RISE Classifier

Fernando Vázquez Novoa

9th April 2021

1 Introduction

The objective of this work is to implement and validate a rule-based classifier. The rule-based classifier implemented specifically on this work is the RISE algorithm.
The selected language to use in the development of this algorithm is Python. A rule writer in order to store the final rules with their accuracy and coverage obtained from the training data was developed. And also a rule interpreter was developed in order to read these rules and apply them to a test data.

1.1 Background and Approach

Instance-Based Learning

Instance-based learning is the group of algorithms that compare new problem instances with instances already seen in training step, the instances seen in training step are stored in memory, and classifies the new instances according to the nearest or more similar one stored in memory. The performance of these type of algorithms is strongly dependant on the distance metric used.
These type of algorithms has some limitations and problems. They are very sensitive to irrelevant attributes, these attributes can affect the classification task more than the relevant attributes and produce the algorithm to misclassify examples. Another big problem that the IBL algorithms should face is their sensitivity to noise, incorrect instances that are stored can create regions of examples bad classified.

Rule Induction

Rule Induction algorithms is a family of algorithms that extract from a training set of data extract rules that cover many positive examples and few or no negative ones. It works building rules in order to classify correctly the training data. The new examples are classified by the rule that matches against them, if no rules match an example a default rule is usually selected and for multiple matching there are different strategies in order to decide the rule.
This type of algorithms present some problems. It causes a dwindling number of examples to be available as induction progresses, both within each rule and for successive rules. This effect can cause that the later rules, and the later antecedents of each rule, are induced with insufficient statistical support, causing that they have a greater sensitivity to noise and maybe some missing or incorrect rules or antecedents.

RISE Algorithm

The RISE algorithm is an approach to induction that tries to solve the limitations of instance-based learning and rule induction by unifying the two. This approach treats in the same way the rules and the instances.
A rule in RISE is made by a consequent that is the class that rule predicts and the antecedent that is the conjunction of the conditions for an example to be covered by that rule. The new examples are classified by RISE assigning to them the class of the nearest rule to that example stored in the knowledge base. The distance between a rule and an example is calculated in the following way:

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

Where s is a natural valued parameter that defines the distance computed (1=Manhattan, 2=Euclidean, etc). Delta is the distance for the ith attribute and is computed in the following way:

$$\delta(i) = \begin{cases} 0 & \text{if } A_i \text{ is True} \\ SVDM(r_i, e_i) & \text{if } A_i \text{ is symbolic} \\ \delta_n(i) & \text{if } A_i \text{ is numeric} \end{cases} \tag{2}$$

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

Where q is a natural valued parameter that can be determined empirically or can get the value desired.

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

In the case that more than one rule is at the same distance from an example the rule selected is the one with the highest Laplace accuracy in order to priorize rules with high accuracy as well as strong statistical support.
In the case that the two accuracies are the same the rule that has the more probable class as consequent is selected, in case there is still a draw it is selected randomly.

A rule can cover an example and not win it, the winner rule is the nearest rule to the example following the policy defined.

RISE induces all rules in parallel at the same time. The train data is transformed into the initial set of rules, each instance is going to become a rule in the initial rule set. Then the rules are generalized using their nearest example of the same class and not already covered by it. The symbolic attributes are dropped in order to generalize and the numeric intervals are enlarged. The effect of generalization of each rule is measured in terms of global accuracy and the generalizations are accepted if they improve the global accuracy or if they do not make the set of rules to get a smaller accuracy.

2 Pseudo-code

The code of this implementation is strongly influenced by the pseudo-code and the explanation of the algorithm made on the RISE paper[1]. Some small modifications were introduced in that pseudo-code in order to adapt it to the language of development that is python.
The final pseudo-code of the algorithm is the following:

Algorithm 1 RISE Adaptation

```
 1: procedure RISE (ES training set, Y labels training set)
 2:     Store max value of each numeric attribute
 3:     Store min value of each numeric attribute
 4:     Compute SVDM for all attributes and classes
 5:     $RS \leftarrow ES$
 6:     $I\_Accuracy \leftarrow ACC(RS)$
 7:     while True do
 8:         $Flag \leftarrow False$
 9:         for Each rule in RS do
10:             $NE, Index \leftarrow Get\_Nearest\_Example(ES, rule, Y)$
11:             if Index != -1 then
12:                 $Most\_Specific\_Generalization(R, NE, Index)$
13:                 $Actual\_Accuracy, NNR \leftarrow ACC(RS')$
14:                 if $Actual\_Accuracy >= Initial\_Accuracy$ then
15:                     $Flag \leftarrow True$
16:                     $I\_Accuracy \leftarrow Actual\_Accuracy$
17:                     $RS \leftarrow RS'$
18:                     Update Nearest Rules
19:                     if $Check\_R'\_Is\_Unique! = True$ then Break
20:         if $Flag! = True$ then Compute Acc(RS) and Break
```

