PW1: RISE, a rule-based classifier

Alejandro Suárez Hernández

November, 2017

Abstract

This document revolves around a custom implementation of RISE, a rule-based classifier from the 90s by Pedro Domingos. We explore the learning and inference algorithm of RISE. Moreover we introduce some additional ideas that complement those presented in the original paper. Our implementation is tested with several well-known data sets, widely used in the literature. We expose the generated rules for some examples and quantify the performance of the classifier in terms of classification accuracy and training time.

1 Introduction

RISE [Dom96] is a rule-based classifier that is able to cope with both numerical and nominal attributes. Moreover, it is able to deal with missing values naturally without the need of data imputation. Contrarily to most other rule classifiers like RULES, PRISM or CN2, RISE's rule base evolves from highly specific rules (i.e. rules that describe each of the instances individually) to increasingly general rules.

Rules in RISE are composed of a conjunction of conditions (antecedent) and the implied class of an hypothetical instance that satisfies these conditions (consequent). Conditions are of two kinds: (1) boundaries for numerical input attributes; (2) and equalities for nominal input attributes. Rules do not need to be complete (the smaller the number of conditions, the more general). An example of such rule is:

$$18.92 \leq A_2 \leq 31.67, 2.54 \leq A_3 \leq 16.165, 1 \leq A_8 \leq 3, A_9 = t, A_{13} = g \rightarrow +$$

One of the particularities of RISE is that it heavily relies on the concept of distance between a rule and an instance. This is highly relevant both in the training phase and when classifying a new instance. In the first case, rules are progressively generalized via the so-called *most specific generalization* with the nearest instances. The rule-instance distance is employed to assess this. When classifying an unseen instance that is not completely covered by any of the rules in the rule base, the concept of the distance is very useful because it allows the classifier to make a decision nonetheless.

Our implementation retains most of the features described in the original work by Domingos and briefly summarized here: (1) it supports numerical and nominal attributes; (2) it can deal with missing values gracefully; (3) and it uses the notion of distances to classify examples that are not entirely covered by any rule of the rule base. Moreover we have incorporated several features on top of these:

- Our implementation allows to use several types of distances between categorical value. Domingos focus on the so called *Simplified Value Difference Metric* (SVDM) with a single parameter q (more details in Section 3). We also support the Gödel (or overlap) distance and the Kullback-Leibler (KL) divergence.
- Normalization of the distance. The original work did not contemplate the normalization of the distance. This is most important when the absence of an attribute (missing value) prevents the addition of a contribution term to the overall distance.

2 Rule metrics

As we have mentioned, the computation of distances between rules and instances plays a crucial role both in the training algorithm and in the classification algorithm. We mostly stick to the

strategy proposed in the original paper, with only a few differences: we normalize all the distances between 0 and 1, and we consider additional metrics for comparing nominal values, in addition to SVDM(q):

- The Gödel (or overlap) difference. The difference between two nominal categories is as follows:
$$D(a_i, a_i') = 1 - \delta_{a_i, a_i'}$$
, where $\delta_{i,j}$ is the Kronecker delta, a function that is equal to 1 exclusively when its two subindices are 0 (otherwise $\delta_{a_i, a_i'} = 0$).

- The Kullback-Leibler divergence is not a proper distance since it is not symmetrical. However, it is suitable to capture the notion of simmilarity between probability distributions, and that is the very reason why we have adopted it as an alternative to the SVDM distance. Its expression is:
$$KL(a_i, a_i') = -\sum_{C \in dom(C_i)} \mathbb{P}(C|a_i) \log\left(\frac{\mathbb{P}(C|a_i')}{\mathbb{P}(C|a_i)}\right)$$
. In addition to the asymmetry, one may also observe that the divergence is unbounded in the interval $[0, \infty)$. In order to make sure that the divergence is bounded but still informative, we map this infinite range into the bounded region $[0, 1)$ via the following bijection:
$$KL'(a_i, a_i') = \frac{1 - \exp(-KL(a_i, a_i'))}{1 + \exp(-KL(a_i, a_i'))}$$
.

Now, let us observe that the SVDM(q) distance presented in the paper is defined as follows:

$$SVDM_q(a_i, a_i') = \sum_{C \in dom(C_i)} |\mathbb{P}(C|a_i) - \mathbb{P}(C|a_i')|^q$$

. Contrarily to the distance between real attributes and the other nominal metrics, this metric is not bounded between 0 and 1. However, it is very straightforward to fix this. The SVDM(q) used by us in our implementation is:

$$SVDM'_q(a_i, a_i') = \frac{1}{|dom(C_i)|} SVDM_q(a_i, a_i')$$

.

The last modification that is worth stating is that the distance between a rule and an instance is also averaged using the count of non-missing values. With this, we ensure that the distance is always bounded between 0 and 1 and that instances with missing values do not have advantage over complete instances.

