.

ii. Missing Values

In the first subject, it is crucial to handle the problem of missing data, where two approaches are possible: deletion or imputation. The only dataset with missing values is Hepatitis, with a rate value of 6.01%. Since the number of cases is only 155, the deletion of observations results in a considerable loss of information. Thus, it is chosen to implement imputation. Through that, it is expected to keep the information regarding the observations without inserting possible bias into the data.

Numerical Data:

For the numerical data, the considered metric is the K-Nearest Neighbours Imputer, in which each sample's missing values are imputed according to the mean of the k-nearest neighbours considered. As the function considered from the sckit-learn library is optimised to the general cases of numerical imputation, the best parameters among the tested are the default ones. According to that, k assumes a value of 5, with uniform weight distribution between the neighbours.

Categorical Data:

In the case of categorical features, the missing nominal value, '?', is considered as a new feature named 'Unknown', since a wrong categorical attribution could result in poor clustering, due to the transversal effect of the missing values.

iii. Different Ranges

Then, the normalization of all the numerical data is executed. Since different features have distinct numerical ranges, the weights between them are nonuniformly distributed, inserting biased information in the model. As there is no relevant information pointing out that certain features should have more weight than others, it is implemented a uniform weight distribution along the attributes.

The method considered is Min-Max Scaling, in which each instance has a linear value attribution between 0 (minimum) and 1 (maximum).

iv. Discretization

As the algorithm implemented, RULES, assumes that the attributes are qualitative, it is necessary to perform, as a last step, discretization on the dataset. In that perspective, it is implemented the K-means discretization transform, from the sckit-learn library, in which the output consists of 10 bins encoded as an integer value (ordinal) through a quantile strategy.

v. Data Splitting

Finally, the dataset is split into training and testing, where the set of rules are conceived and assessed, respectively. Thus, the training data is constituted 80% of the original dataset, while the testing represents the remaining 20%.

3. ALGORITHM

The RULES algorithm, a rule-based classifier, is a machine learning technique that works by identifying patterns and rules within a dataset to create predictions on new data. It is a type of supervised learning algorithm that uses a set of predefined rules to classify input data into specific categories. In essence, the RULES algorithm involves a set of IF-THEN rules that define decision boundaries for classifying data points. These rules are created by analyzing the input data and identifying patterns that are common across different categories. Once the rules are established, the algorithm can quickly classify new data based on the rules it has learned. One of the key advantages of RULES is its interpretability, which allows users to understand why certain decisions are being made by examining the rules that have been defined.

However, the effectiveness of the RULES algorithm depends heavily on the quality of the rules that are formed. If the rules are too specific, the algorithm may not generalize well to new data, while if they are too general, the algorithm may not be accurate enough. As a result, the RULES algorithm requires careful tuning and experimentation to achieve optimal results.

In order to develop the algorithm, the descriptions presented in [1] are considered as the base of its development. Therefore, its correspondent pseudocode for rule acquisition, with the correct order of execution, can be described as follows:

Input: df: pandas.DataFrame object, input data frame to train the model.

Output: List of discovered association rules with their corresponding performance measures.

1.1. single_ruler(df):

- Discover single antecedent rules.
- Iterate over all features except the last (which is assumed to be the target class).
  - Iterate over all unique values for the feature.
    - Extract the subset of the DataFrame where the feature has the given value.
    - Check if all instances in the subset have the same class value.
      - If the subset satisfies the rule, create a new rule.
        - Create a tuple for the antecedent and consequent of the rule.
        - Create a boolean mask for the instances that satisfy the antecedent.
        - Count the number of instances that satisfy the antecedent and consequent and only the antecedent.
        - Calculate the precision, recall, and coverage of the rule.
        - Add precision, recall, and coverage to the corresponding lists.
        - Add the rule to the list of rules.
        - Update the unclassified DataFrame with the new rules.
    - End if.
  - End for.
- Return the updated unclassified DataFrame.

1.2. non_single_ruler(df, unclassified):

- Discover rules with more than one antecedent.
- Loop through different numbers of features to generate rules for.
  - Create all possible combinations of columns for this number of features.
  - Loop through each column combination and value combination and generate rules.
    - Create a subset of the input dataset based on the current column and value combinations.
    - Determine the most common class value in the subset.

