Master in Artificial Intelligence
Supervised and Experiential Learning

Rule Induction from a Set of Exemplars (RISE)

Romero Antolin, Axel

December 31, 2023

Contents

1 Introduction

For the initial project in the Supervised and Experiential Learning course, we conducted an implementation and validation of the Rule Induction from a Set of Exemplars (RISE) rule-based classifier. A rule-based classifier is a type of classifier that predicts outcomes based on a series of rules. These rules can range from simple if-then statements to more complex logical expressions. The classifier evaluates each rule and applies the corresponding prediction if the rule conditions are met. In Section 2, we provide a detailed explanation of the RISE algorithm, which involves the automatic learning of rules from a given data set.
The implementation was completed using Python 3, and the corresponding code is accessible within the `source` folder. Additional information regarding the code organization and execution can be found in Section 3. To validate the implementation of the RISE model, we utilized three distinct data sets of varying sizes to assess performance and the ability to classify new instances. The test outcomes can be seen in Section 4.

2 RISE

The RISE algorithm is a rule-based classifier originally developed by Pedro Domingos [1] where the rules are learned by gradually generalizing instances until there is no improvement in accuracy. The main characteristics of this algorithm are that it can handle numerical variables, which is something not very usual in other famous algorithms like RULES, PRISM and CN2, and that it produces a high number of rules that are, in general, very specific. This is due to the fact that the algorithm initially treats all instances in the training data as rules, which are subsequently generalized to improve performance.
In RISE, rules consist of two components: a consequent part that predicts the class and an antecedent part that comprises a combination of conditions. Each condition involves only one attribute, with categorical variables being tested for equality and numeric variables being tested within a closed interval on both sides.

The initial stage of the algorithm involves rule learning. Specifically, the algorithm treats all instances within the dataset as rules and calculates the initial accuracy of the models using a leave-one-out approach. For each instance, the algorithm removes the instance and its corresponding rule and utilizes the remaining rules to classify the instance, subsequently obtaining an average performance measure for the set of rules.
Then, for each rule in the set of rules, we find the nearest instance to the rule not covered by the rule and being of the same class. We said that an instance is covered by a rule if all the attributes of the instance satisfy the antecedent of the rule. The distance $\Delta(R, E)$ used in this algorithm between R and E, where $R = (A_1, A_2, \cdots, A_p, C_R)$ is a rule with class C_R and condition A_i for the ith attribute and $E = (e_i, e_2, \cdots, e_p, C_E)$ is an instance with class C_E and value e_i for the ith attribute, is defined as:

$$\Delta(R, E) = \sum_{i=1}^{a} \delta^s(i) \tag{1}$$

where s is a natural-valued parameter ($s = 1, 2, 3, \cdots$), and the component distance $\delta(i)$ for the ith attribute is:

$$\delta(i) = \begin{cases} 0 & \text{if } A_i = \text{True} \\ SVDM(r_i, e_i) & \text{if } i \text{ is categorical and } A_i \neq \text{True} \\ \delta_{num}(i) & \text{if } i \text{ is numeric and } A_i \neq \text{True} \end{cases} \tag{2}$$

where $SVDM(r_i, e_i)$ is the simplified value difference metric defined as:

$$SVDM(x_i, x_j) = \sum_{h=1}^{c} |P(C_h|x_i) - P(C_h|x_j)|^q \tag{3}$$

where c is the number of classes, C_h is the hth class, and q is a natural-valued parameter. Then, the distance metric for the numerical attributes is defined as:

$$\delta_{num}(i) = \begin{cases} 0 & \text{if } r_{i,lower} \leq e_i \leq r_{i,upper} \\ \frac{e_i - r_{i,upper}}{e_{i,max} - e_{i,min}} & \text{if } e_i > r_{i,upper} \\ \frac{r_{i,lower} - e_i}{e_{i,max} - e_{i,min}} & \text{if } e_i < r_{i,lower} \end{cases} \tag{4}$$

