Compression-Based Feature Subset Selection

Bernhard Pfahringer

Austrian Research Institute for Artificial Intelligence
Schottengasse 3
A-1010 Vienna
Austria

E-mail: bernhard@ai.univie.ac.at
Phone: (+43 1) 533-6112
Fax: (+43 1) 532-0652

Keywords: Machine Learning, Inductive Learning, Minimum Description Length Principle, Noise

Abstract

Irrelevant and redundant features may reduce both predictive accuracy and comprehensibility of induced concepts. Most common Machine Learning approaches for selecting a good subset of relevant features rely on cross-validation. As an alternative, we present the application of a particular Minimum Description Length (MDL) measure to the task of feature subset selection. Using the MDL principle allows taking into account all of the available data at once. The new measure is information-theoretically plausible and yet still simple and therefore efficiently computable. We show empirically that this new method for judging the value of feature subsets is more efficient than and performs at least as well as methods based on cross-validation. Domains with both a large number of training examples and a large number of possible features yield the biggest gains in efficiency. Thus our new approach seems to scale up better to large learning problems than previous methods.

Submitted to IJCAI95.

1 Introduction

The problem of *feature subset selection* involves finding a good subset of all given features. Some of these attributes may be irrelevant or redundant. *Goodness* is usually defined by some objective function like predictive accuracy of the resulting concept hypothesis, its structural complexity or – probably domain-dependent – the cost of tests or combinations thereof. As especially predictive accuracy cannot reliably be estimated from the complete set of training examples, usually some kind of cross-validation has to be used [John et al. 94, Kohavi 95, Caruana & Freitag 94].

In this paper we describe the application of the *Minimum Description Length* (MDL) principle [Rissanen 78] as an objective function for judging *goodness* of a feature subset. Feature subsets are used to construct so called *simple decision tables* [Kohavi 95]. MDL takes into account both the simplicity and the accuracy of the simple decision tables induced by particular feature subsets. MDL uses all of the training data at once eliminating the need for costly cross-validation runs. The MDL Principle has been successfully applied in Machine Learning before, for inducing decision trees [Quinlan & Rivest 89, Quinlan 93, Wallace & Patrick 93, Forsyth 93], for inducing production rules [Pfahringer 95], for constructing new attributes [Pfahringer 94], and in ILP [Muggleton et al. 92].

Section 2 will describe, how *simple decision tables* are constructed for given feature subsets and how they are used for classifying test examples. In section 3 we will describe a new MDL coding schema for simple decision tables. Experimental considerations and setup, and empirical results using this new coding schema are reported in section 4. Section 5 discusses open problems and further research directions.

2 Simple Decision Tables

[Kohavi 95] describes simple decision tables and their usage for classification as follows:

- For any feature subset construct a decision table by simply projecting all given training examples on the feature subset.
- For all – after projection – identical examples count class frequencies and assign the majority class to every entry. Also compute the overall majority class.
- When classifying new examples, look up the projected example in the decision table. If there is an entry found, return the appropriate majority class of that entry as the classification result, else return the global majority class.

	A_1	A_2	A_3	A_4	Class
Ex_1	1	1	0	0	+
Ex_2	1	1	0	1	–
Ex_3	1	1	1	1	+
Ex_4	1	0	0	0	–
Ex_5	0	1	0	1	–
Ex_6	1	1	1	0	+
Ex_7	1	1	0	1	+

Table 1: A set of training examples for an artificial 2-class learning task involving 4 boolean attributes.

	A_1	A_2	Class	Examples
$Entry_1$	1	1	+	$Ex_1, Ex_2, Ex_3, Ex_6, Ex_7$
$Entry_2$	1	0	–	Ex_4
$Entry_3$	0	1	–	Ex_5

Table 2: Simple decision table for feature subset $\{A_1, A_2\}$ for the examples given in table 1. The global default class would be +, as there are 4 examples for this class versus only 3 for the other one.

