Feature Transformation through Rule Induction: A Case Study with the *k*-NN Classifier

Antal van den Bosch

ILK / Computational Linguistics and AI,
P.O. Box 90153, NL-5000 LE Tilburg, The Netherlands
Antal.vdnBosch@uvt.nl
http://ilk.uvt.nl/

Abstract. A extension to the *k*-nearest neighbor classifier is described in which automatically induced rules are used as binary features, which are active in an instance when the left-hand side of the corresponding rule matches with the instance. The RIPPER rule induction algorithm is employed to produce the rules. The similarity between a memory instance and a new instance is based on the rules the two instances share. We report on experiments that indicate that (i) the method equals the generalization performances of RIPPER and *k*-NN classification on average, and (ii) when the original multi-valued features are combined with the transformed rule-based features, some significant improvements in *k*-NN classification are observed, particularly with artificial benchmark tasks.

1 Rules as features

A well-established machine-learning solution to classification problems is rule induction [1–3]. The goal of rule induction is generally to induce a set of rules from data that captures all generalizable knowledge within that data, and that is as small as possible at the same time. Classification in rule-induction classifiers is typically based on the firing of a rule on a test instance, triggered by matching feature values at the left-hand side of the rule. Rules can be of various normal forms, and are typically ordered; with ordered rules, the first rule that fires determines the classification outcome and halts the classification process. The appropriate content and ordering of rules is hard to find in general, so at the heart of most rule induction systems strong search algorithms operate that attempt to minimize search through the space of possible rule sets and orderings.

Although rules appear quite different objects from instances as used in *k*-nearest neighbor classification or instance-based learning [4], there is a continuum between them. Rules can be seen as generalized instances; they represent a subset of training instances that match on the conditions on the left-hand side of the rule. Therefore, *k*-NN classification can naturally be applied to rules. For example, Domingos [5] describes the RISE system, in which rules are (carefully) generalised from instances, and in which the *k*-NN classification rule searches for nearest neighbors within these rules when classifying new instances.

From another point of view, sets of instances covered by different rules sometimes overlap. In other words, seen from the instance perspective, a single instance can match more than one rule. Consider the schematic example displayed in Figure 1. Three instances with three multi-valued features match individually with one or two of the four rules; for example, the first instance matches with rule 1 (if $f1 = A$ then $c = Z$) and with rule 3 (if $f2 = C$ then $c = Z$).

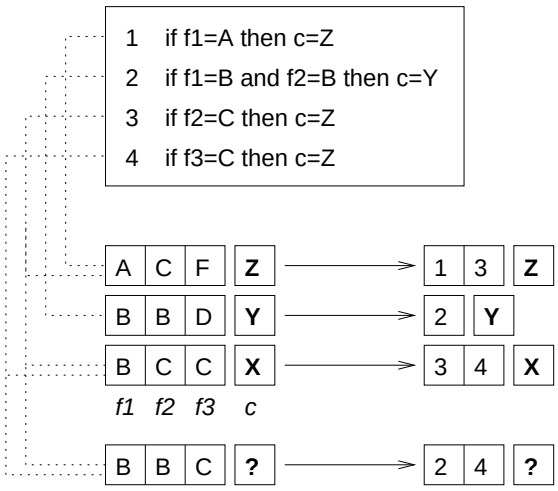

Fig. 1. Schematic visualization of the encoding of multi-valued instances via matching rules to rule-indexed instances, characterised by the numbers of the rules that match them. $f1$, $f2$, and $f3$ represent the three features. c represents the class label.

In this perspective, it is possible to index instances by the rules that apply to them. For example, in Figure 1, the first instance can be indexed by its matching rules 1 and 3. When the left-hand sides of rules are seen as potentially complex features (in which the presence of some combination of feature values is queried) that are strong predictors of a single class, indexing instances by the rules that apply to them is essentially a feature transformation method. Sébag and Schoenauer [6] describe a similar method in which they transform instances of a regression task through induced rules – the method proposed here can be seen as an extension of the work presented in [6] to classification. Sébag and Schoenauer propose to use feature transformation particularly because of its interesting dimension reduction capabilities; the dimensionality of the feature space of a multi-feature task can be reduced if a rule set can be found that is smaller in the number of rules than the original number of features.

An instance base that has been transformed in the way described above can be used again as training material for a k-NN classifier, trivially. Like the k-NN classifier trained on the instance base before the transformation, this new k-NN classifier can compute distances between transformed instances with any distance

kernel, possibly composed of kernel elements such as the Overlap metric [4], the modified value-difference metric [7], distance weighting [8], feature weighting [9], and combinations thereof. For example, a simple kernel such as the Overlap metric without feature weighting [4] would compute the similarity between two instances by summing the numbers of matching features (i.e. those features at which both instances have the same value). Trained on the transformed data, a k-NN classifier with this kernel computes the similarity between two instances essentially by counting the number of rules that fire on both instances. Subsequently, it uses this similarity to find and rank the k nearest neighbor instances to the new instance to be classified.