Once we have the nearest instance to the actual rule, we generalize the rule to make it cover the instance. We have the rule $R = (A_1, A_2, \cdots, A_p, C_R)$ and the instance $E = (e_1, e_2, \cdots, e_p, C_E)$ where the condition A_i is either $True$, $e_i = r_i$ (categorical) or $r_{i,lower} \leq e_i \leq r_{i,upper}$ (numerical). For each attribute i, if $A_i = True$ then do nothing; if i is categorical and $e_i \neq r_i$ then $A_i = True$; if i is numerical and $e_i > r_{i,upper}$ then $r_{i,upper} = e_i$; and if i is numeric and $e_i < r_{i,lower}$ then $r_{i,lower} = e_i$.
With the generalized rule, we create a new set of rules where we replace the actual rule with the generalized one. With this new set of rules, we calculate the accuracy using the same procedure of leave-one-out and, if the value is higher than before we maintain this new set of rules.
We repeat this process until there is no improvement in the accuracy of the set of rules. When the process has finished, we have the final set of rules derived from the training data that will be used for the prediction of new instances, which is the second part of the model. In this algorithm, we classify new instances by assigning them the class of the nearest rule using the specific distance metric defined before.

3 Methodology

In this section, we explain the implementation of the RISE algorithm in Python 3 and the experimental approach to analyze the performance of the algorithm and the ability to generate accurate classifications.

3.1 Implementation

In the previous section, we explained the theoretical background of the RISE algorithm that we used to implement it in Python 3 following the original paper [1] with some small changes. All the code is available in the `source` folder and the instructions to execute the code will be given at the end of the section. The implementation of the algorithm is in the `RISE.py` and the script used to execute the algorithm for the experimental phase is `main.py`. We created a Python class called `RISE` with the following parameters:

- `dataset_name`: name of the data set to use to train and evaluate the RISE algorithm. This value has to be one valid name of a csv file in the `data` folder.
- `target_name`: name of the class variable.
- `cat_variables`: list with the name of the categorical variables of the data set.
- `test_split`: the value of the percentage of instances of the data set separated for the evaluation phase, the test data. The rest of the instances are used for training.
- `max_iter`: maximum number of iterations in the training process.

There are different important parts in the `RISE` class of the algorithm that we are going to explain in detail in the following paragraphs. First, we integrated into the class the import and preprocessing

of the data. We import the data set using the `dataset_name` parameter where the file has to be in csv format. For the preprocessing, first, we check if there are missing data and, in the case of missing values, we imputed them with mean or mode depending if the variable is numerical or categorical. Then, for the categorical variables of the data set, we encoded the values to avoid characters and have integers that make it easier to calculate distances and manage the rules. Finally, we performed the train and test split of the data for the training and evaluation using the test size of the parameter `test_split`.

Then, to represent the rules and be able to calculate distances between them and instances, we use arrays where we store the values of the attributes to form the antecedent part. For the categorical variables, we have the value and for the numerical variables we have an interval represented as "value-value". The last value of the array is the consequent part, the predicted class.

The next step is the implementation of the training phase with the `fit()` method. In Algorithm 1 we can see the implemented algorithm of the main procedure of the RISE that we explained in Section 2. The initial part is the initialization of the set of rules where each instance is transformed into a rule with the `initialize_rules()` method. As a result, we obtain an array with all the rules where the size is equal to the size of the training data. Then, we calculate the initial accuracy of the set of rules with `estimate_accuracy()` and start the while loop until there is no improvement in the accuracy. For the estimation of the accuracy, we used the leave-one-out method where, when we predict the class of an instance, we remove the rule related to that instance and try to predict it with the rest of the rules. If we do not add this procedure we will obtain always a precision of 1 since for the prediction of each instance we will use the rule created from that instance. The method `estimate_accuracy()` classifies each instance with the class label of the nearest rule using the distance function explained in Section 2. The accuracy can be estimated with the actual set of rules or with a specific set that can be introduced as a parameter.

