Methods for Rule Conflict Resolution

Tony Lindgren

Department of Computer and Systems Sciences,
Stockholm University and Royal Institute of Technology,
Forum 100,
164 40 Kista, Sweden
tony@dsv.su.se

Abstract. When using unordered rule sets, conflicts can arise between the rules, i.e., two or more rules cover the same example but predict different classes. This paper gives a survey of methods used to solve this type of conflict and introduces a novel method called Recursive Induction. In total nine methods for resolving rule conflicts are scrutinised. The methods are explained in detail, compared and evaluated empirically on an number of domains. The results show that Recursive Induction outperforms all previously used methods.

1 Introduction

Two major strategies are used to induce rules in machine learning systems: Divide-And-Conquer (DAC) [9] and Separate-And-Conquer (SAC) [4]. These two strategies induce rules with different characteristics. The main difference is that rules generated by DAC cannot overlap while rules generated by SAC can. Below follows an informal description of DAC and SAC that explains this difference.

DAC divides an overly general rule on an attribute, creating as many new specialised rules (nodes) as there are values for the attribute (in the discrete case). Learning stops when a specific stopping criterion is met (e.g., class purity). A node that has met the stopping criterion is said to be a leaf node. When all nodes have met the stopping criterion the algorithm is finished. The product of DAC induction can be visualised as an upside down tree (hence its other common name, decision tree induction) with the class labels on the end of the branches (the leaf nodes). Each possible path from the root to a leaf can be transformed into a rule or be directly used for classification.

SAC does not divide an attribute on its values exhaustively but rather separates the rule by specialising/generalising the rule with one attribute-value pair at a time (depending if it starts with a overly general rule or an overly specific rule). The stopping criterion is then checked, if it is met, the examples covered by the rule are separated out of the training set. The algorithm then starts over again with an overly general/specific rule to cover the examples left in the training set. This continues until the training set is empty, i.e., all examples have been covered. The induction strategy that SAC uses can lead to rules that cover the same examples. The rules can be ordered in a hierarchy to avoid conflicts (which results in so called decision lists [12]) or they may be used without any ordering [2, 3].

After inducing rules using SAC, when using these rules for classification, the algorithm may be faced with the problem of having two (or more) rules covering the same example while having different majority classes. This type of conflict must be handled by the classification algorithm. How this type of conflict can be handled is the focus of this work. The problem of solving rule conflicts has not attracted much interest even though there can be significant gains in accuracy made from choosing the right method for solving rule conflicts.

This paper gives a survey of different methods that have been used to solve the problem of rule conflicts in the past, and also introduces a novel method for solving rule conflicts called Recursive Induction. The different methods are compared empirically on a number of domains. It is shown that Recursive Induction outperforms the previously used methods.

The paper is organised as follows. In section 2, a review of the different methods is given, including the novel method Recursive Induction. In section 3, the different methods are compared empirically and the results are presented and discussed. Finally, in section 4, conclusions are made and possible future work is outlined.

2 Different ways of resolving classification conflicts

In this section, different methods to solve rule conflicts are described. The workings of each method will be illustrated on a hypothetical scenario presented in Figure 1.

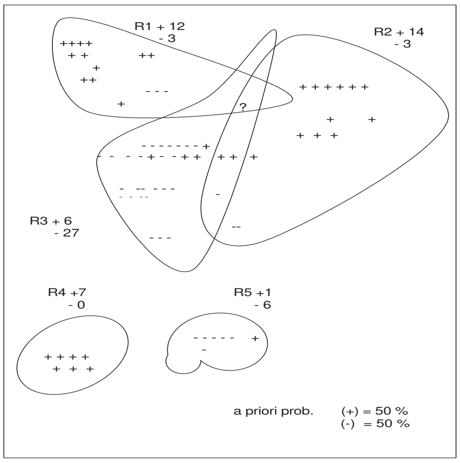

Fig. 1. Three rules covering an example to be classified (marked with '?'). The training examples are labelled with their respective classes (+ and −).