Tables 1 and 2 exemplify simple decision tables, or **DTM** (for Decision Table Majority) as [Kohavi 95] calls them. Table 1 lists some training examples for an arbitrary 2-class learning task involving 4 boolean attributes. Table 2 depicts the respective decision table for the feature subset $\{A_1, A_2\}$. A new example [1011] would be classified as "–" using $entry_2$, and e.g. [0010] would be classified as "+" using the global majority class, as there is no entry for $A_1 = 0, A_2 = 0$ in the table.

To estimate the predictive accuracy (which is used as the objective function) of a given feature subset, an *incremental k-fold cross-validation* (see also section 4) is used. The optimal feature subset is selected by performing a best-first search in the space of all possible feature subsets. In the next section we will show how this incremental k-fold cross-validation can be replaced by a more efficient MDL evaluation.

3 An MDL Measure for Simple Decision Tables

Empirical induction is always faced with the problem of *overfitting* the data, especially in the presence of noise or irrelevant or redundant attributes. The MDL principle is a possible solution as it measures both the simplicity and the accuracy of a particular theory in a common currency, namely in terms of the number of bits needed for encoding. A very good introduction to MDL and also its close relation to Bayesian theory can be found in [Cheeseman 90]. He defines the message length of a theory (called *model* in his article) as:

```
Total message length = Message length to describe the model +
                       Message length to describe the data,
                       given the model.
```

This way a more complex theory will need more bits to be encoded, but might save bits when encoding more data correctly. The theory with the *minimal* total message length is also the *most probable* theory explaining the data [Rissanen 78]. Now the problem for machine learning consists of finding the appropriate coding schemes for the particular kinds of models induced, be it decision trees, propositional rules, Prolog programs, neural networks, or simple decision tables. Actually we don't really need to encode, we just need a formula estimating the number of bits needed if we encoded a theory and data in terms of that theory.

[Forsyth 93] introduces a well-performing formula for encoding decision trees, which we have modified for encoding propositional rules in [Pfahringer 95].

We define the cost of a rule set as follows:

$$cost(ruleset) = n_{nc} * e_{nc} + \Sigma cost(rule_i)$$
$$cost(rule_i) = rc_i + e_i * n_i$$
$$rc_i = Cost_{Premises} + Cost_{Outcome}$$

where n_i is the number of examples covered by the respective rule, and e_i is the average entropy of the classification outcome of that rule defined by:

$$e_i = -(p * log(p) + (1-p) * log(1-p))$$

where p is the proportion of positive examples covered by $rule_i$.[1] Note that $e_i * n_i$ is the number of bits needed by an optimal or 'Huffman' coding of the classifications at $rule_i$ in terms of the relative frequencies of positive and negative examples at $rule_i$. n_{nc} is the total number of examples not covered by the rule set and e_{nc} is the according entropy of this set. One can interpret this as the cost for an empty rule added at the end of the rule set classifying all left-over

[1] $0 * log(0)$ is defined to equal 0.

examples as *negative*. So this is the penalty for one kind of error: omissions or false-negatives. The syntactic complexity of a single rule is accounted for by rc_i which sums up the cost for encoding both the premises ($Cost_{Premises}$) and the conclusion ($Cost_{Outcome}$) for each rule.

Modifying this formula for simple decision tables is straightforward, if we interpret table-entries as rules and take advantage of the highly redundant structure of these "rules": every rule (entry) tests exactly the same attributes. So we only need to encode the feature subset once and then encode the classification outcome of every entry, but need not account for the syntactic complexity of the premises of single rules (entries), i.e. $Cost_{Premises} = 0$ for single rules. But we still need to encode the conclusion of each rule ($Cost_{Outcome}$). A simple-minded local approach could set $Cost_{Outcome} = 1$ for each rule, but we describe a more efficient [2] global encoding below. Furthermore every training example is covered by the decision table (because of the way this table is constructed; but examples might of course be wrongly classified by a given table), therefore $n_{nc} = 0$, too. So we get:

$$cost(decisiontable) = AttrCost + OutcomeCost + \Sigma cost(entry_i)$$
$$cost(entry_i) = e_i * n_i$$

where n_i is the number of examples covered by the respective entry in the decision table, and e_i is the average entropy of the classification outcome of that entry defined as above for rules. So $AttrCost$ and $OutcomeCost$ together measure the syntactic complexity of a decision table, and $\Sigma cost(entry_i)$ measures the table's predictive performance.