Then, for each rule, we find the nearest instance to the rule not covered and with the same class label with the `nearest_instance_to_rule()` and the process of generalization of the rule, explained in Section 2, is implemented in the `rule_generalization()`. Finally, we evaluate the new accuracy of the updated set of rules and decide whether keep the new rule or not. After some steps, we obtained the final set of rules that can be used to classify new instances. In this implementation, we add a maximum number of iterations, which is something that was not implemented in the original paper.

We also add a procedure for the elimination of repeated rules that was not explained in the original pseudo-code. In Algorithm 1 this procedure is also added. It consists of checking if each rule is also present in the previous rules and, if not, continuing with the normal procedure. After the generalization of the rule, we check again if the created rule is already present in the set of rules, if not we keep the rule.

Once we have the final set of rules derived from the training data, we implemented a method for the prediction of new instances using the trained set of rules, `predict()`. Also, to evaluate the performance of the set of rules, we implemented the `evaluate()` method which, for both train and test, computes different metrics to evaluate the model and creates a txt file with the value of the metrics. The details of the evaluation metrics will be given in the next section.

For the execution of the RISE algorithm and use it with a custom data set we have to use the `RISE` class and the `main.py` script in the `source` folder containing the code used in the experimental analysis. To use a custom data set we have to add it to the `data` folder in a csv file. To train the model we have to execute the `fit()` method that returns the final set of rules and the `evaluation()` method to test the model and obtain a txt file with the metrics.
During the implementation of the algorithm we had to face different problems, but the most important

Algorithm 1 RISE algorithm

```
rule_set ← initialize_rules()
initial_accuracy ← estimate_accuracy(Rule_set)
acc ← Initial_accuracy
acc_increased ← True
c ← 0
while acc_increased and c < max_iter do
    for rule in Rule_set do
        new_set_rules ← copy(rule_set)
        if rule repeated then
            new_set_rules[i] ← None
        else
            nearest_instance ← nearest_instance_to_rule(rule)
            new_rule ← rule_generalization(rule, nearest_instance)
            new_set_rules[i] ← new_rule
        end if
        new_acc ← estimate_accuracy(new_set_rules)
    end for
    if new_acc < acc then
        acc ← new_acc
        if new_rule repeated then
            new_set_rules[i] ← None
        end if
        set_rules ← new_set_rules
    end if
    acc_increased ← acc > initial_accuracy
    initial_accuracy ← acc
    c ← c + 1
end while
```

and the one that we dedicate more time was the optimization of the training. The execution time of the training is very long even with the process of optimization and this is due to the high number of times that we have to iterate over all the instances of the data set. Because of this problem, we could not manage to test the model with a data set with a high number of instances and it is the main problem of this algorithm.

The following code is an example of how to execute the RISE algorithm with this implementation for the Heart Disease data set:

```
cat_variables_heart = ['sex', 'chest', 'sugar', 'ecg', 'angina', 'slope',
'thal', 'disease']

model = RISE(dataset_name='heart',
        target_name='disease',
        cat_variables=cat_variables_heart,
        test_split=0.2,
        max_iter=10)
final_rules_heart = model.fit()
model.evaluate()
```

3.2 Evaluation

For the evaluation of the implementation of the RISE algorithm, we first try to test it in three data sets from the UCI Machine Learning Repository of different sizes and with numerical and categorical attributes. Also, we included some data sets with missing data to test the preprocessing part of the algorithm.
As mentioned in the previous section, the execution time of the RISE algorithm is very long and we could not manage to test it in a large data set since the different tests produced an expected training time of 7 days per iteration. We decided to, a part of the two first UCI data sets, include one simple and small data set from the theoretical slides used to test the implementation and it is easy to interpret the results. The data sets from the UCI Machine Learning Repository are:

- **Heart Disease**. Data set with 270 instances and 14 attributes where the class variable is `disease`. The set of variables contains numerical and categorical attributes and there is no missing data. The variables are:
  - age: age in years
  - sex: sex (1 = male; 0 = female)
  - chest: chest pain type where 1: typical angina; 2: atypical angina; 3: non-anginal pain; and 4: asymptomatic
  - rbp: resting blood pressure (in mm Hg on admission to the hospital)
  - cholesterol: serum cholesterol in mg/dl
  - sugar: (fasting blood sugar $>$ 120 mg/dl) (1 = true; 0 = false)
  - ecg: resting electrocardiographic results where 0: normal; 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of $>$ 0.05 mV); and 2: showing probable or definite left ventricular hypertrophy
  - max hr: maximum heart rate achieved
  - angina: exercise-induced angina (1 = yes; 0 = no)
  - oldpeack: ST depression induced by exercise relative to rest
  - slope: the slope of the peak exercise ST segment where 1: upsloping; 2: flat; and 3: downsloping

  - vessels: number of major vessels (0-3) coloured by fluoroscopy
  - thal: 3 = normal; 6 = fixed defect; 7 = reversible defect
  - disease: diagnosis of heart disease (angiographic disease status) where 0: $< 50\%$ diameter narrowing; and 1: $> 50\%$ diameter narrowing
- **Breast Cancer**. Data set with 699 instances and 10 attributes where the class variable is `Class`. In this case, the set of variables contains only numerical attributes and there are 16 instances that have missing data. The variables are:
  - Clump Thickness: Represents the thickness of the tumour in mm
  - Uniformity of Cell Size: Represents the uniformity in the size of the cells
  - Uniformity of Cell Shape: Represents the uniformity in the shape of the cells
  - Marginal Adhesion: Represents the degree of adhesion of the tumour cells to nearby cells
  - Single Epithelial Cell Size: Represents the size of the epithelial cells. It is a numerical attribute
  - Bare Nuclei: Represents the presence of a nucleus in the tumour cells
  - Bland Chromatin: Represents the degree of chromatin staining in the tumour cells
  - Normal Nucleoli: Represents the size and shape of the nucleoli in the tumour cells
  - Mitoses: Represents the number of mitoses (cell division) in the tumour cells
  - Class: Represents the diagnosis of the tumour (0 for benign, 1 for malignant)

To measure the performance and ability to classify new instances of the RISE model we used train/test cross-validation where the training split was used to train the model and the test split to evaluate the performance in a new set of instances. The metrics used for the evaluation are:

- **Accuracy**: Train and test general accuracy of the model. This is the ratio between the number of instances correctly classified using the set of rules and the total number of instances.
- **Rule recall**: The ratio between the number of instances satisfying the antecedent of the rule and the consequent of the rule, and the total number of instances belonging to the same class label that rule is classifying.
- **Rule accuracy**: The ratio between the number of instances satisfying the antecedent of the rule and the consequent of the rule, and the number of instances satisfying the antecedent of the rule.
- **Rule coverage**: The ratio between the number of instances satisfying the antecedent of the rule and the total number of instances.

4 Results

In this section, we will present the results obtained in the evaluation of the RISE algorithm in the three data sets and the procedure explained in the previous section. We will present the values of the metrics and the most important rules derived from the data for each one of the data sets. All the results are stored in the `results` folder where, for each data set, we generate two csv files with the final set of rules ordered by the recall and coverage metric for both train and test, a txt file with details of the training process and a txt file with the results of the evaluation.

4.1 Toy data set

This small data set was used for testing the implementation of the RISE algorithm to find errors in the implementation. As it is a small data set with a small number of attributes it is easy to interpret. The general accuracy for both training and testing is 1.