In Figure 1, a new instance (bottom) matches rules 2 and 4, thereby (partially) matching the second and third memory instances. If, for example, rule 4 would have a higher overall weight than rule 2, the third memory instance would become the nearest neighbor. The k-NN classification rule with $k = 1$ then would state that the class of the nearest neighbor transfers to the new instance, which would mean that class X would be copied – which is a different class than those predicted either by rule 2 or 4. This is a marked difference with classification in RIPPER, where the class is assigned directly to the new instance by the rule that fires first. It is possible that many classifications in this approach would be identical to those made by RIPPER, but the above example illustrates that k-NN can produce different classes even if the features represent the original rules. It is an empirical question whether the k-NN approach would have some consistent advantage in the cases where classification diverges.

In this paper, which extends the preliminary work focusing on natural language processing tasks reported in [10], we describe an implementation of this idea. We measure the effects of changing the representation of instances of some classification task by transforming their original features into features based on rules, which are in turn induced by a rule-induction algorithm on the same data. Before and after the transformation we learn the task with the k-NN classifier, running 10-fold cross validation experiments, and show that the approach works out promising for a sample of benchmark tasks. We find that the method works roughly equally well as RIPPER on these data sets. Also, the method does not improve on k-NN classification on the same data before the transformation. Only when the transformed features are *added* to the original multi-valued features, improvements are observed.

The paper is structured as follows. First, we describe the feature transformation process to convert multi-valued features to binary rule features, in Section 2. In Section 3 we outline the experimental setup; how we run comparative experiments with k-NN, RIPPER, and the transformed k-NN. We list the benchmark data used, and describe a classifier-wrapping-based intermediate procedure to estimate optimal algorithmic parameter settings. Subsequently we present the results of our experiments in Section 4, and reflect on the outcomes in the concluding Section 5.

2 Feature transformation through rule induction: algorithm

The feature transformation procedure described in the previous section and illustrated in Figure 1 uses the following procedure: given a training set Tr containing labeled instances of a certain classification task:

1. Apply RIPPER [3] to Tr, and collect the set of n rules induced on Tr, R_{Tr}
2. Transform Tr into Tr':
   - initialize Tr' by copying all instances from Tr to Tr' retaining only their class label;
   - `foreach` rule $r \epsilon R_{Tr}$ with indices $index_1 \ldots index_n$
     `foreach` instance $i \epsilon Tr$
     if conditions of r match i,
     add $index_r$ as an active binary feature to i' in Tr'
     `end`
     `end`

After transformation a k-NN classifier can be trained on Tr' (i.e. the transformed training set is read in the k-NN classifier's memory), and can then classify a new instances I:

1. Transform I into I':
   - initialize I' as a featureless example labeled with I's class
   - `foreach` rule $r \epsilon R_{Tr}$ with indices $index_1 \ldots index_n$
     if conditions of r match I,
     add $index_r$ as an active binary feature to I'
     `end`
2. Find the k nearest neighbors of I' in Tr' according to the classifier's similarity metric
3. Extrapolate the majority class of the k nearest neighbors to I' according to the classifier's class vote weighting method.

The choice for RIPPER [3] is arbitrary; this could have been any rule induction algorithm. RIPPER is chosen on grounds of the general observation that it is fast and produces results competitive with other rule induction algorithms. RIPPER, belonging to the family of sequential covering rule inducers, induces rules per class, in a predetermined class ordering. By default, the ordering is from low-frequency classes to high frequency classes, leaving the most frequent class as the default rule (which is generally beneficial for the total description length of the rule set). Within a class RIPPER searches for the shortest rules with the best coverage and accuracy. Central to RIPPER's heuristic algorithm is splitting the training set in two. On the basis of one part it induces a list of candidate rules. When a candidate rule classifies instances in the other part of the split training set below a threshold, and/or it is too long to be estimated to be useful, it is discarded.

3 Experimental setup

In this section we introduce the labeled data on which we performed our experiments (in Subsection 3.1), and we describe how we ran these experiments. One experiment involves the comparison of the performance of an implementation of a k-NN classifier, namely IB1[1], and RIPPER[2] on a particular labeling task, through 10-fold cross-validation experiments. IB1 is compared in two variants; in one it is trained and tested on the original instance base, in the other it is trained and tested on the transformed instance base. We henceforth refer to the latter variant as T-IB1 (for transformed IB1).

We make a case for an intermediate step in these experiments in which, on the basis of training set partitions, algorithmic parameters of IB1, RIPPER, and T-IB1 are optimized separately in each fold of each experiment. This algorithmic parameter optimization, which is a heuristic classifier wrapping procedure, described in Subsection 3.2, aims to alleviate the problem of the accidental mismatch between the default settings of the respective algorithms and a particular data set, which would obfuscate the comparison. Performing this optimization allows us to state that each algorithm enters the comparison being tuned to the task at hand.