Both $AttrCost$ and $OutcomeCost$ can be estimated by applying the above 'Huffman' coding ideas [3]. For each given attribute we have to encode whether it is used by the respective decision table ('1') or not ('0'). An optimal coding takes into account the relative frequencies of 0s and 1s and needs the following number of bits:

$$AttrCost = N_{attrs} * (-(p * log(p) + (1-p) * log(1-p)))$$

[2] A coding schema is said to be *more efficient* than other coding schemas, if it needs fewer bits for encoding the same information content.

[3] An alternative way of coding chosen subsets adopts Quinlan's [Quinlan 94] idea for encoding exceptions: The cost for choosing I items out of a total of N possible items can be estimated as:

$$Cost = log_2 N + log_2 \left(\left(\begin{array}{c} N \\ I \end{array} \right) \right)$$

As both this formula and Huffman coding give similar (though not identical!) values, we prefer Huffman coding because of its simplicity and its therefore smaller computational cost.

where N_{attrs} is the total number of possible attributes and p is the proportion of attributes actually used by the decision table ($p = |AttrsUsed|/N_{attrs}$).

OutcomeCost is defined similarly: $entry_i$ in the table yields class "+" (1) or class "–" (0). So when encoding all entries in the table simultaneously we get:

$$OutcomeCost = N_{entries} * (-(p * log(p) + (1-p) * log(1-p)))$$

where $N_{entries}$ is the total number of entries and p is the proportion of "positive" (+) entries in the decision table ($p = N_{posentries}/N_{entries}$).

Note that this formula for computing bitcost for a decision table is perfectly symmetric (exchanging "+" and "–" classes yields exactly the same formula) and that this formula can be generalized to N-class decision tables in a straightforward way.

To summarize, the new formula measures cost for encoding all the training examples in terms of the theory (the decision table defined by encoding which feature subset is used and encoding classification outcome for every table entry), and classification errors are accounted for at a per-entry basis using local entropies.

4 Experiments and Empirical Results

For empirical testing of our compression-based heuristic for feature subset selection and comparison to existing approaches we have implemented a specialized best-first search algorithm (see below) that can use either our new heuristic or incremental k-fold cross-validation. For further comparisons C4.5 [Quinlan 93] was used on both the original complete sets of attributes and on the best subsets returned by both heuristics for feature subset selection.

Best-first search is only one of a few possible search strategies. Clearly exhaustive enumeration of all possible subsets is out of the question except for the simplest domains. [Caruana & Freitag 94] compare various forms of greedy hill-climbing:

- Forward Selection (FS): start with an empty set of attributes and greedily add attributes one at a time.
- Backward Elimination (BE): start with the complete set of attributes and greedily delete attributes one at a time.
- Forward Stepwise Selection (FSS): start with an empty set of attributes and greedily add or delete attributes one at a time.
- Backward Stepwise Elimination (BSE): start with the complete set of attributes and greedily delete or add attributes one at a time.

- BSE-Slash: more greedy variant of BSE eliminating (slashing) after every step all attributes not used at that step.

They report favorable results, especially for the later three methods (FSS, BSE, and BSE-Slash), for their domain of calendar scheduling tasks. As these later methods perform bidirectional hill-climbing and as they are equipped with a mechanism for preventing cycles, they can theoretically explore all possible subsets (given enough time). [Skalak 94] successfully applies *random mutation hill-climbing*. [John et al. 94, Kohavi 95] make good use of best-first search for all experiments they report.