2.1 CN2

The system CN2 [2] resolves classification conflicts between rules in the following way. Given the examples in Figure 1, the class frequencies of the rules that cover the example to be classified (marked with '?') are calculated:

$$C(+) = covers(R_1, +) + covers(R_2, +) + covers(R_3, +) = 32$$
$$C(-) = covers(R_1, -) + covers(R_2, -) + covers(R_3, -) = 33$$

where $covers(R, C)$ gives the number of examples of class C that are covered by Rule R. This means that CN2 would classify the example as belonging to the negative class $(-)$. More generally:

$$CN2 = argmax_{C_i \in Classes} \sum_{j=1}^{|CovRules|} covers(R_j, C_i)$$

where *CovRules* is the set of rules that cover the example to be classified, and *covers* is the function defined above.

2.2 Naive Bayes classification

In the machine learning system of Rule Discovery System (RDS) [10], naive Bayes is used to resolve rule conflicts. Bayes theorem is as follows:

$$P(C|R_1 \wedge \ldots \wedge R_n) = P(C)\frac{P(R_1 \wedge \ldots \wedge R_n|C)}{P(R_1 \wedge \ldots \wedge R_n)}$$

where C is a class label for the example to be classified and $R_1 \ldots R_n$ are the rules that cover the example. As usual, since $P(R_1 \wedge \ldots \wedge R_n)$ does not affect the relative order of different hypotheses according to probability, it is ignored. Assuming (naively) that $P(R_1 \wedge \ldots \wedge R_n|C) = P(R_1|C) \ldots P(R_n|C)$, the maximum a posteriori probable hypothesis (MAP) is:

$$h_{MAP} = argmax_{C_i \in Classes} P(C_i) \prod_{Rj \in Rules}^{|CovRules|} P(R_j|C_i)$$

where *CovRules* is the set of rules that covers the example to be classified. If we again consider the example shown in Figure 1, we get:

$$P(+|R_1 \wedge R_2 \wedge R_3) = P(+) * P(R_1|+) * P(R_2|+) * P(R_3|+) =$$
$$40/80 * 12/40 * 14/40 * 6/40 = 0.0079$$
$$P(-|R_1 \wedge R_2 \wedge R_3) = P(-) * P(R_1|-) * P(R_2|-) * P(R_3|-) =$$
$$40/80 * 3/40 * 3/40 * 27/40 = 0.0019$$

This means that the naive Bayes classification results in that the example with the unknown class label is classified as positive (+).

Note that if a rule involved in a conflict does not cover any examples of a particular class, this would eliminate the chances for that class to be selected, even if there are several other rules that cover the example with a high probability for that class. To overcome this problem, Laplace-1 correction (described in [6]) is used in the experiments.

2.3 C5.0/See5

The machine learning system C5.0/See5 produced by RuleQuest Research [11] uses a voting method to solve rule conflicts. Each applicable rule votes for its predicted class with a weight equal to the probability of the given class, the votes are summed up, and the class with the highest probability is chosen as the final prediction.

$$See5(voting) = argmax_{C_i \in Classes} \sum_{j=1}^{|CovRules|} \begin{cases} P(C_i|R_j) \; if \; P = max \; P \; for \; R_j \\ 0 \qquad\qquad otherwise \end{cases}$$

Here *CovRules* is the set of rules that covers the example to be classified. In the tutorial of See5 [11] another method for resolving rule conflicts is mentioned, and that uses the rule with the highest probability (confidence) of all the rules in conflict.

$$See5(max\,prob) = argmax_{C_i \in Classes} P(C_i|R_j \in CovRules)$$

Both these methods are used in our experiments. Applying these methods to the situation in Figure 1 leads to the following:

$$See5(voting) = P(+|R_1) + P(+|R_2) + P(+|R_3) = 12/40 + 14/40 + 0 = 26/40$$

$$See5(voting) = P(-|R_1) + P(-|R_2) + P(-|R_3) = 0 + 0 + 27/40 = 27/40$$

$$See5(max\,prob) = P(+|R_1) = 12/40, \;\; See5(max\,prob) = P(-|R_1) = 3/40$$

$$See5(max\,prob) = P(+|R_2) = 14/40, \;\; See5(max\,prob) = P(-|R_2) = 3/40$$

$$See5(max\,prob) = P(+|R_3) = 6/40, \;\; See5(max\,prob) = P(-|R_3) = 27/40$$

Here we see that both methods would assign the negative (−) class to the example.