3 RISE's algorithm

This section is devoted to explain the algorithm of the RISE classifier, both for training and for classification of new instances. It is not our intention to fully cover all the details of RISE, since these are much more thoroughly described in the original paper by Domingos. We would rather give a very brief overview of the general algorithm and then focus on the points that we feel that are not entirely clear in RISE's paper.

3.1 Training

Fig. 1 shows the *Most Specific Generalization* routine foreshadowed in the previous section. This routine is a key piece in the RISE's training algorithm, outlined in Fig. 2. Notice that the algorithm works in a Hill-Climbing fashion, trying to increase the accuracy in each iteration of the main loop. However, observe that if we were trying to increase the accuracy in the conventional sense, the algorithm would be able to generalize very few rules. In effect, the accuracy would be 100% from the very begining, and only generalizations that have no impact on the accuracy whatsoever would be accepted). Therefore, the accuracy is computed classifying each instance leaving out the nearest

rule that covers it, provided that it does not cover any other instance (i.e. if the rule has not been generalized).

It is interesting to notice that the algorithm shown here explicitly handles duplicate rules, leaving just one occurrence of them in the rule base. In our implementation, we ensure this invariant storing the rules in a *hash set*. To allow this, we had to define a custom hash for the rules. This proved to be very effective from a maintanability point of view. Although it has not been tested, it is possible that the algorithm's efficiency also benefits from this decision since rules can be found in constant time (in average), unlike in vectors and lists.

Algorithm 1: RISE algorithm: most specific generalization procedure

Input : $R = (A_1, A_2, \cdots, A_a, C_R), \quad E = (e_1, e_2, \cdots, e_a, C_E)$
Output: Most specific rule that covers E and instances that were already covered by R

```
1 Function MostSpecificGeneralization(R, E)
2     R_msg ← R
3     for i ∈ [1, a] do
4         if R.A_i = nil then continue
5         else if is_symbolic(i) ∧ E.e_i ≠ R.A_i then R_msg.A_i ← nil
6         else if is_numeric(i) ∧ E.e_i > R.A_i.up then R_msg.A_i.up ← E.e_i
7         else if is_numeric(i) ∧ E.e_i < R.A_i.lo then R_msg.A_i.lo ← E.e_i
8     return R_msg
```

Algorithm 2: RISE training procedure

Input : ES: training data
Output: Trained rule set

```
1  Function RISEtrain()
2      RS ← ES
3      acc ← AccLOO(RS, ES)   /* Accuracy with Leave One Out (when classifying an
        instance, we leave out the rules that cover exclusively this instance) */
4      acc_increase ← true
5      while acc_increase ∨ new_rule do
6          acc_increase ← false
7          for R ∈ RS do
8              E ← NearestInstance(R, ES)   /* Nearest instance from the data set
                that is not covered by this rule and has the same class */
9              R' ← MostSpecificGeneralization(R, E)
10             RS' ← RS with R replaced by R'
11             if AccLOO(RS', ES) ≥ acc then
12                 /* In the implementation the accuracy is computed incrementally
                   to avoid the overhead of having to match all the instances with
                   all the rules */
13                 acc ← AccLOO(RS', ES) acc_increase ← true
14                 RS ← unique(RS')   /* unique removes duplicate rules */
```

3.2 Classification

The basic strategy for classifying a new instance is to find the nearest rule in the rule base and output the consequent of this rule. However, it can happen that several rules are at the minimum distance from the instance. In particular, this happens frequently when several rules cover the same instance (the distance is 0). In this case, we break the tie choosing the rule that has the highest *score*. In the original paper, the score of a rule is basically the precision of that rule. We, however, have opted for the harmonic mean between the coverage and the precision of the rule, since it is harder to have further ties and because this score combines two of the most important

quality measures of this rule into one. We do not consider having more ties (if they happen, they are broken arbitrarily). The original algorithm proposes to use the majority class of the tied rules.

4 Experiments and discussion

We have performed two kind of experiments:

- First, we have trained our classifier against entire data sets from the UCI repository and output their rules. We show the resulting rule bases here and comment on them, in order to qualitatively assess the performance of the planner.
- Then we evaluate our implementation of RISE against several data sets. We provide the accuracty and the training time, averaged via 3-Fold Cross validation. This is, the data is shuffled, and 3 partitions are created. RISE is trained 3 times, using one of the folds as a test set and the remaining folds as a training set. The accuracy and the training time is obtained and averaged. The mean and the standard deviation are reported here. We put into context our results with those of the paper.

We use the data sets described in Table 1. Each of these data sets can be found among the files distributed with our implementation, already prepared for being used.