The final set of rules consists of 18 rules using the training data which has 19 instances. We can see that more or less each instance is transformed into a final rule showing the high amount of rules that this algorithm generates. Table 1 shows the most important rules based on recall and coverage. For example, the first rule says that a young person with astigmatism and normal tear production is classified with a hard label. The other attributes that do not appear in this rule are not taken into account for the classification. The recall of the rule is 0.67 which indicates that for the cases with the label RecomendedLense = hard, 66.7% are covered with the rule.

For this data set, since all the attributes are categorical, the conditions are equalities and we can see how the rules have been generalised since some attributes do not appear. Also, the set of important rules presented in Table 1 covers all the possible values of the class variable.

Rule	**Coverage**	**Accuracy**	**Recall**
(Age = young) AND (Astigmatism = yes) AND (TearProduction = normal) THEN RecomendedLense = hard	0.105	1	0.667
(VisualDeficiency = myopia) AND (Astigmatism = yes) AND (TearProduction = normal) THEN RecomendedLense = hard	0.105	1	0.667
(VisualDeficiency = hypermetropia) AND (Astigmatism = no) AND (TearProduction = normal) THEN RecomendedLense = soft	0.157	1	0.6
(VisualDeficiency = hypermetropia) AND (Astigmatism = no) THEN RecomendedLense = none	0.263	1	0.454
⋮	⋮	⋮	⋮

Table 1: Most important rules for the toy data set.

4.2 Heart Disease

The training of the rules for the Heart Disease data set was done with 80% of the data, which is 216 instances, and the rest, 54 instances, were used to evaluate the model. The training took approximately 2h and a total of 8 iterations (the maximum number of iterations was 10).
After the training, we obtain a total accuracy in the training set of 0.990 and a value of 0.769 in the test set. This shows that the RISE algorithm, for this data set, is capable of classifying the instances between the different classes with high precision. The final set of rules consists of 122 different rules where we can see some examples of the most important rules obtained from the data with the RISE algorithm in Table 2 together with the accuracy, recall and coverage of each one. For all the rules we obtain a very low coverage, which indicates that the rules are specific and they only cover a few instances. The high accuracy of the rules is also explained by the low coverage which means that the small number of instances covered by the rule are of the same class as the rule. Then, if we analyze the recall of the rules we can see that, with the three first rules, we cover the 33% of the instances of the class disease = 0.
In the `results` folder there is available the evaluation of the rules in the test set, where we can see that not all of the rules cover instances in the test. Most of the rules have a 0 value for the coverage which makes also a 0 for the recall and impossible to estimate the accuracy. This can be interpreted as that several rules are very specific to the training data and are not very useful in the inference of

new data. This behaviour could explain the overfitting presented by the model in this data set. For the rules that have coverage greater than 0, we obtain an accuracy of 1.

Rule	**Coverage**	**Accuracy**	**Recall**
$(52 \leq \text{age} \leq 67)$ AND $(115 \leq \text{rbp} \leq 152)$ AND $(232 \leq \text{cholesterol} \leq 564)$ AND $(147 \leq \text{max hr} \leq 178)$ AND $(\text{angina} = 0)$ AND $(0.4 \leq \text{oldpeack} \leq 2.3)$ AND $(0 \leq \text{vessels} \leq 0) \implies \text{disease} = 0$	0.060	1	0.111
$(41 \leq \text{age} \leq 59)$ AND $(\text{sex} = 1)$ AND $(118 \leq \text{rbp} \leq 135)$ AND $(186 \leq \text{cholesterol} \leq 258)$ AND $(132 \leq \text{max hr} \leq 190)$ AND $(\text{angina} = 0)$ AND $(0 \leq \text{oldpeack} \leq 1.2)$ AND $(0 \leq \text{vessels} \leq 0) \implies \text{disease} = 0$	0.055	1	0.102
$(29 \leq \text{age} \leq 48)$ AND $(\text{sex} = 1)$ AND $(118 \leq \text{rbp} \leq 140)$ AND $(182 \leq \text{cholesterol} \leq 321)$ AND $(\text{sugar} = 0)$ AND $(156 \leq \text{max hr} \leq 202)$ AND $(\text{angina} = 0)$ AND $(0 \leq \text{oldpeack} \leq 0)$ AND $(\text{slope} = 0)$ AND $(0 \leq \text{vessels} \leq 0)$ AND $(\text{thal} = 0) \implies \text{disease} = 0$	0.055	1	0.102
$(54 \leq \text{age} \leq 61)$ AND $(\text{sex} = 1)$ AND $(\text{chest} = 3)$ AND $(110 \leq \text{rbp} \leq 160)$ AND $(239 \leq \text{cholesterol} \leq 353)$ AND $(\text{sugar} = 0)$ AND $(88 \leq \text{max hr} \leq 145)$ AND $(\text{angina} = 1)$ AND $(0.8 \leq \text{oldpeack} \leq 3.6)$ AND $(\text{slope} = 1)$ AND $(1 \leq \text{vessels} \leq 2)$ AND $(\text{thal} = 2) \implies \text{disease} = 1$	0.046	1	0.101
⋮	⋮	⋮	⋮