2.4 Intersecting Rules

The idea behind Intersecting Rules [7] is that one should use the information in the intersection of the conflicting rules. If there are not any training examples in the intersection of the examples in conflict, Intersecting rules partition these rules in as few non-empty partitions as possible and uses these partitions to find the most probable class.

If we would use Intersecting Rules on the example scenario in Figure 1 where no training example is present in the intersection of these three rules, the partitions would be: $R_2, R_3\, and\, R_1$ as the intersection between R_2 and R_3 is non-empty. This gives the following probabilities:

$$P(+|R_1 \wedge R_2 \wedge R_3) = P(+) * P(R_2 \wedge R_3|+) * P(R_1|+) =$$

$$40/80 * 2/40 * 12/39 = 0.00769$$

$$P(-|R_1 \wedge R_2 \wedge R_3) = P(-) * P(R_2 \wedge R_3|-) * P(R_1|-) =$$

$$40/80 * 1/40 * 3/39 = 0.000962$$

Here Intersecting Rules would assign the positive (+) class to the example. Consider again the scenario in the Figure 1 with the exception that training examples do reside in the intersection between R_1 and R_3. In this case two possible partitions are possible. The previously used partition and $R_1, R_3 and R_2$. In this paper two different methods are used to handle the problem of multiple partitions. The first method computes the joint class-probability of the partitions and selects the class with the highest probability. The second method computes the maximum class-probability for each partition and selects the class with the highest probability, and hence selects one partition (dependency model). The latter method was used in [7], while the former is new.

It is worth noting two things about Intersecting Rules. First, if indeed there are examples in the intersection, Intersecting Rules is really just using Bayes rule to compute the most probable class. If it is necessary to partition the rules, Intersecting Rules is essentially moving from an assumption of dependence between the rules towards independence. Hence if no non-empty partitions are found at all, the method degenerates to the naive Bayes classifier.

2.5 Double Induction

The idea of Double Induction [8] is to induce new rules based on the examples that are covered by the rules in conflict. By doing so, a completely fresh set of rules is obtained, tailor-made to separate the examples in this subset of the whole domain. By concentrating on a small subspace of the example space there is a higher chance to find rules that separate the classes better. These new rules are then used to classify the unlabelled example, either with naive Bayes (if there is a conflict) or directly. The Double Induction algorithm is given in Table 1.

Table 1. The Double Induction algorithm.

```
Input: R_1 = rules from the first induction round, e = example to be
classified, E_1 = training examples
Output: C = a class assigned to e

collect all rules R_{1,e} ⊆ R_1 that cover e
if conflictingRules(R_{1,e}) then
   collect all training examples E_2 ⊆ E_1 covered by R_{1,e}
   induce new rules R_2 from E_2
   collect all rules R_{2,e} ⊆ R_2 that cover e
   if conflictingRules(R_{2,e}) then
      let C = naiveBayes(R_{2,e},E_2)
   else let C = majorityClass(R_{2,e})
else let C = majorityClass(R_{1,e})
```

2.6 Recursive Induction

Recursive Induction extends Double Induction in the following way: When inducing new rules from the examples covered by the conflicting rules, if these newly induced rules have rule conflicts between them Recursive Induction does yet another induction, using the examples covered by the rules of this "new" rule conflict. Two different stopping conditions are used by Recursive Induction. The first stops learning if the induction algorithm cannot find rules that differ from the rules that were previously induced. The second stopping condition uses the naive Bayes most probable class value as a quality measure of the rules induced. If the value for the most probable class decreases from one induction round to the next, the learning is stopped. In our experiments, we use two different versions of Recursive Induction, one which only uses the first stopping condition and another version which utilises both stopping conditions. If any of theses conditions hold, then the algorithm stops and naive Bayes is used to solve the conflict (using the previous set of rules). Otherwise new rules are induced recursively until no conflict is present in the rule set. The Recursive Induction algorithm is given in Table 2.