- Check if all instances in the subset belong to the most common class.
- Generate a rule based on the column and value combinations and the most common class.
- Check if the rule is redundant.
  - If not, add it to the list of rules.
- End for.
- Return the updated unclassified DataFrame.

1.3. unclassified_ruler(df, unclassified):

- Discover rules for the unclassified instances.
- Loop through the unclassified instances.
  - Create a subset of the input dataset containing the current instance.
  - Call the non_single_ruler() method on this subset to generate a rule.
  - Check if the rule is redundant.
    - If not, add it to the list of rules.
  - End if.
- End for.

Moreover, it is essential to denote that the rules are 100% precise if there are no duplicated instances with different class labels.

Finally, the implementation is conceived in Python's programming language, through which it is expected to apply a set of classification rules to the validation/test datasets and obtain the global classification accuracy of each model.

4. ANALYSIS

a. Evaluation Metrics

Firstly, previously to the analysis of the results, it is necessary to clarify the evaluation metrics employed.

Regarding the overall algorithm performance, the metrics utilised, in the train and test phases, are the following:

- **Number of Rules**: indicates the number of rules conceived by the model in the learning process. These are the same in the train and test phases.
- **Accuracy [%]**: the main metric to evaluate the algorithm's performance, which indicates the number of correct predictions. Moreover, it is expected that in the training process, the algorithm must retain the capacity to reach 100% of accuracy. This means that, if rule pruning is not performed, the inherent capacity of the model to reach 100% accuracy must always be present, unless the data is ambiguous.
- **Time [s]**: pretends to clarify the time taken by the algorithm in the train and test stages. This helps to understand the influence of the dataset characteristics, such as the number

of instances, features and classes, in both processes. However, it might be expected that the algorithm will spend largely more time on the training phase.

- **Memory [kB]**: characterises the amount of memory used by the algorithm. Nonetheless, in the test phase, the quantity must be irrelevant (close or equal to null).

Furthermore, regarding the set of rules obtained from the pseudocode disclosed, the main metrics are:

- **Precision [%]**: the precision of a classification rule R is the ratio between the instances satisfying the antecedent and the consequent of R, and the instances satisfying the antecedent of R.
- **Coverage [%]**: the coverage of a classification rule R is the ratio between the instances satisfying the antecedent of R, and the total instances in the training dataset.
- **Recall [%]**: the recall of a classification rule R is the ratio between the instances satisfying the antecedent of R and the consequent of R, and the total instances in the training dataset belonging to the same class label that R is classifying.

According to the previous definitions, some behaviours are desired, even before conveying the results. Relatively to the precision, all the rules should have a value of 100%, since this is an inherent premise of the developed code unless the data contains ambiguity. Additionally, the coverage of the rules, usually, should be higher when the complexity of the rule is lower. For instance, a rule with one antecedent X = a must cover more examples than a rule with the antecedents X = a & Y = b. On the other hand, the recall must increase along with the rules' complexity, since the domain is restricted to the same class label that the rule is classifying.

b. Model's Results Analysis

Through the execution of the code provided on the datasets selected, the first set of metrics mentioned in point 4.a. were obtained. In that perspective, the analysis of the outcomes regarding each dataset will be conducted individually in (i.), Hepatitis, (ii.), Tic-Tac-Toe Endgame, and (iii.), Phoneme.

As a last note, the datasets provided do not distinguish between train and test data. Thus, to avoid biased algorithm analysis, the data is randomised each time the code is executed. Therefore, the results displayed in tables 4.b.i.1, 4.b.ii.1 and 4.b.iii.1 consist of an average of 100 executions and the rules exposed correspond to the best accuracy from these executions. Along these lines, it is feasible to comprehend reliably the average correctness of the model, as well as the maximum capacity for rule extraction.

i. Hepatitis

The metrics related to the overall algorithm in the Hepatitis dataset are presented in table 4.b.i.1.

Table 4.b.i.1 – Performance of the RULES in the Hepatitis dataset.