Table 1: Data sets used for testing our RISE implementation

Data set	**Records**	**Nom. Attr.**	**Num. Attr**	**Classes**	**Missing(%)**
Audiology	226	69	0	24	2
Credit	690	9	6	2	0.6
Hepatitits	155	13	6	2	5.7
Votes	435	16	0	2	5.6
Iris	150	0	4	3	0
Chess	3196	36	0	2	0
Lenses	24	4	0	3	0
Soybean	47	35	0	4	0
Splices	3190	60	0	3	0
Zoo	101	16	1	7	0

4.1 Assessing the generated rule sets

We have chosen the *Iris* and the *Contact Lenses* data sets to show the qualitative performance of the algorithm. The reason is that the first of these two data sets contain exclusively numerical attributes, while the second has only nominal attributes. Moreover, the resulting rules are compact enough to be interpretable and provide a good illustration of the algorithm.

This is an excerpt of the rule base obtained for the *Iris* data set:

```
26 rules:
5.8<=sepal-length<=7.7, 2.5<=sepal-width<=3.6, 4.9<=petal-length<=6.9,
    1.8<=petal-width<=2.5 -> Iris-virginica (coverage: 78%, precision:
    100%, f1: 0.876404)
5.6<=sepal-length<=7.7, 2.6<=sepal-width<=3.6, 4.9<=petal-length<=6.9,
    1.8<=petal-width<=2.5 -> Iris-virginica (coverage: 76%, precision:
    100%, f1: 0.863636)
5.8<=sepal-length<=7.9, 2.6<=sepal-width<=3.8, 4.9<=petal-length<=6.9,
    1.8<=petal-width<=2.5 -> Iris-virginica (coverage: 78%, precision:
    100%, f1: 0.876404)
5.7<=sepal-length<=7.7, 2.5<=sepal-width<=3.6, 4.9<=petal-length<=6.9,
    1.8<=petal-width<=2.5 -> Iris-virginica (coverage: 80%, precision:
    100%, f1: 0.888889)
(...)
```

```
6<=sepal-length<=6.6, 2.2<=sepal-width<=2.9, 4<=petal-length<=4.6, 1<=
    petal-width<=1.3 -> Iris-versicolor (coverage: 12%, precision:
    100%, f1: 0.214286)
4.3<=sepal-length<=5.8, 2.3<=sepal-width<=4.4, 1<=petal-length<=1.9,
    0.1<=petal-width<=0.6 -> Iris-setosa (coverage: 100%, precision:
    100%, f1: 1)
```

In this case, the algorithm started with 150 rules (as many as instances in the data set) and managed to generalize them until reaching 26 rules. We believe this to be a very good result, specially since the resulting rule base is very compact and interpretable. Clearly, the rules consist of a conjunction of inequalities, since the attributes all are real (except the class, of course). Moreover, the coverage, precision and F1 score (harmonic mean of the coverage and the precision) of each rule is shown. We believe that the F1 score is a very good proxy of the quality of a rule, and this is the reason that led us to use this quantity to break ties when there are several candidate rules. One of the most interesting rules, in our view, is the last one. The algorithm is capable of finding a rule that covers all the instances of a given class with 100% of precision, which is a quite remarkable feat.

This is the resulting set of rules obtained from the *Contact Lenses* data set:

```
4 rules:
 -> 3 (coverage: 100%, precision: 62.5%, f1: 0.769231)
spectacle_prescription=1, astigmatic=2, tear_production_rate=2 -> 1 (
    coverage: 75%, precision: 100%, f1: 0.857143)
astigmatic=1, tear_production_rate=2 -> 2 (coverage: 100%, precision:
    83.3333%, f1: 0.909091)
age=1, spectacle_prescription=2, astigmatic=2, tear_production_rate=2
    -> 1 (coverage: 25%, precision: 100%, f1: 0.4)
```

To obtain this rule base, we have employed the SVDM(1) distance. Contrarily to the former case, now the conditions are equalities since we are dealing with categories rather than real attributes. The first rule is quite interesting: it has an empty antecedent (hence the full 100% coverage) and the consequent is the majority class (hence the high precision). This can be interpreted as a fall-back rule when more specialized rules with higher F1 score do not cover a new instance. In this sense, notice that when a rule with an empty antecedent appears, any rule with a lower F1 score can be throwed away with no effect whatsoever, since it will never win any instance. This is the case of the last rule, that even having a 100% of precision, will never be able to win against the first rule. The RISE algorithm, however, is not able to detect such event. Of course, it is worth stating that there is a huge room for improvement and experimenting with different score measures in addition to the precision and the F1 score.

4.2 Evaluating the time and accuracy of the algorithm

All the data sets have been splitted using 3-fold cross validation, and the resulting accuracy and training time have been measured and averaged across the different folds, using different configurations (namely: Godel, SVDM(1), SVDM(2) and KL). The quantitative results of our experiments are shown in Table 2, and are compared to those presented by Domingos.