A few things are worth noting: The number of examples used to induce new rules is always decreasing or constant but is never increasing. So iteratively the example-space that the induction algorithm has to cope with shrinks. When inducing rules, the examples are randomly divided into two halves in order to create a grow set (to induce rules) and a prune set (to prune rules). In Recursive Induction and Double Induction this stochastic selection of examples helps the induction algorithm to avoid getting stuck, i.e., inducing exactly the same set of rules as before.

In [7] it is empirically shown that it is worthwhile to weight the examples in the intersection as more important than examples outside the intersection, for a

Table 2. The Recursive Induction algorithm.

```
Input: R_n = rules from the n:th induction round, e = example to be
classified, E_n = training examples
Output: C = a class assigned to e


collect all rules R_{n,e} ⊆ R_n that cover e
  if conflictingRules(R_{n,e}) then
    collect all training examples E_{n'} ⊆ E_n covered by R_{n,e}
    induce new rules R_{n'} from E_{n'}
    collect all rules R_{n',e} ⊆ R_{n'} that cover e
    if R_{n,e} same R_{n',e} (or nBquality(R_{n,e},E_n) > nBquality(R_{n',e},E'_n)) then
      let C = naiveBayes(R_{n,e},E_n)
    else call Recursive Induction with R_{n'}, e and E_{n'}
  else let C = majorityClass(R_{n,e})
```

correct classification (after all, Bayes theorem only considers the evidence in the intersection). This can be done by a concatenation of all the covered examples of every rule, in which some examples may be present more than once (i.e., a multi-set). An example is present in the multi-set as many times as it is covered by different rules and hence weighted as so many times more important than those examples covered by only one rule. When using this weighting scheme, the induction algorithm focuses on separating the examples located around the example that is to be classified. This weighting scheme is also used in Double Induction.

If the Recursive Induction gets stuck, naive Bayes is used to solve the conflict. The a priori probability is then computed from the examples covered by the previously conflicting rules. This means that the a priori probability reflects the probability distribution of this example-space (contained by the rules in conflict), rather than the a priori probability of the whole domain. This is also the case for Double Induction when using naive Bayes.

Consider again the scenario shown in Figure 1. Given the examples covered by R_1, R_2 and R_3, Separate-and-Conquer may come up with the three new rules shown in Figure 2. The unlabelled example is then classified using these newly induced rules. In our hypothetical scenario, the example is covered by R_6 resulting in that the positive class (+) is assigned to the example.

3 Empirical evaluation

3.1 Experimental setting

The search strategy employed in the experiments is Separate-and-Conquer together with information gain to greedily choose what condition to add when

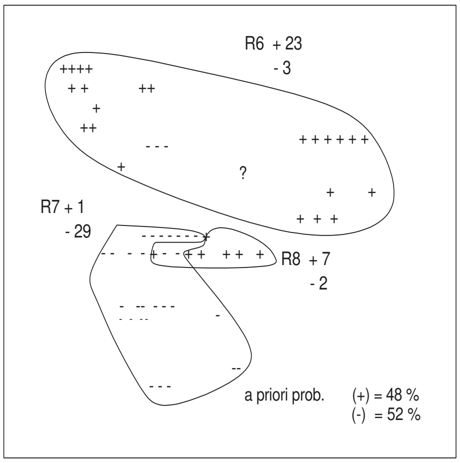

Fig. 2. Three new induced rules.

refining a rule. One half of the training set was used as grow set and the second half was used as pruning set. The rules were pruned using Incremental Reduced Error Pruning (IREP) [5]. These settings were used in all inductions of rules, i.e., both when inducing the initial set of rules and when resolving rule conflicts with Recursive Induction and Double Induction.

The experimental results were obtained by ten-fold cross-validation, with exactly the same rule conflicts to solve for the different methods. All datasets used were taken from the UCI Machine Learning Repository except the King-Rook-King-Illegal (KRKI) database which comes from the Machine Learning group at the University of York, and Titanic dataset which comes from the Delve datasets at the University of Toronto. The datasets used in the experiments are all the possible datasets (in total forty domains) that we could get hold of and run (in reasonable time), and hence use in the experiment. There is quite a diversity of characteristics in the datasets used; from 2 class problems to problems with 24 classes, from datasets with 32 examples to 12960 examples. This diversity will hopefully reveal the behaviour of the different rule conflict resolving methods.