	Characteristics of the Datasets					
	Number of Instances	**Number of Features**	**Number of Rules**	**Accuracy [%]**	**Memory [kB]**	**Time [s]**
Train	124	19	1062	100.0	1664.0	1.989
Test	31	19	1062	80.0*	0.0	0.002

*Highest value = 90.3%; Lowest value = 67.8%

According to the values obtained, it is inferable that the model developed can capture the whole complexity of the training data through the 1062 rules provided. Thus, this indicates that RULES describes the general patterns inherent to the dataset, as well as specific details of the training data (overfitting). The latter is observable through the 20% discrepancy between both accuracies and the fact that the model uses 45% of all the possible rules[3] (2356), a considerable quantity. To avoid that, a lower level of the rule's complexity could be chosen, for instance, opting for more coverage, and generality, than a detailed data description.

Time and memory complexity is significant taking into account the number of instances and features of the dataset. This is due to the creation and storage of the rules conceived after all the possible attribute-value pairs are tested.

ii. Tic-Tac-Toe Endgame

In the Tic-Tac-Toe Endgame dataset the values obtained are in table 4.b.ii.1.

Table 4.b.ii.1 – Performance of the RULES in the Tic-Tac-Toe Endgame dataset.

	Characteristics of the Datasets					
	Number of Instances	**Number of Features**	**Number of Rules**	**Accuracy [%]**	**Memory [kB]**	**Time [s]**
Train	766	9	356	100.0	1600.0	2.213
Test	192	9	356	94.3*	0.0	0.024

*Highest value = 96.4%; Lowest value = 86.1%

On the medium dataset, the model can capture the general pattern of the dataset without significant overfitting, since the difference between training (100.0%) and testing (94.3%) is

[3] All the possible rules are obtained by multiplying the number of features by the number of instances in the training stage. For simplicity purposes, the value each pair instance-feature can assume is fixed.

only 5.7%. Moreover, the number of rules (356) represents only 5% of all the possible combinations (6894).

Comparing the previous results with the ones from i., it is inferable that the RULES model improves the dataset's pattern definition with an increment of instances and a decrement of features. Therefore, performing feature selection as a preprocessing step might be relevant when applying RULES since the algorithm is not able to distinguish between the most and least relevant features.

Additionally, the memory and time usage did not vary significantly from the results in table 4.b.i.1, even though the number of instances is 6 times larger. However, this is possible due to the decrement in the number of features. Thus, feature selection would also improve significantly these complexity metrics.

iii. Phoneme

Lastly, in the Phoneme dataset, the largest one, the following matters were reached.

Table 4.b.iii.1 – Performance of the RULES in the Phoneme dataset.

	Characteristics of the Datasets					
	Number of Instances	**Number of Features**	**Number of Rules**	**Accuracy [%]**	**Memory [kB]**	**Time [s]**
Train	4323	5	9852	95.9%	11408.0	50.423
Test	1081	5	9852	86.1*	0.0	6.723

*Highest value = 87.0%; Lowest value = 84.3%

In the larger dataset, even though the number of features is reduced, the number of instances is significant, taking into account the resources available. Therefore, the memory and time required by the model to perform the training and testing stages are substantial, which might indicate a drawback of RULES for large datasets.

Regarding accuracy, it is notable that the model was not able to completely describe the training data, as on the other datasets, even though it used 46% of all the possible rules (21615). In this execution, after considering all the potential pairs, some instances still lacked classification. Therefore, to include the unclassified instances, the number of rules incremented. Nonetheless, the accuracy of the training set decreased, since the added rules mirrored the inherent ambiguity of the dataset. Hence, these are imprecise (precision different from 100%), indicating that this dataset contains duplicated instances with different class labels.