3.1 Data

We performed experiments on sixteen benchmark data sets from the UCI repository [12]. Fourteen data sets have nominal features; *glass* and *optical* have numeric features. Details on numbers of features, numbers of instances, and numbers and entropies of classes are displayed in Table 3.1.

As visible in Table 3.1, we discern between two groups of tasks. On the one hand we identify a group of eight "artificial" tasks in which sets of features or feature values have some conditional dependencies in their informativeness towards solving the task that are known and imposed by the underlying domain. For example, in the three game data sets (*krvskp*, *connect4*, *tic-tac-toe*), absolute conditional dependencies exist between the presence and location of pieces on the board; the presence and position of no single piece is generally informative unless the presence and position of other pieces is known, to determine the correct outcome. The *monks* data sets are classic puzzle games in which the outcome depends on more than a single value in the input. Two other sets, *nursery* and *car*, are generated from a decision support system [13]; the data is fully compliant with an underlying model, which explicitly models certain conditional dependencies. Cf. [12] for more details on the data sets.

The second group of eight tasks are represented by naturally or uniformly sampled instances. There is no exact knowledge, if at all, of conditional dependencies among features in these tasks, and they have not been predesigned or controlled by known rules.

[1] We used the TiMBL software package, version 5 [11], to run experiments with IB1.
[2] We used RIPPER version 1 release 2.5 for our experiments.

Table 1. Basic statistics of the sixteen used data sets.

Feature dependencies	Task	Number of examples	features	classes	Class entropy
Artificial / known conditional dependencies	car	1,730	6	4	1.21
	connect4	67,559	42	3	1.22
	kr-vs-kp	3,197	36	2	1.00
	monks-1	556	6	2	1.00
	monks-2	601	6	2	0.93
	monks-3	554	6	2	1.00
	nursery	12,961	8	5	1.72
	tic-tac-toe	960	9	2	0.93
Natural / unknown conditional dependencies	audiology	228	69	24	3.41
	bridges	110	7	8	2.50
	glass	214	9	6	2.18
	optical	5,620	64	10	3.32
	promoters	106	57	2	1.00
	splice	3,192	60	3	1.48
	soybean	685	35	19	3.84
	votes	437	16	2	0.96

Conditional dependencies are typically picked up by rule induction and decision trees [14], since these algorithms are specifically geared towards detecting strong combinations of mutually or conditionally dependent feature values, that when taken together as a conjunctive condition (a left hand side of a rule, or a path in a decision tree) may cover many same-class instances. In contrast, k-NN does not search for dependencies among features; instead, the typical similarity kernel assumes feature independence. Only through finding nearest neighbors that share a conditionally dependent set of feature values can a k-NN classifier implicitly make use of these dependencies.

These differences may lead to the expectation that RIPPER is able to learn the tasks with the known conditional dependencies better than IB1, the k-NN classifier, but that T-IB1 would be able to profit from RIPPER's proficiency by using its discovered explicit conditional dependencies as features. As for the natural tasks with unknown dependencies, it is unknown beforehand how the three algorithms would compare learning them.

3.2 Cross-validation and wrapped progressive sampling

We applied the three algorithms RIPPER, IB1, and T-IB1 on the sixteen aforementioned benchmark tasks in 10-fold cross-validation experiments. Rather than running the algorithms with their default settings (T-IB1 inherits all parameters of IB1), we ran a heuristic wrapping process [15] to estimate, separately for each training set partitioning, an optimal setting of algorithmic parameters. The implementations available for RIPPER and IB1 [11] both offer several algorithmic

parameters that, indivually and in combinations, can affect the functioning of the algorithms in unpredictable ways, also considering that many combinations of parameter settings are possible. We briefly summarize which combinations we tested for RIPPER and IB1.

The following RIPPER parameters were combined, leading to a total of $6 \times 2 \times 2 \times 3 \times 3 \times 3 = 648$ setting combinations:

1. `-F`, the minimal number of examples a rule may cover, was varied across $1, 2$ (default), $5, 10, 20, 50$. The default is 2.
2. `-a` determines the order of the classes for which rules are induced. We varied among `-freq` (default) and `+freq`.
3. `-n` determines whether value tests can be negated. We varied among `-n` and `-!n` (default).
4. `-S` simplifies the hypothesis more or less. We varied among the values 0.5 (default), 1.0, 2.0.
5. `-O` governs the number of optimization passes undertaken for each class. We varied among $0, 1, 2$ (default).
6. `-L`, *loss ratio*, sets the relative cost of misclassifying minority classes. We varied among values 0.5, 1.0 (default), 2.0.