Contrary to these lessons from the literature our initial experiments with these different search strategies quickly revealed their ineffectiveness for demanding domains like Parity 5/27 (see below) given reasonable constraints on resources (like limiting the maximum number of node-expansions that yield no improvement over the current optimum). We have therefore added a kind of preprocessing step to our best-first search:

1. Compute the heuristic value for *randomly* chosen subsets of attributes by constructing the respective decision tables until a subset is found such that the heuristic value of that set is *significantly* better than the average of all subsets evaluated so far. Of course, for practical reasons (and to ensure termination) there must be a limit set on the number of random *shots* allowed. If this limit is reached, the best subset so far will be used.

2. Use the best subset from step 1 as the starting point for a resource-limited bidirectional best-first search. (The maximum number of node-expansions yielding no improvement over the currently best subset is limited to 50 for all experiments reported below.)

This modified kind of best-first search proved to be sufficient for all domains investigated below.

Incremental k-fold cross-validation is described in [Kohavi 95]. If it is possible to incrementally add examples to *and* delete examples from the data-structures maintained by the learning algorithm, cross-validation can be done incrementally, which saves a lot of computational work. The computational complexity of incremental k-fold cross-validation is independent of k, meaning e.g. leave-one-out cross-validation can be performed as quickly as, say, 10-fold cross-validation. Incremental cross-validation works as follows:

1. Use all the training data to build up the appropriate data-structures.

2. For every fold of training data:

   (a) Delete all examples of this fold.

(b) Classify all examples of this fold, recording all errors.

(c) Re-add all examples of this fold.

3. Return the average error rate.

Simple decision tables are easily updated incrementally: keeping class frequency counts for every table entry is sufficient. Still we have to expect higher computational cost even for incremental cross-validation than for an MDL-based heuristic. Incremental cross-validation is bound to slower than MDL-based heuristics by atleast a constant factor. Every example must be added twice and be deleted once for incremental cross-validation, whereas one addition per example suffices for MDL-based heuristics. Empirical results reported below show significant runtime differences for the more difficult domains.

4.1 Domains

All the domains used in this empirical evaluation are 2-class problems using only non-numeric attributes (but see above comments regarding N-class problems). All results reported are averages of ten runs each using different random samples for learning, giving the mean value and the standard deviation (if it is different from zero).

- Illegal King-Rook-King Chess Positions (KRK): This domain is used as a standard testbed in ILP [Srinivasan et al. 92] for experiments involving various amounts of noise and varying sizes of training sets. It is also easily transformed into a propositional learning problem. The standard translation is to rewrite each example into a boolean vector containing 18 variables which represent the 18 different test predicate instantiations that can be used for constructing clauses. We used training set sizes of 100, 250, and 500 with zero noise for the experiments summarized in tables 3 and 4. Testing was always performed on an independent set of 5000 test examples.

- Parity 5/5: The target concept is the parity of 5 bits. The 5 additional boolean attributes are random (and therefore irrelevant). The training set contains 100 examples, and all 1024 possible examples are used for testing ([John et al. 94] uses the same settings).

- Monk1, Monk2, Monk2loc, Monk3: These data sets are taken from [Thrun et al. 91]. Monk1, Monk2, and Monk3 all have six attributes yielding 432 possible different examples. Ten sets each were drawn randomly from these 432 examples for learning, testing used all 432 examples (as is done originally in [Thrun et al. 91]). Training set sizes are 124, 169, and 122 respectively. For Monk3 class noise is artificially added by switching the class values of 5% of the training examples. Monk2loc is an interesting

alternative representation for the Monk2 problem translating the 6 nominal attributes into 17 (partially redundant) boolean attributes.

- Mushroom: This is a large example database taken from the UCI repository of machine learning databases [Murphy & Aha 94]. 8124 different examples are described by 22 nominal attributes having up to 12 different possible values. Training sets of size 1000 are drawn randomly, all 8124 examples are used for testing.

