More Efficient Windowing

Johannes Fürnkranz
Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A-1010 Wien, Austria
E-mail: juffi@ai.univie.ac.at

Abstract

Windowing has been proposed as a procedure for efficient memory use in the ID3 decision tree learning algorithm. However, previous work has shown that windowing may often lead to a decrease in performance. In this work, we try to argue that rule learning algorithms are more appropriate for windowing than decision tree algorithms, because the former typically learn and evaluate rules independently and are thus less susceptible to changes in class distributions. Most importantly, we present a new windowing algorithm that achieves additional gains in efficiency by saving promising rules and removing examples covered by these rules from the learning window. While the presented algorithm is only suitable for redundant, noise-free data sets, we will also briefly discuss the problem of noisy data for windowing algorithms.

Introduction

Windowing is a general technique that aims at improving the efficiency of inductive classification learners by identifying an appropriate subset of the given training examples, from which a theory of sufficient quality can be induced. Such procedures are also known as *subsampling*. Windowing has been proposed as a supplement to the inductive decision tree learner ID3 (Quinlan 1983) in order to allow it to tackle tasks which would otherwise have exceeded the memory capacity of the computers of those days.

Despite first successful experiments in the KRKN domain (Quinlan 1983) windowing has not played a major role in machine learning research. One reason for this is certainly the rapid development of computer hardware, which made the motivation for windowing seem less compelling. However, recent work in the areas of *Knowledge Discovery in Databases* (Kivinen & Mannila 1994; Toivonen 1996) and *Intelligent Information Retrieval* (Lewis & Catlett 1994; Yang 1996) has recognized the importance of dimensionality reduction through subsampling for reducing both, learning time and memory requirements.

A good deal of the lack of interest in windowing can also be attributed to an empirical study (Wirth & Catlett 1988) which showed that windowing is unlikely to gain any efficiency. The authors studied windowing with ID3 in various domains and concluded that windowing cannot be recommended as a procedure for improving efficiency. The best results were achieved in noise-free domains, such as the *Mushroom* domain, where windowing was able to perform on the same level as ID3. In noisy domains it can be considerably slower. There is some evidence that slight variations of the basic windowing procedure like the one employed in C4.5 (Quinlan 1993) can improve the performance of windowing, in particular on noise-free domains (Catlett 1991a), but no further empirical study has been devoted to this subject.[1]

Thus, one goal of this paper is to study the suitability of windowing for rule learning algorithms. We will show that windowing can in fact deliver what can be reasonably expected, namely to yield significant gains in efficiency without losing accuracy in noise-free data sets with a fair level of redundancy. However, the main result of the paper is a new algorithm that demonstrates how to increase the efficiency of windowing by using a separate-and-conquer strategy that enables it to ignore parts of the search space that have already been covered and to concentrate on yet uncovered parts. Finally, we will give some thought to the problem of noise for windowing algorithms.

Separate-and-Conquer Rule Learning

We have conducted our study in the framework of *separate-and-conquer* rule learning algorithms that has recently gained in popularity (Fürnkranz 1997b). Our basic learning algorithm, DOS,[2] is a simple propositional version of FOIL (Quinlan 1990). It employs a top-down hill-climbing search on the information gain heuristic. The only stopping criteria are completeness and consistency, i.e., rules are specialized until they do not cover any negative examples, and more rules are added to the theory until all positive examples are covered. Thus DOS has no noise-handling capabilities whatsoever. However, for reasons discussed later in this paper we think that noise is a fundamental problem in windowing, which cannot be solved by simply using a noise-tolerant learning algorithm.

[1] A reviewer has pointed us to recent work by Domingos (1996), who has obtained good results for windowing with a bottom-up rule learner.

[2] Dull Old Separate-and-conquer

```
procedure WIN-DOS-3.1(Examples,InitSize,MaxIncSize)

Train = RANDOMSAMPLE(Examples,InitSize)
Test = Examples \ Train
repeat
    Theory = DOS(Train)
    NewTrain = ∅
    OldTest = ∅
    for Example ∈ Test
        Test = Test \ Example
        if CLASSIFY(Theory,Example) ≠ CLASS(Example)
            NewTrain = NewTrain ∪ Example
        else
            OldTest = OldTest ∪ Example
        if |NewTrain| = MaxIncSize
            exit for
    Test = APPEND(Test,OldTest)
    Train = Train ∪ NewTrain
until NewTrain = ∅
return(Theory)
```

Figure 1: Extending DOS with windowing.