The TiMBL implementation of IB1 [11] offers the parameters listed below, with some constraints. Due to constraints 4 and 5, the total number of settings is composed of four subgroups of combinations: $5 + (5 \times 2 \times 2) + (9 \times 5 \times 4) + (9 \times 5 \times 4 \times 2 \times 2) = 925$ settings:

1. `-k` determines how many groups of equidistant nearest neighbors are used to classify a new example. We vary among values 1 (default), 3, 5, 7, 9, 11, 13, 15, 19, 25, 35.
2. `-w` determines the employed feature weight in the similarity metric. The five options are no weighting, information gain, gain ratio (default), χ^2, or shared variance.
3. `-m` determines the basic type of similarity metric. The choice is between overlap (default), MVDM, or Jeffrey divergence.
4. Only with `-k` set to a value larger than one, distance-weighted class voting is optional (`-d`). Options are to do no weighting (default), or perform inverse-linear weighting, inverse weighting, or weighting through exponential decay.
5. Only with `-m` with MVDM or Jeffrey divergence it is possible to back-off with low-frequent values to the overlap metric through the `-L` frequency threshold. We vary among 1 (default) and 2.

For smaller data sets it is feasible to run all 648 RIPPER experiments and 925 IB1 experiments in wrapped internal n-fold cross-validation experiments within the main 10-fold cross-validation experiments. We adopted this strategy for all data sets having under 1,000 examples (ten out of the sixteen data sets). For the larger data sets, however, current standard computer technology prohibits exhaustive searches in reasonable time. For the six remaining larger data sets (*car*, *connect4*, *kr-vs-kp*, *nursery*, *optical*, and *splice*) we adopted a new method

called wrapped progressive sampling, henceforth referred to as WPS [16]. We briefly describe WPS here.

WPS is a heuristic, non-exhaustive variant of exhaustive wrapping. It borrows a heuristic from progressive sampling [17]. The goal of progressive sampling is to iteratively seek a data set size at which generalization performance on test material converges. More generally, WPS borrows from the idea that aspects of learning (such as weight estimation or rule induction) can be effectively based on small subsamples or partitionings of the full training set [18–20].

WPS takes as input a data set of labeled examples. In our experiments this is one 90% training set out of a single partitioning of a 10-fold CV experiment. First, WPS generates a series of increasing training and test subsets of which the sizes increase quadratically. They are generated as follows. First, let n be the number of labeled examples in a 80% part of the 90% training set partition. A quadratic sequence of d data set sizes is created by using a factor $f = \sqrt[d]{n}$. In all experiments, $d = 20$. A sequence of $i = \{1 \dots d\}$ data sets with sizes $size_i$ is created by letting $size_1 = 1$ and for every $i > 1$, $size_i = size_{i-1} * f$. We then clip the generated list of 20 sizes down to a list containing only the data sets with more than 500 examples. We also include the 500-example data set itself as the first set. For each of the training sets, an accompanying test set is created by taking, from the tail of the 20% compartment of the training set designated for test material, a set that has 20% of the size of its corresponding training set.

The WPS procedure operates on a large list containing all possible combinations of tested parameter settings of the algorithm (925 settings for IB1, 648 settings for RIPPER), and is aimed at finding an optimal combination in this list. The first step of WPS is to perform experiments with all setting combinations in this list on the smallest training and test subset, producing a list of generalization accuracies for each combination. As the second step, a selection of the best-performing setting combinations is made that will continue to be tested in the next step. To remove badly-performing settings from the current set, a simple linear histogram is computed on all accuracies, dividing them in ten equal-width bins, $b_1 \dots b_{10}$. The bin with the highest accuracies, b_{10}, is taken as the first selected bin. Subsequently, every preceding bin is also selected that represents an equal number of settings or more than its subsequent bin. The setting combinations in the selected bins are then passed on to the next experimentation phase on a larger training and test subset. The WPS process is iterated until either one of these stop conditions is met: (1) After the most recent setting selection, only one setting is left and returned as the best setting; or (2) a random selection is made if there are still several equally-well performing settings the selected settings, and the randomly chosen setting is returned.

In order to demonstrate the effect of parameter optimization, Table 3.2 lists the differences in average accuracy yielded by RIPPER and IB1 before and after WPS on the sixteen tasks. As the results show, WPS parameter optimization improves the average accuracy of RIPPER on ten out of the sixteen tasks; of these ten, the improvement is significant in five cases according to a one-tailed unpaired t-test —a crude but critical test— with at least $p < 0.01$. IB1 improves in twelve

Table 2. Parameter optimization effects on RIPPER and IB1 on the sixteen tasks. > indicates a significantly better accuracy of left algorithm than that of the right algorithm at $p < 0.05$ (one-tailed unpaired t-test); >> indicates $p < 0.01$, and >>> indicates $p < 0.001$. <, <<, and <<< indicate the same in the other direction.