In the small changes made to the pseudo-code proposed for the body of the algorithm on the paper I added the storing of the maximum and minimum values of the numeric attributes, used later in the computation of the distances of the attributes with respect to the rules, in the starting of the execution. Also the SVDM is computed for all the attributes and classes in the beginning of the execution (the values are stored and only computed once, no needing to calculate them each time an instance is matched against a rule).

A flag was introduced in order to know when the algorithm tries to generalize rules and all the changes make worse the set of rules in order to break the execution if this happens. Also a break had to be introduced when a rule is deleted. When a rule is deleted the for loop is returned to the rule 0 in order to continue generalizing. This is necessary in order to not get an exception when the loop goes to an index already deleted (when the for loop is started in python it will go until the initial condition, if a rule is deleted the loop is going to continue until an out of range is reached.)

The function Get_Nearest_Example also is changed, when no example is available to be returned for the rule a -1 as index is returned in order to avoid repeating computations for an already covered example and to ensure that are not returned incorrect near examples (for example belonging to a different class than the rule class).
After the adaptation of the body of the algorithm some changes in some functions were also introduced. Not on the pseudo-code but on the way that functions work.
For example the function Most_Especific_Generalization works in the same way as in the original pseudo-code but it is not needed to replace the rule because it changes the specified rule, it does not return a copy of the rule changed.
The function Check_R´_is_Unique is a new function and in addition to check if the newly created rule is unique it deletes the rule in the case it is not unique. Its pseudo-code is the following:

Algorithm 2 Check_R'_Is_Unique

```
procedure Check_R'_Is_Unique (I index of the changed rule)
    for $j = 0; j < len(RS); j = j + 1$ do
        if $j! = i$ then
            if $Rule[i] == Rule[j]$ then
                $Rule[j][coveredElements] = Rule[j][coveredElements] + Rule[i][coveredElements]]$
                Update identifiers of nearest rules
                $Rule[i].pop()$
```

3 Results

The rule-base classifier has to be evaluated into three different datasets, these datasets should have different sizes. The small dataset should have a maximum number of 500 instances, the medium dataset should have between 500 and 2000 instances and the big dataset should have more than 2000 instances.

3.1 Preprocessing

The data is preprocessed before using it to create the rules and evaluate their performance.

Numerical attributes

A Min-Max scaling is applied to all numeric attributes, it converts their values to the range [0,1], this is done in order to make all numeric attributes affect in the same way to the distance between rules and examples. If the numeric attributes have different ranges they will affect in a different way to the computation of th edistance of the rule.

Missing values

The RISE algorithm handles in a correct way the missing values in the attributes. In order to encode the missing values for making the algorithm work with them (with a nan the execution breaks) I filled the symbolic missing values as it is proposed in the paper, with a '?' symbol. It is treated as another valid symbolic value.
The handling of the missing numeric values is not very clear in the paper on my opinion. I proposed two different ways of treating them depending on the step of the algorithm the execution is. First when it is the fase of constructing the initial rules and a missing numeric attribute is found I assign the two values out of the range of the possible values for the numeric attributes. I assign 2 to the start of the interval and -1 to the end (all the attributes have values between 0 and 1 due to the Min-Max scaler applied) and when these values are found the distances to all numeric values of this attribute is treated as 0. When the rule is generalized when an example, this attribute is set to the value of the new example that does not contain a missing value.
In the case that the missing value is in the example that is going to be used to generalize the rule the values of the rule for that attribute are not changed, also the distance of that two pairs of attributes is 0.

Split of datasets

The datasets are splitted into train, validation and test data. The training data is the 75% of the elements of the dataset, the validation data the 15% and finally the test data is the 10% of the instances.

3.2 Small dataset

The small dataset selected is the 'Glass' dataset. It contains 214 instances that have 9 numeric attributes. There are no missing values on this dataset. This dataset in addition to be selected as the small dataset for the evaluation of the algorithm was selected in order to compare the results obtained on this implementations with the results provided on the paper where this dataset was also used to evaluate the algorithm.