Windowing

Figure 1 depicts the basic windowing algorithm that underlies our implementation, which follows the description of (Quinlan 1983). The algorithm starts by picking a random sample of a user-settable size *InitSize* from the total set of *Examples*. It uses these examples for learning a theory with a given learning algorithm, in our case the DOS algorithm briefly described in the previous section. This theory is then tested on the remaining examples and all examples that are misclassified by the current theory are removed from the test set and added to the training set of the next iteration. In order to keep the size of the training set small, another parameter, *MaxIncSize*, controls the maximum number of examples that can be added to the training set in one iteration. If this number is reached, no further examples are tested and the next theory is learned from the new training set. To make sure that all examples are tested in the first few iterations, our implementation appends the examples that have already been tested to the remaining examples in the test set, so that testing will start with new examples in the next iteration.

A More Efficient Version of Windowing

One thing that happens frequently when using windowing with a rule learning algorithm is that good rules have to be discovered again and again in subsequent iterations of the windowing procedure. Although correctly learned rules will add no more examples to the current window, they have to be re-learned in the next iteration as long as the current theory is not complete and consistent with the entire training set. We have developed a new version of windowing, which tries to exploit the property of separate-and-conquer learning algorithms that regions of the example space which are already covered by good rules need not be further considered in subsequent iterations.

```
procedure WIN-DOS-95(Examples,InitSize,MaxIncSize)

Train = RANDOMSAMPLE(Examples,InitSize)
Test = Examples \ Train
OldTheory = ∅
repeat
    NewTheory = DOS(Train)
    Theory = NewTheory ∪ OldTheory
    NewTrain = ∅
    OldTest = ∅
    for Example ∈ Test
        Test = Test \ Example
        if CLASSIFY(Theory,Example) ≠ CLASS(Example)
            NewTrain = NewTrain ∪ Example
        else
            OldTest = OldTest ∪ Example
        if |NewTrain| = MaxIncSize
            exit for
    Test = APPEND(Test,OldTest)
    Train = Train ∪ NewTrain ∪ COVER(OldTheory)
    OldTheory = ∅
    for Rule ∈ Theory
        if CONSISTENT(Rule,NewTrain)
            OldTheory = OldTheory ∪ Rule
            Train = Train \ COVER(Rule)
until NewTrain = ∅
return(Theory)
```

Figure 2: A more efficient version of windowing.

The WIN-DOS-95 algorithms (figure 2) starts just like WIN-DOS-3.1: it selects a random subset of the examples, learns a theory from these examples, and tests it on the remaining examples. However, contrary to WIN-DOS-3.1, it does not merely add all examples that have been incorrectly classified to the window for the next iteration, but also removes all examples that have been classified by consistent rules from this window. A rule is considered consistent, when it did not cover a negative example during the testing phase. Note that this does not necessarily mean that the rule is consistent with all examples in the training set, as it may contradict a testing example which had not been tested before *MaxIncSize* examples were added to the window. Thus apparently consistent rules have to be remembered and tested again in the next iteration. However, we expect that removing examples that are covered by these rules from the window should keep the window size small and thus decrease learning time.