	RIPPER				IB1			
	% Correct test instances				% Correct test instances			
Task	default			optimized	default			optimized
car	87.8	± 3.2	<<<	98.0 ± 0.6	94.0	± 0.9	<<<	96.6 ± 1.2
connect4	75.4	± 0.8	<<	77.0 ± 1.3	71.0	± 0.3	<<<	77.9 ± 2.1
kr-vs-kp	99.2	± 0.4		98.9 ± 0.8	97.7	± 0.5	>>>	96.8 ± 0.7
monks-1	96.1	± 5.9		98.4 ± 4.8	97.0	± 7.4		97.0 ± 7.4
monks-2	64.7	± 4.7	<<<	85.6 ± 11.7	67.0	± 6.2	<<	79.4 ± 10.8
monks-3	98.2	± 2.2		98.7 ± 1.1	97.7	± 1.2	<<	98.9 ± 0.9
nursery	96.6	± 0.7	<<<	98.9 ± 0.5	94.7	± 0.5	<<<	99.3 ± 0.2
tic-tac-toe	97.5	± 1.3	<<<	99.8 ± 0.3	89.5	± 2.6	<<<	99.2 ± 0.6
average	*89.4*			*94.4*	*88.6*			*93.1*
audiology	75.4	± 8.1		76.3 ± 9.0	79.3	± 9.7		79.8 ± 8.7
bridges	58.2	± 14.8		55.5 ± 12.5	54.6	± 8.1		50.9 ± 13.6
glass	65.8	± 9.4		64.6 ± 7.7	73.0	± 5.4		72.2 ± 7.2
optical	91.3	± 1.1		90.7 ± 1.5	98.5	± 0.6		98.6 ± 0.7
promoters	85.8	± 4.8		87.7 ± 5.9	86.7	± 10.6		88.5 ± 10.2
splice	93.4	± 1.6		94.3 ± 1.0	91.9	± 2.0	<<<	95.4 ± 1.0
soybean	91.8	± 2.5		91.5 ± 2.5	91.8	± 3.1	<	94.6 ± 2.9
votes	95.4	± 2.5		95.2 ± 2.8	93.8	± 3.7		95.9 ± 3.2
average	*81.5*			*81.4*	*83.5*			*84.2*

out of sixteen tasks; in eight cases the improvement is significant with at least $p < 0.05$. In one case, *kr-vs-kp*, WPS leads to a significantly lower performance.

On some tasks, such as *tic-tac-toe* and *nursery*, both RIPPER and IB1 perform close to 100% after WPS, where before both yielded much lower accuracies. Both algorithms improve more by WPS on the artifical tasks than on the natural tasks, as also the averages over the averages within the two groups of tasks roughly indicate (the italicized rows in Table 3.2). On the basis of these positive results we also use WPS for T-IB1, for which we provide results in the next section.

4 Results and a variant

In this section we discuss the results obtained in the comparative experiments with WPS-optimized T-IB1 versus RIPPER and IB1. We then analyse the average numbers of rules and active rules per data set, to gain some understanding of the functioning of the feature transformation procedure. To conclude the section we present a variant of the transformation procedure which does not replace the

original features, but which adds the new transformed features to the original ones.

4.1 Comparing RIPPER, IB1, and T-IB1

Table 4.1 lists the average (10-fold cross-validation) accuracies, measured in percentages of correctly classified test instances, of RIPPER, IB1, and T-IB1 (all optimized through WPS) on the sixteen tasks. The T-IB1 results are generally close to those of RIPPER; in two cases RIPPER performs significantly better (*car* and *nursery*), according to one-tailed unpaired t-tests; in two cases case (*monks-2* and *optical*) T-IB1 has the upper hand. There are considerable differences between T-IB1 and IB1, but the differences vary from positive to negative quite widely. Overall, the results suggest that T-IB1 does not clearly deviate from RIPPER, neither from IB1 in a single direction.

Table 3. Average generalization accuracies of RIPPER, T-IB1, and IB1 on the sixteen tasks. > indicates a significantly better accuracy of the left algorithm than that of the right algorithm at $p < 0.05$ (one-tailed unpaired t-test); >> indicates $p < 0.01$; >>> indicates $p < 0.001$.

	% Correct test instances							
Task	RIPPER			T-IB1			IB1	
car	98.0	± 0.6	>>>	89.0	± 3.0	<<<	96.6	± 1.2
connect4	77.0	± 1.3		76.1	± 1.3	<	77.9	± 2.1
kr-vs-kp	98.9	± 0.8		98.9	± 0.8	>>>	96.8	± 0.7
monks-1	98.4	± 4.8		98.4	± 4.8		96.7	± 7.4
monks-2	85.6	± 11.7	<	96.3	± 7.3	>>>	79.4	± 10.8
monks-3	98.7	± 1.2		98.7	± 1.2		98.9	± 0.9
nursery	98.9	± 0.5	>>	97.7	± 1.0	<<<	99.3	± 0.2
tic-tac-toe	99.8	± 0.3		100.0	± 0.0	>>>	99.2	± 0.6
average	*94.4*			*94.4*			*93.1*	
audiology	76.3	± 8.9		77.5	± 10.0		79.8	± 8.7
bridges	55.5	± 12.5		59.1	± 9.3		50.9	± 13.6
glass	64.6	± 7.7		66.1	± 8.5	<	72.2	± 7.2
optical	90.7	± 1.5	<	91.9	± 1.2	<<<	98.6	± 0.7
promoters	85.8	± 4.8		84.1	± 8.1		88.5	± 10.2
splice	94.3	± 1.0		93.9	± 1.4	<<	95.4	± 1.0
soybean	91.5	± 2.5		92.1	± 2.6	<	94.6	± 2.9
votes	95.2	± 2.8		95.2	± 3.1		95.9	± 3.2
average	*81.7*			*82.5*			*84.5*	