Table 2: Most important rules for the Heart Disease data set.

4.3 Breast Cancer

The training of the rules for the Breast Cancer data set was done with 80% of the data, which is 559 instances, and the rest, 140 instances, were used to evaluate the model. The training took approximately 8h and a total of 4 iterations (the maximum number of iterations was 5).
The general accuracy of the final model for the training set is 0.998 and for the test set 0.964. The performance of the RISE algorithm for both training and testing is very good and we can see that the rules learned by the algorithm generalize well and are capable to make good predictions in new external data. The final set of rules consists of 273 different rules and in Table 3 we can see the top 4 most important rules based on the recall and coverage.
If we check in the `results` folder all the rules ordered by higher recall and coverage and we can see that all the initial rules (the most relevant) are for Class = 0. Then, for all the rules in the table, we obtain a similar coverage of around 0.15 which means that each rule covers approximately 15% of the instances of the training set. For the recall, we obtain also similar values between 0.21 and 0.27 where, for example, in the first rule, the value of 0.27 means that 27% of the instances of Class = 0 are covered by the rule. Then, the accuracy of the rules is one as in the case of the previous data set. The last rule presented in Table 3 is the first one for Class = 1 which has very low coverage and recall. These low values and the fact that the most relevant rules are for the 0 class can be explained by the unbalanced nature of the data set where 65.5% of the instances are classified as 0 (benign).