- Parity 5/27: The target concept is the parity of 5 bits. The 27 additional boolean attributes are random (and therefore irrelevant). The training set size is 4000 each, and 2000 examples each are used for testing. This particular domain was first used by [Pagallo & Haussler 90] and is especially difficult to learn for decision tree algorithms. This should not be surprising, as only relatively few subsets of the large number of all possible subsets (2^{32}) yield a classification performance significantly better than random guessing. To test robustness in the presence of class noise, we artificially added noise in certain cases by switching the class values of 5%, 10%, and 15% of the training examples, respectively.

4.2 Results

Table 3 gives the average accuracies using the best simple decision table found by MDL and ICV (incremental cross-validation) directly, using C4.5 with all available attributes and using C4.5 with the respective best subset of attributes. Generally the MDL-based heuristic is never significantly worse then ICV and on average performs slightly better. Comparison with C4.5 shows that both methods seem to select reasonable subsets of attributes, as performance of C4.5 is never significantly degraded when using these subsets.

Performance is significantly improved for two (Monk1 and Monk2loc) of the 4 domains were C4.5 does not perform well on its own. Interestingly, C4.5's performance given the respective attribute subsets is still strictly worse than both MDL's and ICV's simple decision tables for these domains. This might be attributed to over-pruning as the number of training examples is small (less than 200) and pre-selecting a subset of attributes is a strong kind of prepruning, too.

Table 4 tries to compare the structural complexity of the induced simple decision tables (measuring the average number of attributes selected) and the computational efficiency of MDL and ICV. In addition to total runtimes we also report the average number of different subsets evaluated by both heuristics during the search and the average number of such evaluations per seconds. This last measure will of course vary wildly with both the total number of training examples and the total number of possible attributes. These more fine-grained measures clearly show that MDL on average not only evaluates (sometimes significantly)

Domain	Method	Acc	C4.5acc	C4.5subset
KRK100	MDL	94.64±1.74	98.21±0.92	98.06±0.95
	ICV	88.24±4.81	98.21±0.92	93.82±4.81
KRK250	MDL	97.17±1.23	98.79±0.57	98.70±0.76
	ICV	97.33±1.32	98.79±0.57	98.52±0.67
KRK500	MDL	97.90±0.64	99.02±0.62	99.05±0.58
	ICV	97.58±0.81	99.02±0.62	98.72±0.40
Parity5/5	MDL	98.26±1.55	84.37±3.70	85.54±2.17
	ICV	98.26±1.55	84.37±3.70	85.54±2.17
Monk1	MDL	95.00±2.08	76.04±5.93	86.39±4.20
	ICV	95.00±2.08	76.04±5.93	86.39±4.20
Monk2	MDL	76.57±1.84	64.63±1.92	65.32±1.65
	ICV	72.99±5.53	64.63±1.92	65.42±1.55
Monk2loc	MDL	98.70±1.09	69.58±4.27	88.87±2.48
	ICV	98.70±1.09	69.58±4.27	88.87±2.48
Monk3	MDL	91.62±3.94	89.21±4.76	88.06±4.20
	ICV	92.31±2.55	89.21±4.76	89.07±4.34
Mushroom	MDL	99.88±0.12	99.67±0.42	99.84±0.22
	ICV	99.89±0.09	99.67±0.42	99.64±0.42

Table 3: Average accuracies (Acc) and standard deviations for MDL and ICV simple decision tables, and for C4.5 using all attributes (C4.5acc) and for C4.5 using the respective subsets (C4.5subset) for various domains.

more subsets per time unit but also evaluates a lesser number of subsets in total. Both of these factors account for the better total runtimes of MDL when compared to ICV.

In summary, MDL performs slightly better than ICV for all simple domains, and differences are more pronounced in favor of MDL for the more complex domains Monk2loc and Mushroom. Especially for Mushroom MDL finds significantly smaller subsets (of comparable predictive accuracy, see table 3) in roughly a fourth of runtime.

