# Constraint Based Incremental Learning of Classification Rules

Arkadiusz Wojna

Institute of Informatics, Warsaw University
ul. Banacha 2, 02-097 Warsaw, Poland
http://www.mimuw.edu.pl/~awojna
wojna@mimuw.edu.pl

**Abstract.** We present a modification of a simple incremental procedure maintaining the set of all current reduct rules. It reduces searching to the part of the rule space limited by a dynamic monotonic constraint. Efficiency problems and their solutions for the class of coverage based constraints are discussed and an illustrative example is provided.
**Keywords:** rough sets, machine learning, incremental learning, decision algorithms.

## 1 Introduction

In recent years rough sets were intensively studied as a method for approximative concept synthesis from data tables. Many data sources have dynamic character and their size is still increasing. In order to maintain the validity of knowledge extracted from dynamically changing data one should develop incremental learning strategies.

Incremental learning has been already widely studied in machine learning and for the exhaustive overview of these methods the reader is referred e.g. to [1] and [3]. This paper examines the problematics on the ground of rough sets introduced by Pawlak [5]. Different incremental algorithms maintaining reducts were proposed e.g. [4], [7] and experimental results comparing nonicremental and incremental methods for reduct generation may be found in [9].

The subject of the paper is an incremental method maintaining a set of reduct rules. Shan and Ziarko [7] described an algorithm generating all reduct rules. This paper presents its more practical version based on the notion of dynamic monotonic constraint that reduced the size of the rule space to be searched. The idea of searching for rules satisfing user requirements has been already used in nonincremental approach e.g. [2], [8]. Different properties and accelerating methods of the proposed solution are described and an experimental example that demonstrates potential advantages of the constraint based approach is provided.

## 2 Classification Rules and Constraints

We denote a finite set of binary attributes by $A$ and a finite set of decisions by $V$. The domain of all objects is defined by $U = \{0,1\}^A$. The input of an incremental algorithm is a finite sequence of pairs $(u_i, d_i)$ called a *sample*, where $u_i \in U$ is an object and $d_i \in V$ is a decision for $u_i$. The notion of a sample corresponds to the notion of a decision table [5] in nonincremental approach. For a given sample $s$ we denote the set of all examples from $s$ with a decision $d$ by $Class_s(d)$.

A *classification rule* is an implication $\alpha \Rightarrow d$ where $\alpha$ is a conjunction of literals of attributes from $A$ and $d \in V$. The support of a sample $s$ for a conjunction $\alpha$ is defined by $[\alpha]_s = \{(u, d) \in s : u \text{ satisfies } \alpha\}$ and for a rule $\alpha \Rightarrow d$ is defined by $[\alpha \Rightarrow d]_s = [\alpha]_s \cap Class_s(d)$.

A rule $\alpha \Rightarrow d$ is *certain* for a sample $s$ if for each pair $(u_i, d_i)$ in $s$ such that $u_i$ satisfies $\alpha$ the decisions are equal $d_i = d$. A certain rule $\alpha \Rightarrow d$ is a *reduct rule* if $\alpha$ is a minimal conjunction in the sense of literal set inclusion among all conjunctions occurring on the lefthand side of a certain rule with the same decision $d$. The set of all reduct rules with the decision $d$ for a sample $s$ is denoted by $RedRul_s(d)$. We use two measures for rules: *confidence* and *coverage* [2], [5], [6], [8]:

$$confidence_s(\alpha \Rightarrow d) = \begin{cases} 0 & \text{if } [\alpha]_s = \emptyset \\ \frac{\|[\alpha \Rightarrow d]_s\|}{\|[\alpha]_s\|} & \text{if } [\alpha]_s \neq \emptyset \end{cases}$$

$$coverage_s(\alpha \Rightarrow d) = \frac{\|[\alpha \Rightarrow d]_s\|}{\|Class_s(d)\|}$$

Usually the set of all reduct rules is very large and only a small subset, that can be described by a monotonic constraint, is relevant. A monotonic constraint is a set of rules $C$ such that if a rule $\alpha \Rightarrow d$ belongs to $C$ then for each $B \subseteq Literals(\alpha)$ the rule $\bigwedge B \Rightarrow d$ also belongs to $C$. We restrict the space of reduct rules to bounded by $C$: $RedRul_s^C(d) = C \cap RedRul_s(d)$. Throughout the paper, somewhat informally, we denote the description of a monotonic constraint and the set of rules defined by the monotonic constraint with the same symbol $C$.