We think that, in general, our results are competitive with those in the literature. Our implementation is able to win in some domains, and even if it does not, it comes very close to the original RISE algorithm. We have not compared the elapsed time because of the technology gap between nowadays and the time when the original paper was released. However, we have provided the elapsed time during training to give an idea of how much is required to train a RISE classifier. We can see that when the size of the data set grows moderately high (e.g. *Chess* and *Splices*) the amount of time required for training starts experiences a sudden increase. Moreover, these two are data sets with a large number of attributes (36 and 60, respectively).

Another aspect that is worth highlighting is that, in most of the domains, the SVDM and the KL metrics are superior to Gödel (only in *Zoo* it is able to tie with KL). One of the most spectacular improvements of SVDM/KL with respect to Gödel occurs in the *Spice* data set, jumping from around a 70% to over 90%.

Table 2: Evaluation of the RISE algorithm in the different data sets. The first column shows the average time spent training the classifier. The next 4 column show the accuracy under different nominal distance settings (except for IRIS, that does not have nominal values). The last column shows the accuracy results presented by Domingos in his original paper.

Dataset	Elapsed(s)	GODEL	SVDM(1)	SVDM(2)	KL	Domingos
Audiology	1.06 (± 0.09)	78.75 (± 2.28)	**79.66 (± 1.53)**	78.76 (± 0.13)	78.75 (± 2.28)	77.0 (± 5.3)
Credit	1.67 (± 0.26)	80.14 (± 4.81)	81.01 (± 3.64)	80.58 (± 3.30)	81.59 (± 3.22)	**83.3 (± 2.4)**
Hepatitis	0.12 (± 0.02)	81.21 (± 4.25)	83.15 (± 5.05)	83.18 (± 3.51)	**84.44 (± 4.95)**	78.3 (± 5.7)
Votes	0.30(± 0.02)	91.49 (± 2.53)	91.72 (± 2.25)	91.72 (± 2.25)	91.72 (± 2.25)	**95.2 (± 1.5)**
Iris	0.02 (± 0.003)	93.33 (± 3.40)				**94.0 (± 2.8)**
Chess	161.74 (± 47.63)	89.74 (± 0.16)	95.78 (± 0.60)	95.28 (± 0.42)	95.28 (± 0.423)	**98.2 (± 0.6)**
Lenses	0.0008 (± 0.0004)	**83.33 (± 5.89)**	**83.33 (± 5.90)**	**83.33 (± 5.89)**	**83.33 (± 5.89)**	77.2 (± 12.8)
Soybean	0.01 (± 0.001)	**100.00 (± 0.00)**	**100.00 (± 0.00)**	**100.00 (± 0.00)**	**100.00 (± 0.00)**	**100.0 (± 0.0)**
Splices	67.79 (± 15.16)	70.66 (± 0.53)	93.48 (± 0.42)	**94.39 (± 0.16)**	89.72 (± 4.34)	93.1 (±1.6)
Zoo	0.03 (± 0.005)	**94.05 (± 4.95)**	90.2453 (± 5.78051)	92.2655 (± 7.09792)	**94.05 (± 2.48)**	93.9 (± 3.7)

5 Practical details about the implementation

The implementation has been developed in C++. We use the C++11 standard, so it is necessary a compiler that supports it (we have used GCC 5.4). The application is self-contained: there is no need of 3rd party libraries to build everything. In order to compile the library and the binaries it is enough to execute `make all` from the implementation base folder:

```
$ make -j4
```

This will create a `build` directory with the compiled objects, the shared library and all the executables. The most important binary is the one called `./build/rise_classifier` (the other ones are modular tests). It should be always executed from the `build` folder so it can find the data sets (the path is coded relative to the executable location as `../Data`). The usage of the binary is as follows:

```
./rise_classifier
Usage: rise_classifier datasetname {godel|svdm|kl} [q] #folds
```

The first argument of the program must be one of the data sets in the `Data` folder. The second argument is the type of distance that is considered between nominal values. The next argument should be the q parameter of the SVDM distance if `svdm` is the chosen distance (this is ommited otherwise). The number of folds to train and test with k-fold cross validation. If the number of folds is 1, the whole data set if used for training (there is no testing phase), and the rule base is output to the screen. Moreover, during the execution of the algorithm, there is periodic feedback reporting the evolution of the rule set. Examples of calls:

```
./rise_classifier crx godel 10 # run on the Credit Approval data set
                               # with Godel distance and 10 folds.
./rise_classifier lenses svdm 2 1 # run on the C. Lenses data set using
                                  # SVDM(2) distance. Output rules.
./rise_classifier hepatitis kl 1 # run on the Hepatitis data set using
                                 # the KL distance. Output rules
```

The implementation, this report and other additional files have been made available via a repository[1].

References

[Dom96] Pedro Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24(2):141–168, 1996.

[1] https://github.com/sprkrd/rise-implementation