Accuracy

The accuracies shown in the following table are the mean values from 3 executions:

Glass results				
Measure	s=1, q=1	s=1, q=2	s=2, q=1	s=2, q=2
Initial Train Accuracy	0.726	0.735	0.704	0.688
Final Train Accuracy	0.993	0.994	0.986	0.960
Validation Accuracy	0.683	0.662	0.724	0.721
Test Accuracy	0.657	0.701	0.672	0.758

Different values of s and q are used in order to see if there is a best combination of values. It seems that the q=2 and s=2 is the one that returns the better results.
The results shown in the table are an average, big variances were obtained in the results and some were discarded. The big variances in the results obtained may be due to the fact that is a small dataset and the way the division of the data is performed may affect the final results in a strong way. For example, if in the training partition there are a small number of instances of one class and so many of the other, the rules are going to be biased by this partition.
These results are similar to the ones obtained using this dataset in the paper. The results summarized on the paper had an accuracy of 70.6% with a deviation of +/- 5.8%. All the results shown in the table are values that are in this interval so I suppose that the implementation of the algorithm is correct.

Rules

The rules extracted from this dataset are in the file *"my_rulesGlass.txt"* that is appended with this report. The example of rules appended got an accuracy in test and validation of 70% and were obtained using q=1 and s=1.
While the algorithm generalizes the rules some rules become identical to another already existent in the set of rules, in this case the rule is deleted and only a copy is stored in the rule set. In this specific case the algorithm deleted 100 rules, it started with 149 rules and ended with 49 rules.

Interpretation of the Rules

The rules obtained start by the number of rule it is and end with the class they classify the example into, the number of elements of the training dataset it wins (coverage) and the accuracy (the number of this instances the rule wins and are of the same class that it classifies divided by the total number of instances it wins). The rules contain conditions, each condition is separated from the rest by the word 'AND', also they are separated from the starting of the rule by the word 'AND' and from the end of the rule, the conditions for this dataset are all numeric conditions, they contain the lower bound of the interval, then the name of the attribute and after the upper bound of the condition.
When an example is matched against this type of rules the distance computed by each attribute is 0 if that attribute's value is between the range of the corresponding condition on that rule. In other case the distance of each attribute is calculated following the equation (4). The example gets its distance computed applying the equation (2).

Execution time

The execution time spent by the algorithm on extracting and generalizing the rules for this dataset is small and in the majority of cases it does not last more than about 3/4 minutes. The fastest execution lasted 12 seconds to extract and generalize the rules. The usual time to generalize the rules is about 200 seconds and the lowest execution lasted 403 seconds.

3.3 Medium dataset

The medium selected dataset is the 'Montgomery' dataset obtained from the Data World webpage. It is a dataset that contains information about different professions and their status, salary, grade etc. It is a dataset with 589 instances and 5 attributes for each instance. In the 5 attributes there are 3 numerical attributes and 2 symbolic attributes.

Accuracy

The accuracies shown for this dataset are the mean value of the accuracies obtained in three executions. The mean accuracies are in the following table:

Montgomery results				
Measure	s=1, q=1	s=1, q=2	s=2, q=1	s=2, q=2
Initial Train Accuracy	0.855	0.86	0.876	0.856
Final Train Accuracy	0.971	0.971	0.978	0.972
Validation Accuracy	0.90	0.76	0.905	0.860
Test Accuracy	0.827	0.796	0.809	0.709

Different values of s and q are used in order to see if there is a best combination of values. It seems that the s=1, q=1 and s=2, q=1 are the combinations that return better validation and test accuracy.
On this dataset the results obtained are better than in the small dataset. The initial train accuracy is higher in all the experiments and it is about 86%, the lower obtained value was a 82% and the maximum was 89%.
When the training ends for the train data the set of modified rules increases its accuracy untill about a 97% or 98% depending on the execution.
With respect to the validation and test accuracies it can be seen that the variation of the results is higher. The accuracies obtained go from a 90.5% to a 70.9%. The variation of the accuracies depend in a strong way of the split obtained for each execution, depending on the split it seems to return very different values.
Despite this big variation in the validation and test results, the performance of the algorithm is very good on this dataset as the smallest validation/test accuracy expected (at least the smaller obtained in the experiments) is of 70.6%.
The results obtained seem to be better on this dataset than in the small dataset. This can happen because as it is a bigger dataset the number of

examples that is used to extract the initial rules and improve them is bigger and they represent better all the changes in the dataset.

Rules

The rules extracted from this dataset are in the file *”my_rulesMont.txt”* that is appended with this report.
It can be seen that the number of deleted rules in proportion to the size of the dataset is higher than in the small dataset. From 450 initial rules the algorithm generalizes rules and deletes identical constructed rules until a number of 100 rules is left.