In the next sections we focus our attention on two types of a monotonic constraint: the first one $RedRul_s^{coverage>\theta}(d)$ bases on a fixed coverage threshold $\theta \in [0, 1]$:

$$\{r \in RedRul_s(d) : coverage_s(r) > \theta\}$$

and the other one $RedRul_s^{best-k}(d)$ includes always the set of exactly $k$ best reduct rules:

$$\{r \in RedRul_s(d) : \|\{r' \in RedRul_s(d) : coverage_s(r') > coverage_s(r)\}\| < k\}$$

## 3 Incremental Constraint Based Algorithm

The algorithm [7] computing all reduct rules starts with the set of the most general rules one for each decision class and after each new example is added it extends each rule that is inconsistent with the example by adding the literals excluding the example.

Since the space of rules is usually too large for searching for all reduct rules, we propose a modified version of the incremental algorithm using a dynamic monotonic constraint that may change after each new example is added. The algorithm limits the set of maintained reduct rules to rules satisfying the constraint. Let $C$ denote the considered monotonic constraint. During computation the algorithm always maintains the following sets: $s$ — the set of training examples, $Rules(d)$ — the set of reduct rules with the decision $d$, $CCand(d)$ — the set of candidates for reduct rules with the decision $d$ satisfying the constraint $C$ and $nonCCand(d)$ — the set of candidates for reduct rules with the decision $d$ not satisfying $C$.

Like in [7] the algorithm starts with the set of the most general rules one for each decision class and for each new example it executes procedure *learn*. The difference is that the constraint based algorithm extends candidates only from the sets $CCand$ set leaving the sets $nonCCand$ unchanged:

```
Algorithm 1 learn(u,d)
$s := s + (u, d)$;
update the constraint $C$;
for each $d' \in V$ do
step 1:
  move all rules $r \in Rules(d')$ such that $r \notin C$ to $nonCCand(d')$;
  move all rules $r \in Rules(d')$ inconsistent with $(u, d)$ to $CCand(d')$;
  move all certain rules $r \in nonCCand(d')$ such that $r \in C$ to $Rules(d')$;
  move all rules $r \in nonCCand(d')$ such that $r \in C$ to $CCand(d')$;
step 2:
  while $CCand(d') \neq \emptyset$ do
    remove an arbitrary rule $\alpha \Rightarrow d'$ from $CCand(d')$;
    find an example $(u'', d'')$ inconsistent with the rule $\alpha \Rightarrow d'$;
    for each attribute $a \in A \setminus Attributes(\alpha)$ do
      $l$ :=literal for $a$ which excludes $u''$;
      if $\alpha \wedge l \Rightarrow d'$ is not subsumed
            by another rule from $Rules(d') \cup CCand(d') \cup nonCCand(d')$ then
        if $\alpha \wedge l \Rightarrow d' \notin C$ then $nonCCand(d') := nonCCand(d') \cup \{\alpha \wedge l \Rightarrow d'\}$
        else if $\alpha \wedge l \Rightarrow d'$ is certain then $Rules(d') := Rules(d') \cup \{\alpha \wedge l \Rightarrow d'\}$
        else $CCand(d') := CCand(d') \cup \{\alpha \wedge l \Rightarrow d'\}$;
```

At the beginning of the procedure $learn(u, d)$ the sets $Rules(d')$ are assumed to contain all reduct rules satisfying the constraint $C$ and $nonCCand(d')$ are assumed to contain all generated up to now rules not satisfying $C$, both according to a sample $s$ before adding a new example $(u, d)$. The sets $CCand(d')$ should be empty.

In the step 1 the procedure moves rules according to changes in the sample $s$ and the constraint $C$: reduct rules for a previous sample may be inconsistent with a new example $(u, d)$ and the modified constraint may both include new candidate and reduct rules and exclude previously covered reduct rules. Time needed for this step may vary significantly in dependence on a used constraint.

