Pre-Processing by a Cost-Sensitive Literal Reduction Algorithm

Lavrac, N.; Gamberger, D.; Turney, Peter

Preprocessing by a cost-sensitive literal reduction algorithm: REDUCE [1]

Nada Lavrač

J. Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia

Dragan Gamberger

Rudjer Bošković Institute
Bijenička 54, 10000 Zagreb, Croatia

Peter Turney

Institute for Information Technology
National Research Council Canada
M-50 Montreal Road, Ottawa, Ontario, Canada, K1A 0R6

Abstract

This study is concerned with whether it is possible to detect what information contained in the training data and background knowledge is relevant for solving the learning problem, and whether irrelevant information can be eliminated in preprocessing before starting the learning process. A case study of data preprocessing for a hybrid genetic algorithm shows that the elimination of irrelevant features can substantially improve the efficiency of learning. In addition, cost-sensitive feature elimination can be effective for reducing costs of induced hypotheses.

1 Introduction

The problem of relevance was addressed in early research on inductive concept learning [11]. Recently, this problem has also attracted much attention in the context of feature selection in attribute-value learning [1,5,14]. Basically one can say that all learners are concerned with the selection of 'good' literals or features which will be used to construct the hypothesis.

This study is concerned with whether it is possible to detect what information contained in the training data and background knowledge is relevant for solving the learning problem,

[1] ISSEK Workshop *Mathematical and Statistical Methods in AI*, September 19–21, 1996, Udine, Italy

and whether irrelevant information can be eliminated in preprocessing before learning. An important difference between our approach and most other approaches is that, when deciding about the relevance of literals, we are concerned with finding 'globally relevant' literals w.r.t. the entire set of training examples, as opposed to finding the 'good literals' in the given local training set (i.e., a set of examples covered by the currently constructed rule in rule induction systems, or a set of covered examples in the current node of a decision tree in TDIDT systems). This is important since the elimination of globally irrelevant literals guarantees that literal elimination will not harm the hypothesis formation process and that during the reduction of the hypothesis space the optimal problem solution will not be discarded. The aim of this study is to distinguish between a set of literals that are relevant for learning and a set of irrelevant literals that can be discarded before learning (i.e., before even entering the 'good literal' competition). Such filtering of irrelevant literals can thus be viewed as a part of preprocessing of the set of training examples.

This paper presents a case study of data preprocessing for a hybrid genetic algorithm which shows that the elimination of irrelevant features can substantially improve the efficiency of learning. In addition, cost-sensitive feature elimination can be effective for reducing costs of induced hypotheses.

The paper is organized as follows: Section 2 introduces the representational formalism, the so-called p/n pairs of examples, gives the definition of irrelevant literals and presents a theorem which is the basis for literal elimination. Section 3 presents the cost-sensitive literal elimination algorithm REDUCE. Section 4 introduces the problem domain, the 20 and the 24 trains East-West Challenges, and presents the results of our experiments that show that the performance of a hybrid genetic algorithm RL-ICET [16] can be significantly improved by applying REDUCE in preprocessing of the dataset. The results of RL-ICET are also compared to those of C4.5 [13].

2 Relevance of literals and features

Consider a two-class learning problem where training set E consists of positive and negative examples of a concept, and examples $e \in E$ are tuples of truth-values of terms in a hypothesis language. The set of all terms, called *literals*, is denoted by L.

2.1 Representation of training examples

Let us represent the training set E as a table where rows correspond to training examples and columns correspond to literals. An element in the table has the value *true* when the example satisfies the condition (literal) in the column of the table, otherwise its value is $false$. If the training set does not have the form of tuples of truth-values, a transformation to this form is performed in preprocessing of the training set.

2.1.1 Learning of propositional descriptions

In the attribute-value learning setting, the transformation procedure is based on analysis of the values of examples in the training set. For each attribute A_i, let v_{ix} $(x = 1..k_{ip})$ be the k_{ip} different values of the attribute that appear in the positive examples and let w_{iy} $(y = 1..k_{in})$ be the k_{in} different values appearing in the negative examples. The transformation results in a set of literals L:

- For discrete attributes A_i, literals of the form $A_i = v_{ix}$ and $A_i \neq w_{iy}$ are generated.
- For continuous attributes A_i, literals of the form $A_i \leq (v_{ix} + w_{iy})/2$ are created for all neighboring value pairs (v_{ix}, w_{iy}), and literals literals $A_i > (v_{ix} + w_{iy})/2$ for all neighboring pairs (w_{iy}, v_{ix}). The motivation is similar to that suggested in [2].
- For integer valued attributes A_i, literals are generated as if A_i were both discrete and continuous, resulting in literals of four different forms: $A_i \leq (v_{ix} + w_{iy})/2$, $A_i > (v_{ix} + w_{iy})/2$, $A_i = v_{ix}$, and $A_i \neq w_{iy}$.