4.2 Feature transformation statistics

In Table 4.2 we provide the average numbers of induced rules by WPS-optimized RIPPER (i.e., the number of transformed features) over 10-fold cross-validation partitionings of the sixteen benchmark task data sets. The table also lists the average number of active rules per instance. The number of induced rules exceeds the number of original features in four out of the sixteen tasks (note that the rule features are binary, and the original features are usually multivalued or numeric). This number ranges from 0.7 for *monks-2*, for which RIPPER hardly induces rules aside from the default rule, to 61.9 for *optical* (optical written digit recognition), for which RIPPER induces about six rules on average for each digit to be recognized.

The average number of active rules in instances is often lower than 1, which means that in these tasks often no conditions match to individual examples. (Note that we do not encode the default rule as a bit, since it has no conditions to match on – and to encode it would imply to have an uninformative binary feature that is always active.) On the other hand, for six tasks the average number of matching rules per instance is above 1; the maximal value is 3 rules per instance with the *optical* task again. Remarkably, the *optical* task is one on which T-IB1 performs significantly better than RIPPER – but so does the *monks-2* task with the lowest number of active rules. There is no obvious correlation between the average number of active rules and any positive or negative effect on the performance of T-IB1 as compared to RIPPER or IB1.

There are no remarkable discrepancies in the average numbers of active rules in training and test data; the rules induced on the training set do not appear to overfit or underfit on average.

4.3 An extension: IB1+T

To extend the current analysis, we introduce a second hybrid k-NN classifier that *combines* the original features of IB1 (multivalued, symbolic) with the new transformed features used in T-IB1. We name this new hybrid IB1+T, for "IB1 plus the transformed features". Rather than a transformation, IB1+T utilizes an enrichment of the original set of features. It could be the case that the k-NN classifier is able to compute better distances between instances when more features are available, especially since these new features might be complementary to the basic features. Table 5 displays the average generalization accuracies of IB1+T on the sixteen tasks, and compares them to the results of IB1. As before, WPS was used for both algorithms.

The results in Table 5 indicate that IB1+T performs quite close to IB1; it improves over IB1 significantly in five out of the sixteen tasks. Four of these are artificial tasks (*kr-vs-kp*, *monks-1*, *monks-2*, and *tic-tac-toe*); on *nursery*, IB1+T performs significantly worse. It is notewothy that on two artificial tasks, *monks-1* and *tic-tac-toe*, IB1+T attains 100% accuracy, where neither RIPPER nor IB1 learn this task entirely flawlessly.

Table 4. The original number of features and average numbers of induced rules per task, and average number of active rules per instance, for the sixteen tasks.

Task	# Original features	Average # induced rules		Average # active rules training data	test data
car	6	35.4	± 4.5	0.51	0.51
connect4	42	15.5	± 6.1	0.62	0.62
kr-vs-kp	36	13.8	± 4.1	0.93	0.94
monks-1	6	5.5	± 5.6	0.59	0.64
monks-2	6	0.7	± 0.8	0.03	0.02
monks-3	6	2.6	± 2.1	0.62	0.63
nursery	8	20.1	± 26.0	1.67	1.67
tic-tac-toe	9	9.0	± 1.2	0.96	0.94
audiology	69	20.8	± 4.1	1.60	1.63
bridges	7	8.9	± 4.0	0.46	0.51
glass	9	10.9	± 2.4	2.26	2.17
optical	64	61.9	± 8.3	3.05	2.99
promoters	57	2.9	± 1.2	0.74	0.63
splice	60	12.8	± 6.7	1.70	1.72
soybean	35	27.4	± 2.7	1.59	1.71
votes	16	2.2	± 1.2	0.68	0.68

5 Discussion

Transforming the representation of instances by features discovered by a rule induction algorithm on these instances, and performing k-NN classification on this transformed data, appears, in view of the presented results, a viable alternative approach to using the rules directly, as in standard rule induction. This result is in line with results reported by [6] on regression tasks, and by Domingos on the RISE algorithm [5]. A marked difference is that in RISE, the rules are the *instances* in k-NN classification (and due to the careful generalization strategy of RISE, they can be very instance-specific), while in T-IB1 the rules are the *features* by which the original instances are indexed. When a nearest neighbor is found to a query instance in T-IB1, it is because the two instances share one or more matching rules. The actual classification that is transferred from the memory instance to the new instance is the classification that this memory item is stored with – which may well be another class than any of its matching rules predict.