For the constraint *coverage* $> 0$ migration only for rules that cover a new object $u$ is possible, for constraints with positive coverage threshold other rules with the decision $d$ can migrate and for $best - k$ constraints checking constraint satisfiability becomes much more complex. In the last case a good solution is to assume the ranking based on the current set of reduct rules and do the step 2 correcting the ranking every time when a new reduct rule is found.

In the step 2 the procedure extends all candidates satisfing $C$. Candidates that were previously in $nonCCand(d')$ may be inconsistent with any example in the sample $s$, not always with the last one $(d, u)$. Therefore the procedure must search the sample $s$ for an inconsistent example. In order to avoid searching the whole sample for each candidate the procedure may assign to each extended rule $\alpha \wedge l \Rightarrow d'$ the position in $s$ where an inconsistent example for the previous rule $\alpha \Rightarrow d'$ was found and continue searching from this place. After an inconsistent example $(u'', d'')$ is found, the candidate is extended with all literals excluding $u''$. The next time consuming operation is subsumption checking. If an extension is not subsumed by another rule it is directed to the appropriate set, otherwise it is removed.

**Theorem 1.** *At the end of the procedure learn the union* $\bigcup_{d \in V} Rules(d)$ *is always equal to the set of all reduct rules satisfying the constraint* $C$ *for the sample* $s$.

## 4 Improving Efficiency

One of the properties of the algorithm presented in the previous section is that it never reduces rules. Generating more and more new rules without any reduction prolongs checking for subsumptions and leads to the lack of memory. In order to avoid the problem the following solution may be used. Every time after a rule $\alpha \Rightarrow d'$ is added to the set $nonCCand(d')$ it is also reduced as much as it is possible:

```
Algorithm 2  reduce($\alpha \Rightarrow d'$)
reduce the rule $\alpha \Rightarrow d'$ to $\beta \Rightarrow d'$
    where $\beta$ is any minimal conjunction subsuming $\alpha$ such that $\beta \Rightarrow d' \notin C$;
```

The presented improvement applies to constraints that have "shrinking" property what means that new examples may lead to excluding a rule from a constraint. An example of a "shrinking" constraint is *coverage* $> \theta$ for any $\theta > 0$, whereas the constraint *coverage* $> 0$ does not have this property.

However, this modification brings another undesirable phenomenon affecting efficiency namely "shimmering" of rules what means that a single rule may be generated and reduced many times while the constraint is changing dynamically and repeated computation of rule parameters significantly slowers the performance. We present two methods to deal with this problem.

The first one consists in maintaining two buffers: $BufExt$ saves rules for which the extending operation was already performed and $BufRed$ saves rules that were reduced. The buffers are usually too limited for keeping all rules that appeared in the process of learning. Therefore a certain measure is applied to estimate which rules are the most probable to be reused in the near future. For coverage based constraints coverage is a good measure for it. The following procedure *saveExtended* is executed each time when a rule is extended:

**Algorithm 3** *saveExtended(r)*

```
if the buffer $BufExt$ is not full then add r to $BufExt$;
else if $coverage_s(r) < \max_{r' \in BufExt} coverage_s(r')$
    then replace a rule with the maximal coverage in $BufExt$ with r ;
```

The analogical procedure *saveReduced* is executed when a rule is reduced:

**Algorithm 4** *saveReduced(r)*

```
if the buffer $BufRed$ is not full then add r to $BufRed$;
else if $coverage_s(r) > \min_{r' \in BufRed} coverage_s(r')$
    then replace a rule with the minimal coverage in $BufRed$ with r ;
```

When the procedure *learn* needs to compute parameters for a new generated or reduced rule first it checks whether the rule is still available in the corresponding buffer.

Another solution that reduces "shimmering" is grouping examples. Instead of learning each new example separately first the algorithm gets a large group of examples and then starts learning rules. The learning process for a group of examples may last much longer than for a single example. However, notice that the procedure *learn* may be easily split into two parts: the first one corrects the contents of the maintained sets and the parameters of rules according to the sample including a new group and the next one generates new rules. The first part is always short hence the second one is critical for time performance. Therefore a good assumption for the second part is to be ready to stop learning and classify a new object with a current set of rules every time when the classification procedure is called. It requires from the algorithm to use rules with confidence less than 1 for classification. In this proposition a strategy of choosing rules for extension is important. The higher confidence a rule saves after updating by a new group of examples the more reliable it is for the classification procedure. Therefore a good strategy is to start extending with a rule having confidence nearest to 1 and move towards rules with lower confidence. In this way more reliable rules are adapted to a new group first. The latter solution provides also a good background for distributed computation.