To minimise the amount of work within Recursive Induction, Double Induction and Intersecting Rules, the conflicts already seen (for a particular fold) are saved to allow reuse of the generated rules in the second round whenever the same conflict is observed again (within the same fold).

3.2 Experimental results

All rule conflict resolution methods were tested on forty domains. In nine of these domains there was no conflict between the rules. The domains without conflict were: Bupa liver-disorders, Ionosphere, Sonar, Pima indians diabetes, Glass, Spambase, SPECTF heart, Post operation patients and Haberman's survival data. The accuracies for the different methods are shown in Table 3. In the first column the domain is denoted, then follows the accuracies for the different methods. The methods are (from left to right): Recursive Induction using naive Bayes as stopping criterion (RI-s), Recursive Induction (RI), Double Induction (DI), Intersecting Rules using highest class-probability (IR), Intersecting Rules using joint highest class-probability (IR-j), naive Bayes (Bayes), Frequency-based classification (CN2), See5 using voting (See5-v) and See5 using highest probability (See5-hp). If there exists a single method that achieves higher accuracy than all other methods for a domain, then this method's accuracy is denoted by bold typesetting in Table 3.

Table 3. Accuracies of the different rule conflict resolving methods.

Domain	**RI-s**	**RI**	**DI**	**IR**	**IR-j**	**Bayes**	**CN2**	**See5-v**	**See5-hp**
Balance scale	82.07	**83.13**	82.43	82.95	79.96	81.90	82.25	81.55	77.15
Car evaluation	82.75	**84.47**	83.70	80.84	80.84	79.69	78.23	82.62	81.67
TicTacToe	98.28	98.39	98.39	98.39	98.28	97.93	85.65	94.49	98.28
Breast cancer	70.38	70.00	70.77	72.69	72.69	72.31	73.46	71.92	69.62
Dermatology	**88.59**	88.29	87.99	85.29	84.98	85.59	83.48	86.49	86.49
C. Voting	94.19	**95.20**	94.95	94.95	94.70	94.44	94.70	94.95	94.95
KRKI	99.34	99.34	99.23	98.68	98.68	98.02	94.84	98.24	99.12
New thyroid	89.29	89.29	89.29	88.78	88.78	88.78	88.78	88.78	88.78
Lymphograpy	81.48	82.22	82.22	79.26	79.26	80.00	77.78	79.26	80.74
Primary tumor	39.48	**42.72**	42.39	40.13	39.81	37.22	35.60	40.78	41.42
Shuttle	98.02	98.02	98.02	96.05	96.05	93.68	93.68	94.47	94.47
Credit	83.92	**85.98**	85.19	83.12	83.60	83.44	82.96	80.89	85.03
Ecoli	80.39	**81.70**	80.72	80.39	80.39	80.07	77.45	80.39	80.39
Hepatitis	80.85	80.85	79.43	79.43	79.43	79.43	79.43	80.85	80.85
Iris	90.51	90.51	90.51	90.51	90.51	90.51	88.32	88.32	90.51
Nursery	80.31	**81.09**	80.65	79.96	79.88	79.87	75.59	79.61	80.49
C. bands	76.78	**77.80**	76.99	73.12	73.52	73.93	69.45	72.10	72.51
Cl. heart	59.06	**59.42**	59.06	57.61	57.61	57.25	57.25	57.97	58.33
Image segmentation	71.73	71.73	71.73	69.11	69.11	69.11	69.11	68.59	67.54
Wine	86.42	86.42	86.42	87.04	87.04	87.04	86.42	86.42	86.42
Audiology	66.48	67.58	67.58	65.38	65.38	59.89	59.89	65.38	65.38
B. c. Wisconsin	93.43	93.43	93.30	93.03	93.03	93.30	92.90	93.16	92.76
KR vs KP	99.24	**99.35**	99.21	99.24	99.24	98.49	95.83	98.69	98.83
Lung cancer	55.17	55.17	**58.62**	51.72	51.72	51.72	51.72	51.72	55.17
Mushroom	100.00	100.00	100.00	100.00	100.00	100.00	98.46	100.00	100.00
Promoters	79.80	78.79	78.79	74.75	73.74	78.79	75.76	79.80	79.80
Sick euthyroid	95.34	**96.42**	95.48	94.37	94.47	92.11	91.90	94.02	92.84
Soybean large	76.07	**78.21**	74.29	63.21	63.21	61.79	60.00	61.43	60.36
Tae	49.64	**56.20**	51.20	44.53	45.26	43.80	43.80	41.61	40.88
Zoo	91.30	91.30	91.30	91.30	91.30	91.30	91.30	89.13	89.13
Titanic	78.81	78.91	78.86	79.11	79.11	78.61	68.32	78.91	78.86