Our results with the transformation, however, indicate that it does not lead to improved performance. We found that T-IB1, the k-NN classifier with the transformed features, hardly deviates in its performance from RIPPER, and performs widely differently from IB1 both in positive and negative directions. On the one hand, the results do imply that RIPPER's ordered list of rules can be flattened in a k-NN setting, in which the ordering is lost and all matching rules are considered simultaneously, without loss of performance as compared to RIP-

Table 5. Average generalization accuracies of IB1 and IB1+T on the sixteen tasks. > indicates a significantly better accuracy of the immediate left algorithm as compared to the immediate right algorithm at $p < 0.05$ (one-tailed unpaired t-test); >> indicates $p < 0.01$, and >>> indicates $p < 0.001$.

	% Correct test instances				
Task	IB1			IB1+T	
car	96.6	± 1.2		96.5	± 0.9
connect4	77.9	± 2.1		78.0	± 1.3
kr-vs-kp	96.8	± 0.7	<<<	99.1	± 0.6
monks-1	97.0	± 7.4	<<<	100.0	± 0.0
monks-2	79.4	± 10.8	<<<	97.7	± 4.4
monks-3	98.9	± 0.9		98.4	± 2.0
nursery	99.3	± 0.2	>>	98.7	± 0.6
tic-tac-toe	99.2	± 0.6	<<<	100.0	± 0.0
average	*93.1*			*96.1*	
audiology	79.8	± 8.7		77.0	± 11.2
bridges	50.9	± 13.6	<	61.8	± 14.0
glass	72.2	± 7.2		73.0	± 9.0
optical	98.6	± 0.7		98.1	± 1.1
promoters	88.6	± 10.2		89.6	± 10.5
splice	95.4	± 1.0		95.6	± 0.9
soybean	94.6	± 2.9		93.0	± 2.2
votes	95.9	± 3.2		94.3	± 3.4
average	*84.5*			*85.3*	

PER in fourteen out of the sixteen cases we investigated (with two significant improvements of T-IB1 over RIPPER).

The lack of an overall positive payoff of the method may be explained by the fact that RIPPER is doing its best to minimize the overlap between rules. Indeed, as Table 4.2 already witnessed, in many tasks instances are indexed by less than one rule on average (the default rule is not encoded in T-IB1), and matching on one indexed rule feature in most cases will likely lead to the same classification as the firing of that rule in RIPPER would lead to.

We then turned to IB1+T, which adds the transformed features to the original multivalued features rather than using only the transformed features. This hybrid performs reasonably well as compared to IB1. It yields significant improvements in five out of the sixteen cases, counter to one significant loss in performance and ten non-significant differences.

The clearest and most significant improvements in the experiments with the transformed features reported here, but also with the wrapped progressive sampling method for algorithmic parameter optimization, are those obtained on the artificial data sets. We argued earlier that the data representing the artificial

tasks are based on a finite number of predetermined and known conditional dependencies among features and feature values, and that RIPPER should be able to discover these conditional dependencies. We found a slight advantage of RIPPER over IB1 on the artificial tasks. Interestingly, in a subsequent experiment we found the best average accuracies (up to 100% on *tic-tac-toe*) on some of these tasks with IB1+T. In terms of the average of cross-validation averages (which is just a rough indicator), RIPPER's and T-IB1's average score on the artifical tasks is 94.4%, IB1's is 93.1%, while IB1+T's is an impressively higher 96.1%. Apparently, adding the transformed features to the original ones enables the k-NN classifier to estimate distances better in tasks with clear conditional dependencies. This is likely the key result of this study.

As a case in point we briefly illustrate the case of *tic-tac-toe*, a data set containing the 958 possible endings of the simple three-by-three board game. The task is to identify whether the board constitutes a win for X. This can be solved, as RIPPER often does, by discovering eight rules that test on the two diagonal, three horizontal and three vertical lines to detect a three-X-in-a-row situation. Occasionally, however, RIPPER induces a rule that tests on the presence of the O symbol in four locations, not three in a row, predicting a "yes" (a win for X), which is 100% accurate in RIPPER's validation part of the training set. The problem is that such a rule occasionally predicts "yes" incorrectly when the test instance on which the rule fires represents a draw situation (a "no" outcome). Classifying the same test instance, IB1+T finds a nearest neighbor in the training set (its optimized parameter setting has $k = 1$) that matches the rule, but at the same time represents a draw situation. The boards of the two instances only match in six out of nine positions, but it is nevertheless the closest match, yielding a correct classification; matching on the board implicitly corrects the incorrect prediction of the matching rule.

5.1 Future research

One aspect deserving deeper investigation is the rule induction method underlying the feature transformation. RIPPER tends to avoid the overlap of rules, while part of the potential of the T-IB1 approach in improving the base rule inducer lies in cases when instances match with several rules at the same time. Rule induction methods allowing more overlap and redundancy (e.g. [21]) between rules could be more appropriate. Since the classes of the rules are not used in T-IB1, the base system does not even need to be a supervised rule inducer; rather, a frequent itemset miner [22] could be used, which would likely produce overlapping index features.