Table 5 summarizes all results for Parity 5/27. MDL is significantly quicker than ICV for all levels of noise, and induces simpler subsets than ICV for all but the 15% noise level. At zero noise MDL exactly identifies the necessary and sufficient set of attributes whereas ICV adds 3.5 irrelevant attributes on average. A difference between MDL and ICV in terms of the number of irrelevant attributes included is also present for both 5% and 10% of class noise, though less pronounced. This difference might be explained by MDL assessing both simplicity and accuracy of a decision table, whereas ICV has no inherent notion

Domain	Method	#Attrs	#Evals	#Evals/sec	Runtime
KRK100	MDL	4.3±0.5	657±37	54.6±1.1	12.04±0.56
	ICV	5.2±1.1	861±112	44.1±1.9	19.50±2.42
KRK250	MDL	4.9±0.9	659±27	38.6±0.8	17.11±0.93
	ICV	4.8±1.2	733±34	21.3±0.5	34.52±2.20
KRK500	MDL	5.6±0.8	681±30	24.9±4.9	29.77±12.64
	ICV	5.2±1.0	733±31	11.1±0.6	66.06±4.21
Parity5/5	MDL	5.0±0.0	814±164	40.7±7.7	0.81±0.16
	ICV	5.0±0.0	865±53	37.1±2.3	0.87±0.05
Monk1	MDL	3.0±0.0	64±0	36.6±0.6	1.75±0.03
	ICV	3.0±0.0	64±0	32.9±0.4	1.95±0.02
Monk2	MDL	5.2±0.4	63±2	34.8±3.6	1.84±0.20
	ICV	4.9±0.9	64±0	28.1±0.7	2.28±0.06
Monk2loc	MDL	6.0±0.0	763±51	41.7±0.8	18.30±1.45
	ICV	6.7±0.6	778±75	25.5±0.4	30.47±2.95
Monk3	MDL	2.3±0.5	64±0	35.1±1.0	1.83±0.06
	ICV	2.5±0.5	64±0	32.5±0.4	1.97±0.02
Mushroom	MDL	4.2±0.4	1280±247	18.8±0.8	67.62±10.79
	ICV	7.3±0.5	1679±134	4.5±0.2	373.14±41.09

Table 4: Average number of selected attributes (#Attrs), average number of subsets evaluated (#Evals), average number of subsets evaluated per second (#Evals/sec), and average total runtimes (Runtime), and all standard deviations for various domains.

of simplicity; it optimizes solely accuracy.

To summarize, MDL empirically performs at least as well as ICV in terms of selecting good subsets of attributes. For complex domains MDL is significantly more efficient than ICV in terms of runtime. Both heuristics seem to find reasonable subsets when compared with C4.5 and both might act as a preprocessing filter for C4.5 for domains where C4.5 performs not so well on its own.

5 Conclusions, Related Work, and Further Research

We have defined an MDL measure for simple decision tables. This new measure is information-theoretically plausible in the way it encodes such tables and the examples and it also gives good results in the experiments reported above. It seems to be a more efficient alternative to standard cross-validation, which opti-

	MDL (0%)	ICV (0%)	MDL (5%)	ICV (5%)
Accuracy	100.0	99.97±0.06	100.0	94.91±0.15
C4.5acc	50.85±2.41	50.85±2.41	52.40±2.73	52.40±2.73
C4.5subset	100.0	100.0	100.0	95.16±0.16
#RelAttrs	5	5	5	4.8±0.6
#IrrelAttrs	0	3.5±0.5	2±0	3.7±1.5
#Evals	1609±173	2735±116	3232±1032	3092±1280
#Evals/sec	3.40±0.49	1.13±0.03	2.89±0.62	1.41±0.07
Runtime	490±123	2425±164	1239±628	2220±1026
	MDL (10%)	ICV (10%)	MDL (15%)	ICV (15%)
Accuracy	99.91±0.19	99.62±0.15	99.71±0.35	99.63±0.16
C4.5acc	50.84±1.52	50.84±1.52	50.27±1.99	50.27±1.99
C4.5subset	100.0	100.0	100	100
#RelAttrs	5	5	5	5
#IrrelAttrs	2.9±0.3	3	3	3
#Evals	2959±1086	4248±710	2957±1294	3407±815
#Evals/sec	2.69±0.58	1.19±0.06	2.57±0.50	1.09±0.07
Runtime	1261±729	3824±766	1283±915	3138±833