Table 4 shows the significant differences for each method compared with the other methods. Each line in the table shows the number of wins and losses for the method compared to other methods (the methods' names in top of the column). An asterix (*) denotes that the amount of wins and losses is statically significant. The p-value of significance is set to 0.05, the p-value computed with an exact version of McNemar's test. This test was also originally used to compute the statistical significance of the accuracies.

The methods in Table 4 can be divided into three sets with roughly equal performance. The first group contains Recursive Induction, Double Induction and Recursive Induction using naive Bayes. The second group is Intersecting Rules, Intersecting Rules using joint probability, naive Bayes, See5-voting and See5 using highest probability. In the last group CN2 resides.

The first group includes methods which all outperform the rest of the methods. They all win significantly more times than they loose. Within this group RI is best, beating RI-s with 4–0 and DI with 4–0. Second in the group is DI which beats RI-s with 2–0. All methods in the second group beat CN2 significantly. But internally no significant differences are present in the group. The methods in the second group could be ranked in the following way according to their performance. Best is both Intersecting Rules methods, then comes both See5 methods followed by naive Bayes. CN2 shows to perform significantly worse than all other methods and is in a group of its own.

Table 4. Result of McNemar's test comparing wins and losses for all domains with a significant difference

	RI-s	**RI**	**DI**	**IR**	**IR-j**	**Bayes**	**CN2**	**See5-v**	**See5-hp**
RI-s	-	0,4	0,2	7,0*	7,0*	9,0*	14,0*	10,0*	7,0*
RI	4,0	-	4,0	10,0*	10,0*	12,0*	16,0*	12,0*	10,0*
DI	2,0	0,4	-	6,0*	7,0*	10,0*	14,0*	10,0*	9,0*
IR	0,7*	0,10*	0,6*	-	2,0	5,0	9,0*	3,1	4,1
IR-j	0,7*	0,10*	0,7*	0,2	-	4,0	9,0*	4,1	4,1
Bayes	0,9*	0,12*	0,10*	0,5	0,4	-	11,0*	3,3	2,6
CN2	0,14*	0,16*	0,14*	0,9*	0,9*	0,11*	-	1,11*	2,11*
See5-v	0,10*	0,12*	0,10*	1,3	1,4	3,3	11,1*	-	4,4
See5-hp	0,7*	0,10*	0,9*	1,4	1,4	6,2	11,2*	4,4	-

Table 5 compares all wins and losses, not just the significant wins and losses as Table 4. Otherwise the table has a similar structure. An asterix (*) denotes that the amount of wins and losses is statically significant. The p-value of significance is set to 0.05, the p-value computed with an exact version of McNemar's test. The results from Table 4 are amplified in Table 5.

Recursive Induction is significantly better than any other method. Then comes Double Induction and Recursive Induction with stop, between them their is no significant difference, even tough DI wins with 16–7. Both methods are significantly better than all other methods except RI. Intersecting Rules is significantly better than naive Bayes, See5 and CN2. Intersecting Rules using joint probability comes next followed by See5-highest probability, See5-voting, naive Bayes and CN2. All methods are significantly better than CN2.