The presented algorithm may be also adapted to the case when it is given a very large set of examples $s$ at once. Like in the incremental algorithm it executes the procedure *learn* for successive examples in $s$. Because of the size the computation for the whole sample would last very long and would block classification procedure calls. To avoid it the learning procedure is always stopped when the classification procedure is called and waits until the classification is completed. Classification uses a current set of computed rules. Many of them may be still inconsistent with a number of examples, therefore before classification the algorithm needs to calculate qualitative parameters of rules: confidence and coverage, according to the whole sample $s$. It imposes the additional condition that a used classifier accepts rules with confidence less than 1.

Computing parameters for a set of rules consumes much less time than generating this set but computing them every time when the classification procedure is called is usually still too expensive for a large set of rules and a large set of objects. In order to avoid the problem the algorithm may perform the following operations. For a particular object to classify it may compute parameters only for rules covering the object. Once computed parameters for a rule may be preserved as long as the rule is held in the corresponding union $Rules(d) \cup Cand(d)$. Independently of classification procedure calls the learning procedure may stop at regular intervals and compute parameters for rules generated since the previous stop. The choice of appropriate data structures may significantly accelerate computation of parameters for rules and objects.

In case when all methods of improving efficiency fail, the exhaustive search may be replaced immediately by any heuristic search.

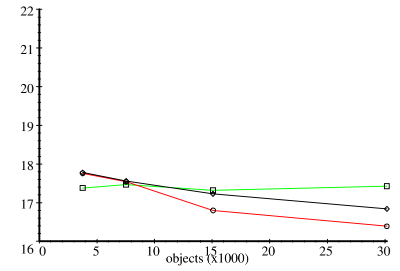
## 5 Illustrative Example

We present experimental results for the data set Income (13 attributes, 30162 training cases, 15060 testing cases) from the repository at University of California, Irvine (http://kdd.ics.uci.edu). In preprocessing discretization was used and 32 binary attributes were chosen by greedy heuristic algorithm optimizing discernibility.

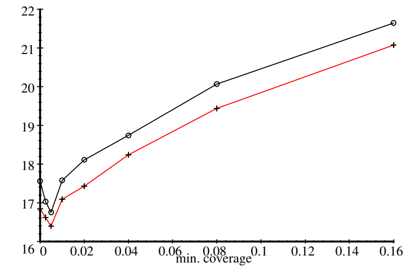
The learning procedure was executed for groups of examples. We used the incremental constraint based algorithm with the modification that rules were extended not in all possible directions but only with rules that have the best confidence and the best coverage if there are ties in confidence for at least one covered object.

For each coverage threshold 0, 0.05 and 0.2 we performed series of computation in the following way. First the procedure *learn* was executed for the first 1/8 part of the training set and the testing set was classified. In each next step the number of examples equal to the number of examples received in all previous steps was added and learned and the next test of the testing set was performed. In this way the size of successive groups of examples grew exponentially. Each test object was classified with the decision of the best covering rule in the union $\bigcup_{d \in V} Rules(d) \cup nonCCand(d)$ according to the confidence and in case of ties to the coverage.

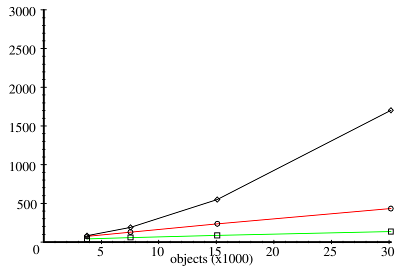
The results are presented on the graphs below. Left side graphs present the classification error, time and number of rules obtained in three series of incremental learning with different constraints: $coverage > 0.2$ (light line with boxes), $coverage > 0.05$ (medium dark line with circles) and $coverage > 0$ (dark line with diamonds). Right side graphs present the final results of incremental (medium dark line with crosses) and nonincremental (dark line with circles) learning for different coverage based constraints.