Rule	**Coverage**	**Accuracy**	**Recall**
$(2 \leq$ Clump Thickness $\leq 6)$ AND $(1 \leq$ Uniformity of Cell Size $\leq 2)$ AND $(1 \leq$ Uniformity of Cell Shape $\leq 2)$ AND $(1 \leq$ Marginal Adhesion $\leq 1)$ AND $(1 \leq$ Single Epithelial Cell Size $\leq 2)$ AND $(1 \leq$ Bare Nuclei $\leq 2)$ AND $(2 \leq$ Bland Chromatin $\leq 7)$ AND $(1 \leq$ Normal Nucleoli $\leq 1)$ AND $(1 \leq$ Mitoses $\leq 1) \Longrightarrow$ Class $= 0$	0.175	1	0.270
$(1 \leq$ Clump Thickness $\leq 2)$ AND $(1 \leq$ Uniformity of Cell Size $\leq 1)$ AND $(1 \leq$ Uniformity of Cell Shape $\leq 1)$ AND $(1 \leq$ Marginal Adhesion $\leq 3)$ AND $(2 \leq$ Single Epithelial Cell Size $\leq 2)$ AND $(1 \leq$ Bare Nuclei $\leq 1)$ AND $(1 \leq$ Bland Chromatin $\leq 3)$ AND $(1 \leq$ Normal Nucleoli $\leq 1)$ AND $(1 \leq$ Mitoses $\leq 1) \Longrightarrow$ Class $= 0$	0.152	1	0.234
$(1 \leq$ Clump Thickness $\leq 2)$ AND $(1 \leq$ Uniformity of Cell Size $\leq 1)$ AND $(1 \leq$ Uniformity of Cell Shape $\leq 1)$ AND $(1 \leq$ Marginal Adhesion $\leq 2)$ AND $(2 \leq$ Single Epithelial Cell Size $\leq 2)$ AND $(1 \leq$ Bare Nuclei $\leq 1)$ AND $(1 \leq$ Bland Chromatin $\leq 3)$ AND $(1 \leq$ Normal Nucleoli $\leq 1)$ AND $(1 \leq$ Mitoses $\leq 1) \Longrightarrow$ Class $= 0$	0.150	1	0.231
$(1 \leq$ Clump Thickness $\leq 3)$ AND $(1 \leq$ Uniformity of Cell Size $\leq 1)$ AND $(1 \leq$ Uniformity of Cell Shape $\leq 1)$ AND $(1 \leq$ Marginal Adhesion $\leq 1)$ AND $(2 \leq$ Single Epithelial Cell Size $\leq 2)$ AND $(1 \leq$ Bare Nuclei $\leq 1)$ AND $(2 \leq$ Bland Chromatin $\leq 3)$ AND $(1 \leq$ Normal Nucleoli $\leq 2)$ AND $(1 \leq$ Mitoses $\leq 1) \Longrightarrow$ Class $= 0$	0.136	1	0.210
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(6 \leq$ Clump Thickness $\leq 10)$ AND $(10 \leq$ Uniformity of Cell Size $\leq 10)$ AND $(9 \leq$ Uniformity of Cell Shape $\leq 10)$ AND $(1 \leq$ Marginal Adhesion $\leq 3)$ AND $(6 \leq$ Single Epithelial Cell Size $\leq 10)$ AND $(1 \leq$ Bare Nuclei $\leq 10)$ AND $(2 \leq$ Bland Chromatin $\leq 7)$ AND $(3 \leq$ Normal Nucleoli $\leq 8)$ AND $(1 \leq$ Mitoses $\leq 3) \Longrightarrow$ Class $= 1$	0.010	1	0.030
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 3: Most important rules for the Breast Cancer data set.

5 Conclusions

In this practical work, we implemented a rule-based classifier from scratch and evaluated its performance using various datasets. Based on the results of our experiments we can see that the RISE algorithm has a high classification capacity, as evidenced by its high train and test accuracy. By applying this algorithm, we can create a set of rules that can be used to predict new instances with high precision.
However, this algorithm has certain limitations that make its use challenging. Firstly, the number of generated rules is very high. In the beginning, the algorithm creates one rule for each instance and then generalizes them. Even with smaller datasets, we observed that the number of rules generated was excessively high, exceeding one hundred for each dataset. This makes interpreting the model and the set of rules challenging, particularly as most of the rules are very specific and derived from the training data but not used to predict new instances.

Another critical issue with this algorithm is the prolonged execution time required for the training phase. The computational expense associated with checking and generalizing all the rules significantly increases the training time, which becomes exceedingly long. In our experiments, we had to train our model for more than six hours to obtain a final set of rules. Consequently, this algorithm is unsuitable for use with large datasets (comprising more than 2000 instances) as the execution time would be prohibitively long, possibly taking days.
During the implementation phase, a significant amount of time was dedicated to testing and experimenting with methods to improve the code's efficiency and reduce the execution time. As future work, further optimization of the code would be beneficial, leading to a final version of the algorithm capable of handling large datasets.

References

[1] Pedro Domingos. "Unifying Instance-Based and Rule-Based Induction". In: *Mach. Learn.* 24.2 (Aug. 1996), pp. 141–168. ISSN: 0885-6125. DOI: 10.1023/A:1018006431188. URL: https://doi.org/10.1023/A:1018006431188.