In preliminary experiments it turned out that one problem that happens more frequently in WIN-DOS-95 than in regular windowing or basic separate-and-conquer learning is that of over-specialized rules. Often a consistent rule is found at a low example size, but other rules are found later that cover all of the examples this special rule covers. Note that this problem cannot be removed with a syntactic generality test: consider, for example, the case where a rule stating that a KRK position is illegal if the two kings are on the same square is learned from a small set of the data, and a more general rule is discovered later, which states that all

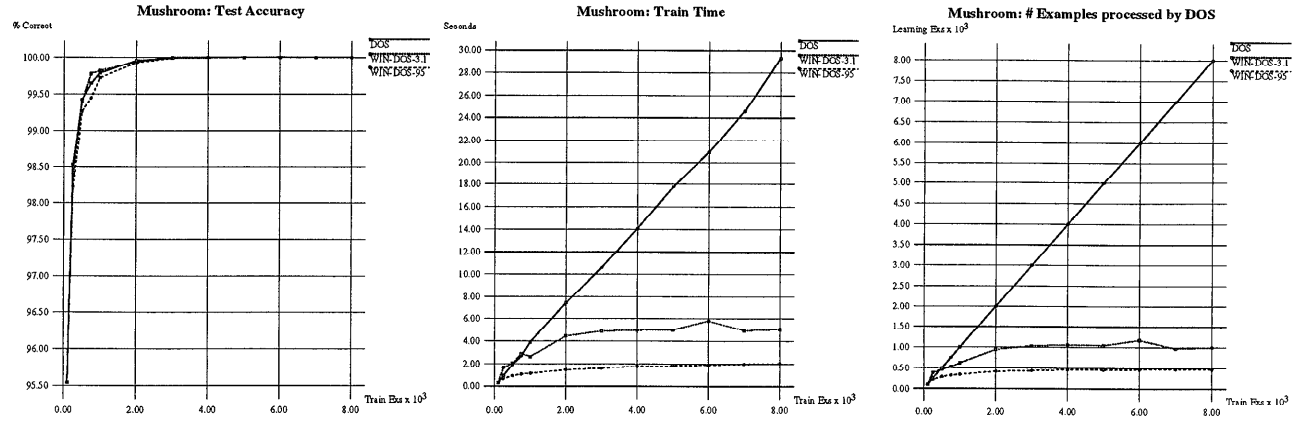

Figure 3: Results in the *Mushroom* domain.

positions are illegal in which the two kings occupy adjacent squares. Sometimes the examples of the special case can also be covered by more than one of the other rules. We have thus developed the following procedure to remove redundant rules: After a theory has been learned, each rule is tested on the complete training set and rules are ordered according to the number of examples they cover. Starting with the rule with the least coverage, each rule is tested whether the examples it covers are also covered by the remaining rules. If so, the rule is removed. This procedure can be implemented quite efficiently and will only be performed once at the end of each of the three algorithms.

Experimental Evaluation

We have compared both versions of windowing on a variety of noise-free domains. In each domain we ran a series of experiments with varying training set sizes. For each training set size 10 different subsets of this size were selected from the entire set of preclassified examples. All three algorithms, DOS, WIN-DOS-3.1, and WIN-DOS-95 were run on each of these subsets and the results of the 10 experiments were averaged. For each experiment we measured the accuracy of the learned theory on the entire example set, the total run-time of the algorithm,[3] and the total number of examples that are processed by the basic learning algorithm. For DOS, this is of course the size of the respective training set, while for the windowing algorithms this is the sum of the training set sizes of all iterations of windowing. All experiments shown below were conducted with a setting of $InitSize = 100$ and $MaxIncSize = 50$. This setting is briefly discussed in the next section.

Figure 3 shows the accuracy, run-time, and number of processed examples results for the three algorithms in the 8124 example *Mushroom* database. WIN-DOS-3.1 seems to be effective in this domain, at least for higher (> 1000) training set sizes. Our improved version, WIN-DOS-95, clearly outperforms simple windowing in terms of run-time, while there are no significant differences in terms of accuracy. WIN-DOS-3.1 needs to submit a total of about 1000 examples to the DOS learner, while WIN-DOS-95 can save about half of them. In a typical run with the above parameter settings, WIN-DOS-3.1 needs about 4 to 5 iterations, the last of them using a window size of about 200 to 350 examples for learning the correct concept. WIN-DOS-95 needs about the same number of iterations, but it rarely keeps more than 150 examples in its window.