Table 5. Result of McNemar's test comparing wins and losses for all domains

	RI-s	RI	DI	IR	IR-j	Bayes	CN2	See5-v	See5-hp
RI-s	-	4,15*	7,16	18,7*	21,7*	25,3*	26,3*	20,5*	18,5*
RI	15,4*	-	18,2*	24,4*	26,3*	26,1*	28,1*	24,3*	25,1*
DI	16,7	2,18*	-	19,5*	24,3*	22,3*	26,1*	23,4*	26,2*
IR	7,18*	4,24*	5,19*	-	6,4	17,6*	24,2*	17,6*	15,11
IR-j	7,21*	3,26*	3,24*	4,6	-	16,7	21,3*	16,8	13,12
Bayes	3,25*	1,26*	3,22*	6,17*	7,16	-	18,3*	13,15	9,18
CN2	3,26*	1,28*	1,26*	2,24*	3,21*	3,18*	-	6,21*	5,24*
See5-v	5,20*	3,24*	4,23*	6,17*	8,16	15,13	21,6*	-	10,10
See5-hp	5,18*	1,25*	2,26*	11,15	12,13	18,9	24,5*	10,10	-

3.3 Discussion about the results

One conclusion that can be drawn from the experiments is that the computationally expensive methods obtain higher accuracy than methods that are less computationally expensive. Both RI methods have a very high computional cost but also perform very well. The worst-case time complexity for SAC is quadratic [1] in the number of examples, and since DI uses SAC to induce new rules this holds for DI as well. This figure for RI depends on the number of recursive steps it takes and is even worse. For IR this figure is even worse in theory as the computational cost grows exponentially with the number of rules in conflict. In reality though it is faster than RI as the number of rules in conflict is often not so high compared to the number of examples covered by the rules in conflict. In fact most of the time there is no need to partition the rules at all. The rest of the methods have the same computional cost. This is also reflected in their performance with the exception of CN2, which for some reason fails miserably.

One of the reasons why DI performs better than RI-s is because the stopping criterion hinders RI-s to do any new inductions at all. This is the case when the naive Bayes value decreases compared to the originally induced rules. When this happens RI-s behaves just like naive Bayes. Still this does not happen that often as it does perform better than naive Bayes.

4 Conclusions

In this paper a survey of different rule resolving methods have been presented and compared. A novel method called Recursive Induction based on Double Induction was also presented. Recursive Induction proved to solve rule conflicts so good that it with statistical significance achieved higher accuracy than all other methods except for Double Induction. On the other side of the scale is CN2, as this method was beaten significantly by all other methods. Double Induction and Recursive Induction using naive Bayes as stopping criterion were both shown to significantly outperform all other methods except Recursive Induction. Intersection rules was shown to significantly outperform naive Bayes and See5-voting. Between the rest of the methods no significant differences were found. One of the major drawbacks of using Recursive Induction is the computional cost. However, if accuracy is of uttermost importance and quick response time is not, then it is a useful technique.

One issue that needs further investigation when it comes to Recursive Induction and Double Induction is the use of other weighting schemes than the one used in our experiments, which is quite conservative. Another issue to address is to find a more suitable stopping criterion than to use naive Bayes as a quality measure together with Recursive Induction.

References

1. H. Boström and P. Idestam-Almquist. Induction of logic programs by example-guided unfolding. *Journal of Logic Programming, Vol. 40 (2-3)*, pages 226–237, 1999.
2. P. Clark and R. Boswell. Rule induction with CN2: some recent improvements. In *Proceedings of the Fifth European Working Session on Learning*, pages 151–163, Berlin, 1991. Springer-Verlag.
3. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning, 3, 261-283*, 1989.
4. J. Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 1999.
5. J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In W.W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 70–77. Morgan Kaufmann, 1994.
6. R. Kohavi, B. Becker, and D. Sommerfield. Improving simple bayes. In *Proceedings of the European Conference on Machine Learning*, 1997.
7. T. Lindgren and H. Boström. Classification with intersecting rules. In *Proceedings of the 13th International Conference on Algorithmic Learning Theory (ALT'02)*, pages 395–402. Springer-Verlag, 2002.
8. T. Lindgren and H. Boström. Resolving rule conflicts with double induction. In *Proceedings of the 5th International Symposium on Intelligent Data Analysis*. Springer-Verlag, 2003.
9. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
10. RDS. Rule discovery system (RDS) – 1.0, Compumine AB, 2003. `www.compumine.com`.
11. RuleQuest Research. See 5: an informal tutorial, 2003. `www.rulequest.com/see5-win.html`.
12. R. Rivest. Learning decision lists. *Machine Learning, 2(3), 229-246*, 1987.