Interpretation of the Rules

The rules obtained start by the number of rule it is and end with the class they classify the example into, the number of elements of the training dataset it wins (coverage) and the accuracy (the number of this instances the rule wins and are of the same class that it classifies divided by the total number of instances it wins). The rules contain conditions, each condition is separated from the rest by the word 'AND', also they are separated from the starting of the rule by the word 'AND' and from the end of the rule. The conditions for this dataset are three numeric conditions and two symbolic conditions. The numeric conditions contain the name of the attribute surrounded by the lower bound of the interval and the upper bound. The symbolic conditions contain the name of the attribute and a TRUE, if they were generalized and dropped or an equal and the value of the symbolic condition if during the generalization the condition was not dropped.
When an example is matched against this type of rules the distance computed by each numeric attribute is 0 if that attribute's value is between the range of the corresponding condition on that rule. In other case the distance is calculated applying the equation (4). With symbolic attributes the distance is computed using the equation (3). And the distance of the whole example is computed using the equation (2).

Execution time

The execution time spent for the algorithm on extracting and generalizing the rules for this dataset changed drastically depending on the split of the data. The smallest execution time obtained was of 507 seconds. The highest execution time obtained was of 7584 seconds. The most common

execution time tend to last about 6000-7000 seconds.
It is a slow algorithm that spends a lot of time in extracting and generalizing the rules. In comparison with other rule-based classifiers algorithms is one of the slowest algorithms. A comparison of the execution times of different rule-based classifiers algorithms can be seen in [2]

3.4 Large dataset

The dataset selected with more than 2000 instances was obtained from the Data World webpage. It is the 'studentInfo.csv' dataset. This dataset contains 11 attributes per instance and it contains some missing values already filled with a '?' and only in one attribute ('imd_band'). The attributes of this dataset are numerical and symbolic attributes (8 symbolic, 3 numeric). The original dataset contained more than 30000 instances, in order to make it feasible to perform more than one run of the algorithm and be able to calculate the average accuracies it were deleted instances until 2667 instances were left. It was left this number of instances in order to get a training set of more or less 2000 instances at least (using as training set a 75% of the data).
When the deletion of the data was done it was assured that all the classes in the original dataset still have representation on the new dataset, and that the representation was not biased with respect to the original dataset.

Accuracy

Student results				
Measure	s=1, q=1	s=1, q=2	s=2, q=1	s=2, q=2
Initial Train Accuracy	0.338	0.338	0.334	0.341
Final Train Accuracy	0.626	0.606	0.618	0.547
Validation Accuracy	0.508	0.436	0.432	0.472
Test Accuracy	0.496	0.42	0.463	0.46

The same combination of values evaluated on the two previous datasets was also evaluated on this dataset. In this case the combination that seems better is the q=1, s=1.
The rules extracted for this dataset return a lower initial accuracy than in

the other two datasets,. In the glass dataset it was about a 70% and in the Montgomery dataset about a 86%, while in this dataset the initial accuracy of the rules extracted form the train data is about a 33%.
Taking into account these results it can be assumed that this is a dataset harder than the two previous datasets. During the training the initial low accuracy is improved while the rules are generalized until about a 60%. This value is also very low compared with the values obtained for the other two datasets that were about a 97% the Montgomery dataset and about a 99% in the glass dataset.
Finally the test and validation accuracies are low, lower than a 50% in the majority of experiments and in the higher ones it reached about a 52%.

This dataset obtains lower values than the previous ones and can be due to the fact that it may be considered a harder to classify dataset. From the eleven attributes it contains it can be observed that the majority of them take the same values between the different classes. And it seems that there are no attribute value that belongs to only one class.
The fact that is a more difficult dataset to classify can be observed also in the fact that a very small number of rules is deleted when generalizing this dataset and a small number of generalizations occur.

Rules

The rules extracted from this dataset are in the file *my_rulesStud.txt* that is appended with this report.
The number of rules deleted in proportion to the size of the dataset is very small. On the example rules appended the final number of rules is 1985 and the initial number is of 2040 rules. This means that the algorithm was only able to delete 55 rules out of 2040, a very small number.
In other executions the algorithm deleted rules until leaving a final set of rules with 1965 rules.
This can appear to happen due to a bad implementation of the algorithm but as it was said in the small dataset the implementation of the algorithm returns similar values as the ones in the paper[1].

Interpretation of the Rules

The interpretation of the rules can be done exactly in the same way that the interpretation of the rules explained for the medium dataset. It contains numeric and symbolic attributes and the same equations stated in the previous sections are valid to measure the distance between an example and a rule. These rules also contain the number of rule, their coverage and their accuracy.