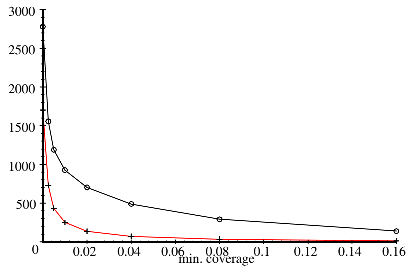
Income - classification error (%)



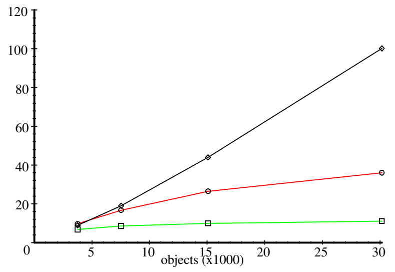
Income - classification error (%)



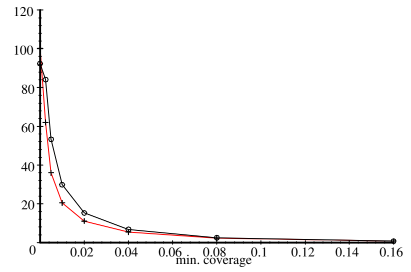
Income - time (sec)



Income - time (sec)



Income - rules (×1000)



Income - rules (×1000)

In the presented example the application of stronger constraints brought a significant reduction of used memory and time and very small deterioration of accuracy or even improvement for low coverage thresholds. The results show also that accuracy obtained with the small part of the training set used in a learning process is not significantly lower than for the whole training set and finally incremental learning reached better results than nonincremental. Similar properties of test results on other data sets (Shuttle, Letter) indicate that the combination of the incremental approach and a coverage based constraint may be an effective tool for learning concepts from both dynamic and large data sets.

## 6 Conclusions

We have shown how rough set methods can be adapted to dynamically changing data. We proposed a method based on a special type of monotonic constraints that allowed us to reduce searching in the space of rules without substantial changes in the classification quality. The presented method may be adapted to large data sets especially when one implements it using cluster of computers. The experimental example indicates that the incremental approach may preserve all advantages of nonincremental methods and add new ones like reduction in used time and memory and continuous improvement.

The following related problems are the subject for future study: methods for coding arbitrary attributes by binary ones e.g. by discretization or value grouping and efficient methods for computing confidence and coverage for large rule sets because this is the most time consuming operation.

**Acknowledgement** The author is grateful to professor Andrzej Skowron for useful remarks on this presentation. This work was supported by Research Program of European Union, CRIT 2 Esprit Project No. 20288 and grants 8 T11C 025 19 and 8 T11C 009 19 from the Polish National Committee for Scientific Research.

## References

1. P. Langley, *Elements of machine learning*, The MIT Press, 1996
2. M. Kryszkiewicz, H. Rybiński, *Knowledge discovery from large databases using rough sets*, in: Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, Vol. 1, 85-89.
3. R. Michalski, *A theory and methodology of inductive learning*, Machine Learning: An Artificial Intelligence Approach, Tioga, 1983, Vol. 1, 83-134.
4. M. Orłowska, M. Orłowski, *Maintenance of knowledge in dynamic information systems*, in: R. Słowiński (editor), Intelligent decision support - handbook of applications and advances of the rough sets theory, Kluwer Academic Publishers, Dordrecht, 1992, 315-330.
5. Z. Pawlak, *Rough sets - theoretical aspects of reasoning about data*, Kluwer Academic Publishers, Dordrecht, 1991.
6. L. Polkowski, A. Skowron (editors), *Rough sets in knowledge discovery*, Physica-Verlag, Heidelberg, 1998.
7. N. Shan, W. Ziarko, *Data-based acquisition and incremental modification of classification rules*, Computational Intelligence, 11(2), 1995, 357-370.
8. J. Stefanowski, *On rough set based approaches to induction of decision rules*, in: L. Polkowski, A. Skowron (editors), Rough sets in knowledge discovery 1, Physica-Verlag, Heidelberg, 1998, 500-529.
9. R. Susmaga, *Experiments in incremental computation of reducts*, in: L. Polkowski, A. Skowron (editors), Rough sets in knowledge discovery 1, Physica-Verlag, Heidelberg, 1998, 530-553.