There are relations between the merging of feature sets, as in IB1+T, and classifier combination (e.g. voting). We see as an advantage over classifier combinations that the current approach assumes just one classifier, but surely a voting meta-learner could be able to take advantage of the "best of both worlds" as well as IB1+T appears to be doing.

Apart from different base feature inducers and comparisons with classifier combinations, more attention should be payed to analysing conditional depen-

dencies in the natural tasks. Surely conditional dependencies exist, but they are likely sparser and more noisy than in the artificial tasks, and rule induction algorithms may not discover them as well as with the artificial tasks. It is normal for these tasks that induced rules have below-100% accuracies, which may be the root cause why T-IB1 is not able to gain from these relatively inaccurate rules as features. More experiments are needed to investigate the effects on feature transformation of data sparseness, data noise, and rule accuracy.

Acknowledgements

The author wishes to thank William Cohen, Walter Daelemans, Jakub Zavrel, Iris Hendrickx, and three anonymous reviewers for sharing useful comments and insights. This research is funded by the Netherlands Organisation for Scientific Research (NWO).

References

1. Clark, P., Niblett, T.: The CN2 rule induction algorithm. Machine Learning **3** (1989) 261–284
2. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1993)
3. Cohen, W.: Fast effective rule induction. In: Proceedings 12th International Conference on Machine Learning, Morgan Kaufmann (1995) 115–123
4. Aha, D.W., Kibler, D., Albert, M.: Instance-based learning algorithms. Machine Learning **6** (1991) 37–66
5. Domingos, P.: Unifying instance-based and rule-based induction. Machine Learning **24** (1996) 141–168
6. Sebag, M., Schoenauer, M.: A rule-based similarity measure. In Wess, S., Althoff, K.D., Richter, M.M., eds.: Topics in case-based reasoning. Springer Verlag (1994) 119–130
7. Cost, S., Salzberg, S.: A weighted nearest neighbour algorithm for learning with symbolic features. Machine Learning **10** (1993) 57–78
8. Dudani, S.: The distance-weighted k-nearest neighbor rule. In: IEEE Transactions on Systems, Man, and Cybernetics. Volume SMC-6. (1976) 325–327
9. Daelemans, W., van den Bosch, A.: Generalisation performance of backpropagation learning on a syllabification task. In Drossaers, M.F.J., Nijholt, A., eds.: Proceedings of TWLT3: Connectionism and Natural Language Processing, Enschede, Twente University (1992) 27–37
10. van den Bosch, A.: Using induced rules as complex features in memory-based language learning. In: Proceedings of the Fourth Conference on Computational Natural Language Learning and of the Second Learning Language in Logic Workshop, New Brunswick, NJ, ACL (2000) 73–78
11. Daelemans, W., Zavrel, J., Van der Sloot, K., van den Bosch, A.: TiMBL: Tilburg memory based learner, version 5.0, reference guide. ILK Technical Report 03-10, Tilburg University (2003)
12. Blake, C., Merz, C.: UCI repository of machine learning databases (1998) `http://www.ics.uci.edu/ mlearn/MLRepository.html`.

13. Bohanec, M., Rajkovic, V.: DEX: An expert system shell for decision support. Sistemica **1** (1990) 145–157
14. Quinlan, J.R.: Comparing connectionist and symbolic learning methods. In Hanson, S.J., Drastal, G.A., Rivest, R.L., eds.: Computational Learning Theory and Natural Learning Systems. Volume 1: Constraints and Prospects. The MIT Press, Cambridge, MA (1994) 445–456
15. Kohavi, R., John, G.: Wrappers for feature subset selection. Artificial Intelligence Journal **97** (1997) 273–324
16. van den Bosch, A.: Wrapped progressive sampling search for optimizing learning algorithm parameters. In Schomaker, L., Verbrugge, R., Taatgen, N., eds.: Proceedings of the Sixteenth Belgian-Dutch Conference on Artificial Intelligence, University of Groningen (2004) To appear.
17. Provost, F., Jensen, D., Oates, T.: Efficient progressive sampling. In: Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining. (1999) 23–32
18. Chan, P.K., Stolfo, S.J.: A comparative evaluation of voting and meta-learning of partitioned data. In: Proceedings of the Twelfth International Conference on Machine Learning. (1995) 90–98
19. Domingos, P.: Using partitioning to speed up specific-to-general rule induction. In: Proceedings of the AAAI-96 Workshop on Integrating Multiple Learned Models, Portland, OR, AAAI Press (1996) 29–34
20. Scheffer, T., Wrobel, S.: Finding the most interesting patterns in a database quickly by using sequential sampling. Journal of Machine Learning Research **3** (2002) 833–862
21. Cohen, W.W., Singer, Y.: A simple, fast, and effective rule learner. In: Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), Menlo Park, CA, AAAI/MIT Press (1999) 335–342
22. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Databases. (1994)