This domain is particularly interesting, because windowing with the decision tree learner ID3 could not achieve significant run-time gains over pure ID3 in a previous study (figure 2 of (Wirth & Catlett 1988)), while the slightly modified version of windowing used in C4.5 is able to achieve a run-time improvement of only about 15% (p. 59 of (Quinlan 1993)). Our results, on the other hand, show that windowing is able to achieve a significant advantage in terms of run-time at example sizes of about 3000 or higher, where both windowing algorithms reach a plateau.

We think that the reason for these different results is that divide-and-conquer learning as used in ID3 is more sensitive to changes in class distributions in the training examples. The sensitivity of ID3 to such changes is also confirmed by (Quinlan 1993) where it is reported that changing windowing in a way such that the class distribution in the initial window is as uniform as possible produces better results.[4] In our opinion this sensitivity is caused by the need of decision tree algorithms to optimize the class distribution in *all* successor nodes an interior node. On the other hand, different class distributions will only have a minor effect on separate-and-conquer learners, because they are learning *one* rule at a time. Adding uncovered positive examples to the current window will not alter the evaluation of rules

[3]Measured in CPU seconds of a microSPARC 110MHz running compiled Allegro Common Lisp code under SUN Unix 4.1.3.

[4]This equal-frequency subsampling method has first been described in (Breiman *et al.* 1984) for dynamically drawing a subsample at each node in a decision tree from which the best split for this node is determined. In C4.5 this technique is used to seed the initial window.

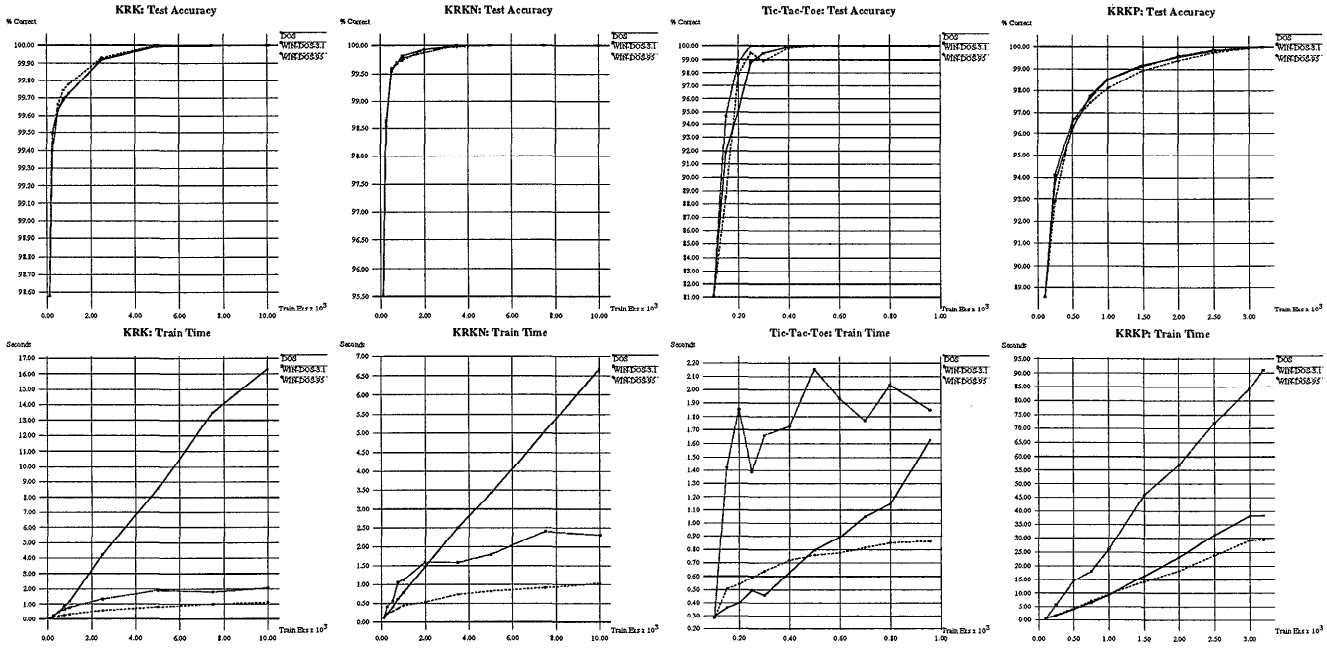

Figure 4: Results in the KRK, KRKN, Tic-Tac-Toe, and KRKP domains.

that do not cover the new examples, but it may cause the selection of a new root node in decision tree learning.