c. Rule's Results Analysis

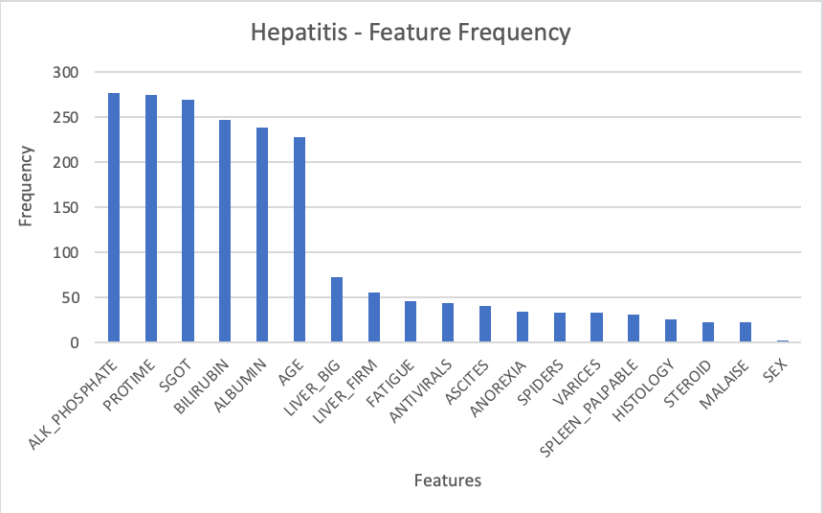
In the present chapter, the study is related to the rules obtained through the model execution. As stated previously, the principal metrics of evaluation are precision, coverage and recall, which the definitions are described in chapter 4.a.. However, since the rules.txt file provides access to the full set of rules, only the most relevant ones for the analysis will be referred to in the

following chapters (i.), (ii.) and (iii.). Moreover, the study also scrutinises the frequency of each feature and the complexity of the rules.

i. Hepatitis

The graphical representation on 4.c.i.1 illustrates that all the features were used during the rule composition. However, the frequencies between features have a considerable difference.

Graph 4.c.i.1 – Feature frequency on the rules for Hepatitis dataset.



According to the results, it is possible to infer that the acquired rules rely mainly on 6 central features (ALK_PHOSPHATE, PROTIME, SGOT, BILIRUBIN, ALBUMIN and AGE). In fact, in Hepatitis with 11 components, it is possible to achieve 90% of the variance, indicating that the main underlying information of the data is contained in a lower set of features [2]. Therefore, performing feature selection would decrease dramatically the complexity of the dataset, with the possibility of an increment in accuracy, as registered in [2] for other machine learning models.

Consequentially, as shown in table 4.c.i.1, 99% of the rules consider solely 2 features in their composition.

Table 4.c.i.1 – Rules' complexity for Hepatitis dataset.

	Number of Rules		
	1 Feature	**2 Features**	**Total**
Hepatitis	15	1047	1062

Thus, even though it might seem an insufficient complex premise, taking into account the number of features and instances, graph 4.c.i.1 and [2] provide sufficient descriptive information to validate these results.

Considering the rules designed, it is understandable that the data does not present any ambiguity, since the precision for each rule is 100%. In terms of coverage, the rule with the highest value has the following configuration:

IF SPIDERS = no & HISTOLOGY = no **THEN** Class = 1 **WITH** Coverage 41.94% & Recall 100.00% & Precision 100.00%

Thus, it is interesting to notice that the SPIDERS and HISTOLOGY features combined, with a 'no' value, can describe 41.94% of the instances in the dataset, demonstrating a strong correlation between both. Logically, in the rules set, neither one of the referred features appears as a single antecedent of a rule, otherwise one of those would surpass the mentioned coverage. Therefore, in the single rule set, the highest coverage and recall regard with respect to ALBUMIN.

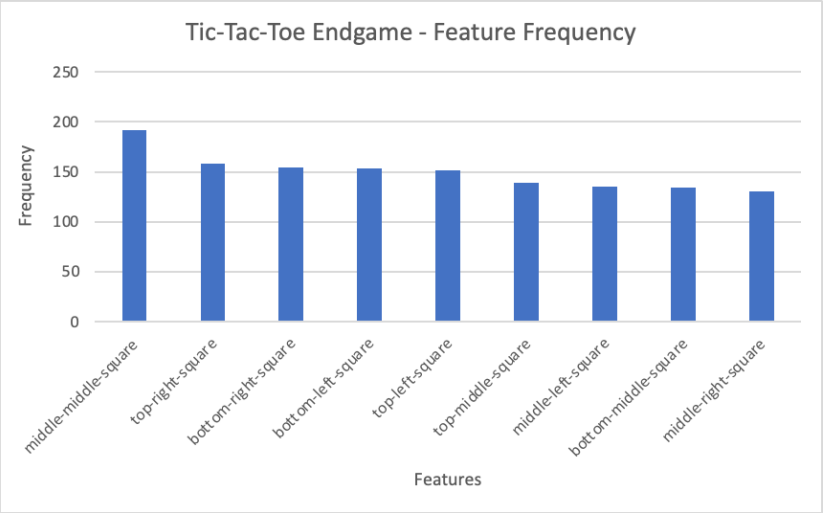
IF ALBUMIN = 8.0 **THEN** Class = 1 **WITH** Coverage 15.32% & Recall 19.19% & Precision 100.00%

Furthermore, when the rules have 2 features as a baseline decision, the recall achieved is always 100%, underlying that 2 features are enough complexity to describe the patterns within the same class. On the other hand, as shown in the latest rule, single antecedent rules never surpass 19.19% on recall.

ii. Tic-Tac-Toe Endgame

Regarding graph 4.c.ii.1, it is observable that the frequency is approximately uniformly distributed.