Execution time

The execution time spent by the algorithm on extracting and generalizing the rules for this dataset changed a lot depending on the split obtained for the training data.
The fastest extracting and generalization of rules lasted 651 seconds. On this execution a very small number of generalizations occured and a small number of rules was dropped, the rules that were the final rules were 2014 rules out of the 2040 initial rules. Only 26 rules were dropped.
The lowest extracting and generalization fo rules lasted 8247 seconds. On this execution more generalizations of the rules occurred and also more rules were dropped, leaving the final set of rules with 1965 rules out of the initial 2040. This is still a big number of rules but as it was said before for this dataset it seem to be very hard to construct good rules in order to classify correctly the different instances.
The average time spent by the algorithm on this dataset goes from the 4000 seconds to the 7000 seconds depending on the execution.

4 Additional comments on the algorithm

Treatment of missing values

The way the algorithm treats the missing values is specified in the previous sections. The datasets evaluated on this document do not contain missing values because I found more appropiate to use these datasets instead some datasets with missing values because of the content of the datasets and their types of attributes.
Some executions were made on datasets with missing values. The 'hypothyroid.arff' dataset is one of the datasets used with missing values. The algorithm was proved that works in datasets with missing values. The results obtained for this dataset are not included on this document because the execution time for this algorithm was very big and it was not possible to run the experiments several times, it is a dataset with 3772 instances and with 29 attributes per instance.
The algorithm works with datasets that contain missing values and it was

proved.

Splitting of data into train, validation and test

The splitting of the data into train, validation and test data was done because it was asked in the instructions of the project. In this case it is exactly the same the validation and test phase and the validation and test accuracy could be computed as the same accuracy. The unique thing that changes is the set and instances of data evaluated, but they could be evaluated all together and get a unique validation or test accuracy.
The differentiation between both was done in order to follow the work instructions.

Best combination of parameters

The different combinations of s and q values showed that the best combination of these values changes depending on the dataset. This result was in part expected.
This means that the best q value should be determined for the dataset that is going to be analyzed and also that the best metric of distance used depends on the dataset where the metric is going to be used, it should also be determined depending on the dataset.

5 Instructions of execution

The code developed for this algorithm is in the folder "Code" that is inside this project. There is a main.py file that contains the code of the main function to be executed and four more files. The rise.py that contains the main code of the algorithm. The read_dataset.py that contains the code that reads the dataset and preprocess it. The rule_writer.py that contains the class that writes the rules extracted from the training data into a file. And finally the rule_interpreter.py that contains the class that reads the rules from a file and applies them to the test set of instances.
The main function is made in order to execute all the steps of the algorithm one after another. If it is desired to execute only a part of the code another function should be developed.
This function was executed on a Linux environment, Ubuntu 20.04 was used and Python 3.7.4 was used. The main libraries needed are numpy, scikit-learn and pandas, their versions are 1.20.1, 0.24.1 and 1.2.3 respectively.

The main function need some arguments. It can get up to five arguments and not all arguments are necessary. The first one is the dataset that is going to be used specified by a -d in the execution. The second argument is the type of file the dataset is, it is specified by a -t, the possibilities are 'arff' or 'csv'. The third argument is the name of the output file and it is specified by a -o. These are the three compulsory arguments to be passed to the main function. Then there are two arguments that are -q and -s, these two arguments have a default value of 1, they are only need to be declared if a different value is wanted for any of them. An example of how the code should be execute is the following one:

```
$ python3 main.py --d=Montgomery-dataset --t=csv /
--o=my_rules.txt --q=2 --s=2
```

The dataset specified should be in the same directory than the main.py file.

6 Conclusion

An implementation of the algorithm is provided and the results obtained using it with different datasets are analyzed, the accuracies obtained for these datasets using different measures of the distance and hyperparameters and the time that the different processing of the datasets lasts.
A pseudo-code of the algorithm with small adaptations in order to make it feasible of being implemented on python is also provided.
On this work I got my first introduction to the development of rule-based classifiers. I learned in depth how the RULE based classifier works and also I got a clearer idea on how this type of algorithms works.
I also improved my capacity of reading and understanding scientific publications as I had to read and understand the paper where the algorithm is explained.

References

[1] Pedro Domingos, "Unifying Instance-Based and Rule-Based Induction" *Machine Learning, 24*, pp. 141-168, 1996.

[2] Miquel Sánchez-Marré, "Rule-based classifiers" *Supervised and Experiential Learning*, pp. 40, 2021.