Figure 4 shows the results of experiments in four other noise-free domains in terms of accuracy and run-time.[5] The graphs for the total number of examples processed by DOS, which we had to omit because of space limitations, were quite similar to the run-time graphs in all important aspects. The first domain is a propositional version of the well-known KRK classification task, which is commonly used as a benchmark for relational learning algorithms. The target concept is to learn rules for recognizing illegal white-to-move chess positions with only the white king and rook and the black king on the board. The propositional version of this domain consists of 18 binary attributes that encode the validity or invalidity of relations like `adjacent`, <, and = between the coordinates of the three pieces on the chess board. This task seems to be well-suited for windowing. Predictive accuracy reaches 100% at about 5000 training examples for all three algorithms. The accuracy differences for lower training set sizes are not significant. At the same size, the run-time of both windowing algorithms reaches a plateau. The results in the KRKN chess endgame domain with training set sizes of up to 10,000 examples are quite similar. However, for smaller training set sizes, which presumably do not contain enough redundancy, windowing can take significantly longer than learning from the complete data set. A similar behavior has been observed in the 958 example *Tic-Tac-Toe* endgame domain.[6] Here, predictive accuracy of all three algorithms reaches 100% at about half the total example set size. Interestingly, WIN-DOS-3.1 reaches this point considerably earlier (at about 250 examples). On the other hand, WIN-DOS-3.1 is not able to achieve an advantage over DOS in terms of run-time, although it is obvious that it would overtake DOS at slightly larger training set sizes. WIN-DOS-95 reaches this break-even point much earlier and continues to build up a significant gain in run-time at larger training set sizes. Thus in all domains discussed so far, WIN-DOS-95 is able to achieve significant run-time gains by avoiding to re-learn good rules that have already been discovered in earlier iterations.

In all domains considered so far, removing a few randomly chosen examples from the larger training set sizes did not affect the learned theories. Intuitively, we would call such a training set *redundant*. In the 3196 example KRKP data set, on the other hand, the algorithms were not able to learn theories that are 100% correct when tested on the complete data set unless they use the entire data set for training. We would call such a data set *non-redundant*, as it seems to be the case that randomly removing only a few examples will already affect the learned theories. In this domain, WIN-DOS-3.1 processes about twice as many examples as DOS for each training set size. Our improved version of windowing, on the other hand, processes only a few more examples than DOS at lower sizes, but seems to be able to exploit some redundancies of the domain at

[5] We have to apologize for the bad readability of the graphs of figure 4. Although the curves are hard to discern, we have decided to leave all of them in the paper, because we think that all important aspects are clearly visible, and that the identification of the curves can be inferred from the description in the text.

[6] Note that this example set is only a subset of the Tic-Tac-Toe data set that has been studied in (Wirth & Catlett 1988). We did not have access to the full data set.

larger training set sizes. The reason for this behavior probably is that even in non-redundant training sets, some parts of the example space may be redundant, i.e. while removing randomly chosen examples from the entire example space will probably affect the learning result, removing randomly chosen examples from certain regions of the example space will not affect the result. WIN-DOS-95 exploits this fact by avoiding to re-learn rules that cover such parts of the example space.

Parameter Settings

All experiments reported above have been performed with *InitSize* = 100 and *MaxIncSize* = 50. Different variations of the *InitSize* parameter have been investigated in (Wirth & Catlett 1988). Their results indicate that the algorithm is quite insensitive to this parameter, which we have empirically confirmed. Nevertheless, it might be worthwhile to use a theoretical analysis of subsampling similar to (Kivinen & Mannila 1994) for determining promising values of this parameter for a particular domain.