Graph 4.c.ii.1 – Feature frequency on the rules for Tic-Tac-Toe Endgame dataset.



Furthermore, the middle-middle-square feature contains the highest frequency, which agrees with the game mechanics. Since it is the central positional square of the game, it is known to be the most relevant to dominate it.

Finally, table 4.c.ii.1 indicates the complexity of the rules.

Table 4.c.ii.1 – Rules' complexity for Tic-Tac-Toe Endgame dataset.

	Number of Rules		
	3 Features	**4 Features**	**Total**
Tic-Tac-Toe Endgame	81	275	356

Through the results achieved, the decisions of the model rely only on 3 (23%) or 4 (77%) features combined. Taking into account the game mechanics and the accuracy achieved (94.3%), it is surprising how the model can reach such a good level, considering at most half of the board information.

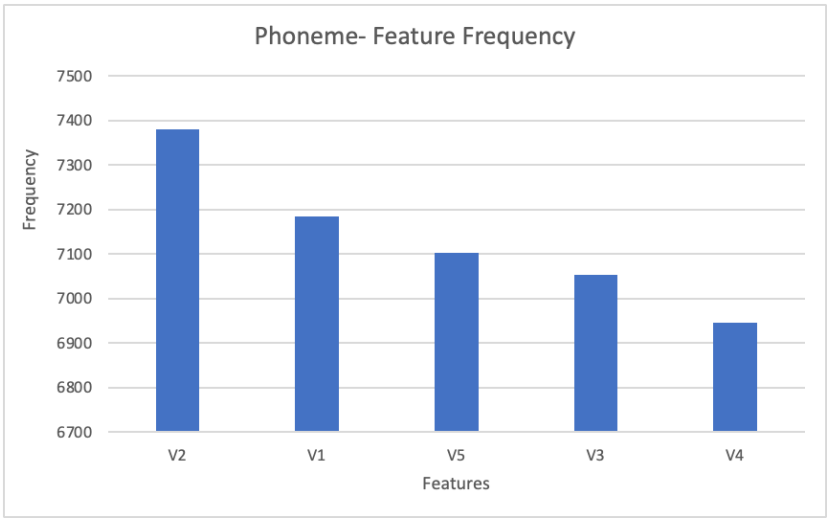
Relatively to ambiguity, the conclusion reached in 4.c.i is transversal to the Tic-Tac-Toe Endgame dataset. Then, the highest level of coverage, 9.53%, was achieved by the following rule:
IF top-left-square = x & middle-middle-square = x & bottom-right-square = x **THEN** Class = 1 **WITH** Coverage 9.53% & Recall 100.00% & Precision 100.00%

Which is on the set of least complex rules for this dataset. Moreover, it is notable to notice that the recall associated is 100%, as in all the other rules conceived. The presence of the middle-middle-square feature in the previous rule is in accordance with its frequency (graph 4.c.ii.1) and importance in the game Tic-Tac-Toe.

iii. Phoneme

Lastly, graph 4.c.iii.1 illustrates the feature frequency distribution of the rules.

Graph 4.c.iii.1 – Feature frequency on the rules for Phoneme dataset.



As can be observable, the discrepancy between features' frequency is relevant, indicating that, possibly, there is a significant difference in their importance. Moreover, the class imbalance, illustrated in table 2.a.1, might also be a plausible reason for the effect.

Despite the previous analysis, table 4.c.iii.1 indicates that, usually, the majority of the features are used in the rule's composition.

Table 4.c.iii.1 – Rules' complexity for Phoneme dataset.