Table 5: Parity 5/27: Average accuracies (Accuracy) and standard deviations for MDL and ICV simple decision tables, and for C4.5 using all attributes (C4.5acc) and for C4.5 using the respective subsets (C4.5subset). Average number of selected relevant (#RelAttrs) and irrelevant attributes (#IrrelAttrs), average number of subsets evaluated (#Evals), average number of subsets evaluated per second (#Evals/sec), and average total runtimes (Runtime), and all standard deviations. All results for class noise levels of 0%, 5%, 10%, and 15% respectively.

mizes solely predictive accuracy for selecting good subsets of features in inductive learning.

Alternative methods for judging attribute relevance are e.g. FOCUS [Almuallim & Dietterich 91] and RELIEF [Kira & Rendell 92]. Their respective shortcomings are detailed in [John et al. 94]. RELIEF is only able to delete irrelevant attribute, but it cannot delete redundant attributes, whereas FOCUS might be trapped into selecting a single attribute having a distinct value for every single training example, which may result in poor predictive accuracy. Both MDL and ICV do not exhibit these problems.

The method proposed in this paper has some shortcomings. Firstly it is strictly limited to propositional learning tasks. One might want to assess the irrelevance or redundancy of relations in first-order learning, too. Unfortunately it

is not obvious how the proposed method could be adapted for first-order learning. Secondly this method cannot handle numerical attributes directly. But we are working on a discretization method based on a similar MDL-based heuristic. Numerical attribute discretization could be used as a preprocessing step for feature subset selection. Thirdly, when using simple decision tables as a preprocessing step for other induction algorithms, one must be careful in matching the implicit biases of the combined algorithms, e.g. C4.5 tends to overprune decision trees induced from feature subsets in some of the domains discussed in section 4.

Further research will concentrate on the following two topics:

- When using simple decision tables for classificaton directly, missing entries could be dealt with in a more sophisticated manner, e.g. the majority class for all near-misses (examples differing in exactly one attribute value) could be used instead of the global majority class default.
- Compression-based feature subset selection could also be used as an input filter for constructive induction [Bloedorn et al. 93, Pfahringer 94], which can help the learning process in cases of an inadequate initial representation language.

In summary, the new MDL measure for simple decision tables seems to be at least as reliable as incremental cross-validation in terms of selecting reasonable subsets of features, and seems to be significantly more efficient than incremental cross-validation for complex learning tasks.

Acknowledgements

Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian Federal Ministry of Science and Research. I would like to thank Gerhard Widmer for constructive discussion and help with this paper, and Johannes Fürnkranz for pointers to relevant literature, and for providing the king-rook-king position generator.

References

[Almuallim & Dietterich 91] Almuallim H., Dietterich T.G.: Learning with Many Irrelevant Features, in Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI Press/MIT Press, Menlo Park, Vol. II, pp.547-552, 1991.

[Bloedorn et al. 93] Bloedorn E., Wnek J., Michalski R.S.: Multistrategy Constructive Induction: AQ17-MCI, in Michalski R.S. and Tecuci G.(eds.), Proceedings of the Second International Workshop on Multistrategy Learning (MSL-93), Harpers Ferry, W.VA., pp.188-206, 1993.

[Caruana & Freitag 94] Caruana R., Freitag D.: Greedy Attribute Selection, in Cohen W.W. and Hirsh H.(eds.), Machine Learning: Proceedings of the Eleventh International Conference (ML94), Morgan Kaufmann, San Mateo, CA, 1994.