However, we attribute more significance to the choice of the *MaxIncSize* parameter, which specifies the maximum number of examples that can be added to a window. Experiments with different settings of this parameter showed that in the KRK domain the performance of the algorithms in terms of run-time and total number of examples needed for learning is best if this parameter is kept comparably low (10 to 50 examples). In this range, the parameter is relatively insensitive to its exact setting. If more examples are added to the window size, performance degrades. For example at *MaxIncSize* = 50, WIN-DOS-3.1 performs about 4 iterations of the basic learning algorithm processing a total of about 700 examples, the final window containing about 250 examples. At *MaxIncSize* = 1000 on the other hand, the basic learning module not only has to process about twice as many examples, but windowing also takes more iterations to converge. Similar behavior can be observed for WIN-DOS-95. Thus it seems to be important to continuously evaluate the learned theories in order to focus the learner on the parts of the search space that have not yet been correctly learned. This finding contradicts the heuristic that is currently employed in C4.5, namely to add at least half of the total misclassified examples. However, this heuristic was formed in order to make windowing more effective in noisy domains (Quinlan 1993), a goal that in our opinion cannot be achieved with merely using a noise-tolerant learner inside the windowing loop for reasons discussed in the next section.

The Problem of Noise in Windowing

An adaptation of the procedures discussed in this paper to noisy domains is a non-trivial endeavor. We see the major problem with windowing in noisy domains in the fact that it will eventually incorporate all noisy examples into the learning window (as they should be misclassified by the learned rules), but a typical window will only contain a subset of the original learning examples. Thus the proportion of noisy examples in the learning window will be much higher than the noise level in the entire data set, which will make learning considerably harder.

Assume for example that Win-DOS, using a noise-tolerant basic inducer, has identified the correct theory from 1000 examples in a 11,000 examples domain, where 10% of the examples are misclassified due to noise. In the next iteration, about 1000 noisy examples will be misclassified by the correct theory and will be added to the window, thus doubling its size. Assuming that the original window also contained about 10% noise, more than half of the examples in the new window are now erroneous, so that the classification of the examples in the new window is in fact mostly random. It can be assumed that many more examples have to be added to the window in order to recover the structure that is inherent in the data. This hypothesis is consistent with the results of (Wirth & Catlett 1988) and (Catlett 1991a), where it was shown that windowing is highly sensitive to noise.

A possible approach to solving this problem may lie in strong assumptions about the noise level in the data, so that examples are only added to the window if their number exceeds a certain percentage of error on the remaining test set. However, we then face the problem of deciding which of the misclassified examples we should add to the window. Adding all of them (or a random subsample) will again unduely increase the noise level.

We are convinced that our approach, which shows a simple way for integrating a learning algorithm into windowing instead of only using it as a wrapper, is a first step on the right track. We believe that the ability of the separate-and-conquer strategy to separate regions of the instance space that have already been sufficiently well explained by some of the learned rules and to concentrate on covering the remaining portions of the instance space can form the basis of noise-tolerant windowing procedures. In fact, since the completion of the draft of this paper, we have been able to obtain very encouraging results in noisy domains with a windowing algorithm based on the ideas presented in this work. Instead of stopping to test a learned theory if enough exceptions have been found for the next window, the approach we take in (Fürnkranz 1997a) tests each rule on the entire remaining data set. This is more costly, but on the other hand allows to immediately identify rules that are consistent with the data and to remove all examples they cover. In order to achieve noise-tolerance, the algorithm attempts to recognize "reliable" instead of consistent rules.

Related Work

There have been several approaches that use subsampling algorithms that differ from windowing. For decision tree algorithms it has been proposed to use dynamic subsampling at each node in order to determine the optimal test. This idea has been originally proposed, but not evaluated in (Breiman *et al.* 1984). Catlett (1991b) has further explored this approach in his work on *peepholing*, which is a sophisticated procedure for using subsampling to eliminate unpromising attributes and thresholds from consideration.

Most closely related to windowing is *uncertainty sam-*

pling (Lewis & Catlett 1994). Here the new window is not selected on the basis of misclassified examples, but on the basis of the learner's confidence in the learned theory. The examples that are classified with the least confidence will be added to the training set in the next iteration.