	Number of Rules					
	1 Feature	**2 Features**	**3 Features**	**4 Features**	**5 Features**	**Total**
Phoneme	2	159	3096	5866	729	9852

Precisely, 98% of the rules use either 3, 4 or 5 features in the classification process. This originates a complex set of rules and a costly process in terms of memory and time, as noticed earlier.

Contrary to the remaining datasets, the Phoneme was exposed for holding some ambiguous data points, since the final unclassified instances resulted in a set of rules with a precision lower than 100%. Precisely, the rule created with the lowest precision (6.67%) is:

IF V1 = 3.0 & V2 = 5.0 & V3 = 9.0 & V4 = 0.0 & V5 = 1.0 **THEN** Class = 1.0 **WITH** Coverage = 0.023% & Recall = 0.01% & Precision = 6.67%

Which does not agree with the following rule:

IF V1 = 3.0 & V2 = 5.0 & V3 = 9.0 & V4 = 0.0 & V5 = 1.0 **THEN** Class = 0.0 **WITH** Coverage = 0.32% & Recall = 0.08% & Precision = 93.33%

Since, for the same antecedents, the classification class is different depending on the instances. As expected, the one with the most elevated precision contains, as well, higher coverage and recall.

Finally, the rule with the highest coverage belongs to the small set of the least complex rules.

IF V1 = 9.0 **THEN** Class = 0 **WITH** Coverage 10.18% & Recall 14.34% & Precision 100.00%

Due to the lack of complexity, the recall is not the highest in the sample, since the majority of rules reach 100%.

5. CODE

a. Organization

The code developed is organized into 3 main classes:

- **Preprocessing.py**: contains the PREPROCESS class, which consists of the preprocessing of the datasets characterised in chapter 2.
- **RULES_ALG.py**: contains the RULES class, which consists of the complete RULES algorithm implementation described on topic 3 of the paper.
- **main.py**: the main .py file, where the preprocessing, training and testing stages are executed.

The description of each function can be accessed through the .py files, as well as the detailed description of each step of the code.

b. Modules

The necessary modules for the code development and the respective versions are the following: **numpy** - version 1.24.2, **pandas** - version 1.5.3**, psutil** - version 5.8.0**, scikit_learn** - version 1.2.2 and **scipy** - version 1.10.1. The python version used is 3.8.8 through PyCharm CE software.

c. Execution

To execute the code through the terminal the following steps should be taken:

1. pip install numpy
2. pip install pandas
3. pip install psutil
4. pip install scikit_learn
5. pip install scipy
6. python3 /PATH_WHERE_THE_FILE_main.py_IS_INSERTED

   Ex: /Users/joaovalerio/Documents/"MAI UPC"/"2 Semester"/SEL/W1/source/main.py
7. /PATH_WHERE_THE_FOLDER_DATA_IS_INSERTED

   Ex: /Users/joaovalerio/Documents/MAI UPC/2 Semester/SEL/W1

From the execution of the code, the output is printed in the terminal. However, for management purposes, the same output can be found on the rules.txt file inside the Data folder.

6. CONCLUSION

All the goals initially proposed were achieved with success and will be explained during the conclusion.

Firstly, it was comprehended that RULES algorithmic complexity depends heavily on the number of features and classes, even though the latter was not demonstrated. Thus, performing preprocessing regarding the features of a dataset might be valuable for the classification task.

Regarding the dataset, it is fundamental to evaluate the intrinsic level of ambiguity relative to the instances. Otherwise, some of the resulting rules are contradictory, prejudicing the learning process.

Moreover, the number of rules created relies on the number of features and the inherent difficulty of the pattern to be learned (instances). Thus, if the latter is too fuzzy, it might result in a substantial quantity of rules, that must be pruned to prevent overfitting.

Furthermore, with respect to the complexity of the rules, their level depends on the number of antecedents, with which, mainly, the coverage and recall vary.

To conclude, RULES is a suitable algorithm for symbolic inductive learning that does not suffer from the irrelevant-condition problem.

7. BIBLIOGRAPHY

[1] PHAM, D. T.; AKSOY, M. S. (1995). *RULES: A Simple Rule Extraction System*. USA: Elsevier Science.

[2] SHAFQAT, H.; VALÉRIO, J.; GRYTØYR, E. (2022). *Work 2: Dimensionality Reduction and Visualization*. Spain: Universitat de Barcelona.