[Cheeseman 90] Cheeseman P.: On Finding the Most Probable Model, in Shrager J., Langley P.(eds.): *Computational Models of Discovery and Theory Formation*, Morgan Kaufmann, Los Altos, CA, 1990.

[Forsyth 93] Forsyth R.S.: Overfitting Revisited: An Information-Theoretic Approach to Simplifying Discrimination Trees, in JETAI 6(3), 1994.

[John et al. 94] John G.H., Kohavi R., Pfleger K.: Irrelevant Features and the Subset Selection Problem, in Cohen W.W. and Hirsh H.(eds.), Machine Learning: Proceedings of the Eleventh International Conference (ML94), Morgan Kaufmann, San Mateo, CA, 1994.

[Kira & Rendell 92] Kira K., Rendell L.A.: A Practical Approach to Feature Selection, in Sleeman D. and Edwards P.(eds.), Machine Learning: Proceedings of the Ninth International Workshop (ML92), Morgan Kaufmann, San Mateo, CA, pp.249-256, 1992.

[Kohavi 95] Kohavi R.: The Power of Decision Tables, Paper, submitted. (also available electronically as ftp://starry.stanford.edu/pub/ronnyk/tables.ps)

[Muggleton et al. 92] Muggleton S., Srinivasan A., Bain M.: Compression, Significance, and Accuracy, in Sleeman D. and Edwards P.(eds.), Machine Learning: Proceedings of the Ninth International Workshop (ML92), Morgan Kaufmann, San Mateo, CA, pp.338-347, 1992.

[Murphy & Aha 94] Murphy P.M., AhA D.W.: UCI repository of machine learning databases. For information contact ml-repository@ics.uci.edu.

[Pagallo & Haussler 90] Pagallo G., Haussler D.: Boolean Feature Discovery in Empirical Learning, Machine Learning, 5(1), 71-100, 1990.

[Pfahringer 94] Pfahringer B.: CiPF 2.0: A Robust Constructive Induction System, Proceedings of the Workshop on Constructive Induction and Change of Representation, 11th International Conference on Machine Learning (ML-94/COLT-94), New Brunswick, New Jersey., 1994. (also available electronically as http://cobar.cs.umass.edu/mlc94/papers/pfahringer.ps)

[Pfahringer 95] Pfahringer B.: A New MDL Measure for Robust Rule Induction, submitted to the 8th European Conference on Machine Learning (ECML95), 1995. (also available electronically as ftp://ftp.ai.univie.ac.at/papers/oefai-tr-94-29.ps.Z)

[Quinlan & Rivest 89] Quinlan J.R, Rivest R.L.: Inferring Decision Trees using the Minimum Description Length Principle, in Information and Computation, 80:227-248, 1989.

[Quinlan 93] Quinlan J.R.: C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, 1993.

[Quinlan 94] Quinlan J.R.: The Minimum Description Length Principle and Categorical Theories, in Cohen W.W. and Hirsh H.(eds.), Machine Learning: Proceedings of the Eleventh International Conference (ML94), Morgan Kaufmann, San Mateo, CA, 1994.

[Rissanen 78] Rissanen J.: Modeling by Shortest Data Description, in Automatica, 14:465-471, 1978.

[Skalak 94] Skalak D.B.: Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms, in Cohen W.W. and Hirsh H.(eds.), Machine Learning: Proceedings of the Eleventh International Conference (ML94), Morgan Kaufmann, San Mateo, CA, 1994.

[Srinivasan et al. 92] Srinivasan A., Muggleton S., Bain M.: Distinguishing Exceptions from Noise in Non-Monotonic Learning, in Proceedings of the 2nd International Workshop on Inductive Logic Programming (ILP), Tokyo, 1992.

[Thrun et al. 91] Thrun S.B., et.al.: The MONK's Problems: A Performance Comparison of Different Learning Algorithms, CMU Tech Report, CMU-CS-91-197, 1991.

[Wallace & Patrick 93] Wallace C.S., Patrick J.D.: Coding Decision Trees, Machine Learning, 11(1), 1993.