Work on *partitioning*, i.e. splitting the example space into segments of equal size and combining the rules learned on each partition, has also produced promising results in noisy domains, but has substantially decreased learning accuracy in non-noisy domains (Domingos 1996). Besides, the technique seems to be tailored to a specific learning algorithm and not generally applicable.

Conclusion and Further Research

We have presented a re-evaluation for windowing using separate-and-conquer rule learning algorithms, which shows that for this type of algorithm significant gains in efficiency are possible. In particular, we have shown that separate-and-conquer algorithms allow a more flexible integration of windowing into the learning algorithm. However, the presented results are limited to redundant, noise-free domains. While we believe that the restriction to noise-free domains can be successfully tackled (Fürnkranz 1997a), it lies in the nature of windowing that it can only work successfully, if there is some redundancy in the domain. Methods for characterizing redundant domains are thus a rewarding topic for further research.

Our implementation currently is limited to 2-class problems and symbolic attributes. The costs for choosing thresholds, the standard procedure for handling numerical data, typically decrease with the number of training examples, which is why we hope that windowing will also be able to speed up learning in these cases. However, it is an open question, how the reduced number of possible thresholds in the learning window affects predictive accuracy. Another assumption made by our algorithm is that the training examples appear in random order or have been prerandomized. Otherwise, the ordering could maliciously effect the performance of the presented algorithms. Additional gains in efficiency could also be achieved by trying to specialize over-general rules, instead of entirely removing them. For this purpose we plan to adapt ideas from research in theory revision (Wrobel 1996).

Acknowledgements

This research is sponsored by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grant number P10489-MAT. Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian Federal Ministry of Science and Transport. I would like to thank Ray Mooney for making his Common Lisp ML library publicly available, which has been used for the implementation of our programs; Gerhard Widmer for his comments on an earlier version of this paper; the maintainers of and contributors to the UCI machine learning repository; and the three anonymous reviewers for valuable suggestions and pointers to relevant literature.

References

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth & Brooks.

Catlett, J. 1991a. Megainduction: A test flight. In Birnbaum, L., and Collins, G., eds., *Proceedings of the 8th International Workshop on Machine Learning (ML-91)*, 596–599. Evanston, IL: Morgan Kaufmann.

Catlett, J. 1991b. *Megainduction: Machine Learning on Very Large Databases*. Ph.D. Dissertation, Basser Department of Computer Science, University of Sydney.

Domingos, P. 1996. Efficient specific-to-general rule induction. In Simoudis, E., and Han, J., eds., *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 319–322. AAAI Press.

Fürnkranz, J. 1997a. Noise-tolerant windowing. Technical Report OEFAI-TR-97-07, Austrian Research Institute for Artificial Intelligence. Submitted to IJCAI-97.

Fürnkranz, J. 1997b. Separate-and-conquer rule learning. *Artificial Intelligence Review*. To appear.

Kivinen, J., and Mannila, H. 1994. The power of sampling in knowledge discovery. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-94)*, 77–85.

Lewis, D. D., and Catlett, J. 1994. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94)*. Morgan Kaufmann.

Quinlan, J. R. 1983. Learning efficient classification procedures and their application to chess end games. In Michalski, R. S.; Carbonell, J. G.; and Mitchell, T. M., eds., *Machine Learning. An Artificial Intelligence Approach*. Palo Alto, CA: Tioga. 463–482.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5:239–266.

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

Toivonen, H. 1996. Sampling large databases for association rules. In *Proceedings of the 22nd Conference on Very Large Data Bases (VLDB-96)*, 134–145.

Wirth, J., and Catlett, J. 1988. Experiments on the costs and benefits of windowing in ID3. In Laird, J., ed., *Proceedings of the 5th International Conference on Machine Learning (ML-88)*, 87–99. Ann Arbor, MI: Morgan Kaufmann.

Wrobel, S. 1996. First order theory refinement. In De Raedt, L., ed., *Advances in Inductive Logic Programming*. Amsterdam, Netherlands: IOS Press. 14–33.

Yang, Y. 1996. Sampling strategies and learning efficiency in text categorization. In Hearst, M., and Hirsh, H., eds., *Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access*, 88–95. AAAI Press. Technical Report SS-96-05.