2.1.2 Inductive logic programming

Inductive logic programming (ILP) refers to first-order learning of relational descriptions in the representation formalism of logic programs [7]. In this setting, a LINUS transformation approach is assumed that is appropriate for a limited hypothesis language of constrained nonrecursive clauses [6,7]. For example, if the training examples about the target relation $daughter(A_1, A_2)$ are given in the training set, and the background knowledge consists of the definitions of a unary relation $female$ and binary relation $parent$, the transformation of training examples results in a matrix of binary values $true$ and $false$ whose rows correspond to training examples, and columns correspond to the following literals: $female(A_1)$, $female(A_2)$, $parent(A_1, A_2)$, $parent(A_2, A_1)$, $parent(A_1, A_1)$, $parent(A_2, A_2)$, and $A_1 = A_2$.

2.2 p/n pairs of examples and relevance of literals

Assume the set of training examples E represented by a truth-value table where columns correspond to the set of literals L, and rows are tuples of truth-values of literals, representing training examples e_i. The table is divided in two parts, P and N, where P are the positive examples, and N are the negative examples. We use $P \cup N$ to denote the table E.

To enable a formal discussion of the relevance of literals, the following definitions are introduced:

Definition 1. *A p/n pair is a pair of training examples where $p \in P$ and $n \in N$.*

Definition 2. *Literal $l \in L$ covers a p/n pair if in column l of the table of training examples E the positive example p has value true and the negative example n has value $false$. The set of all p/n pairs covered by literal l will be denoted by $E(l)$.*

Definition 3. *Literal l covers literal l' if $E(l') \subseteq E(l)$.*

To illustrate the above definitions consider a simple learning problem with 5 training examples: three positive p_1, p_2 and p_3, and two negative n_1 and n_2, described by truth-values of literals $l_i \in L$. The truth-value matrix E, showing just some of the truth-values, is given in Table 1.

Examples		*Literals*						
		... ...	l_2	... ...	l_4	... ...	l_8	... ...
	p_1							
P	p_2		*true*		*true*		*false*	
	p_3							
	n_1		*false*		*false*		*true*	
N	n_2		*false*		*true*		*true*	

Table 1: Coverage of literals, coverage of p/n pairs.

Literal l_2 in Table 1 seems to be relevant for the formation of the inductive hypothesis since it is true for a positive example and false for both negative examples. This is due to the fact that l_2 covers a positive example, and does not cover the negative examples, and is thus a reasonable ingredient of the hypothesis that should cover the positive examples and should not cover the negatives examples.

Literal l_2 covers two p/n pairs: $E(l_2) = \{p_2/n_1, p_2/n_2\}$. Literal l_8 is inappropriate for constructing a hypothesis, since it does not cover any p/n pair: $E(l_8) = \emptyset$. Literal l_4 seems to be less relevant than l_2 and more relevant than l_8; it covers only one p/n pair: $E(l_4) = \{p_2/n_1\}$. Literal l_2 covers l_4 and l_8, and literal l_4 covers l_8, since $E(l_8) \subseteq E(l_4) \subseteq E(l_2)$.

Table 1 thus gives the following intuition: the more p/n pairs a literal covers the more relevant it is for hypothesis formation. This may be formalized by the next definition.

Definition 4a. *Literal l' is irrelevant if there exists a literal $l \in L$ such that l covers l' ($E(l') \subseteq E(l)$). In other words, literal l' is irrelevant if it covers a subset of p/n pairs covered by some other literal $l \in L$.*

Assume that literals are assigned costs (for instance, cost can be a measure of complexity - the more complex the literal, the higher its cost). Let $c(l)$ denote the cost of literal $l \in L$. The definition of irrelevance needs to be modified to take into account the cost of the literals.

Definition 4b. *Literal l' is irrelevant if there exists a literal $l \in L$ such that l covers l' ($E(l') \subseteq E(l)$) and the cost of l is lower than the cost of l' ($c(l) \leq c(l')$).*

Our claim is that irrelevant literals can be eliminated in preprocessing. This claim is based on the following theorem, which assumes that the hypothesis language $\mathcal{L}$ is rich enough to allow for a complete and consistent hypothesis H to be induced from the set of training examples E.[2]

Theorem 1. Assume a training set E and a set of literals L such that a complete and consistent hypothesis H can be found. Let $L' \subseteq L$. A complete and consistent hypothesis H can be found using only literals from the set L' if and only if for each possible p/n pair from the training set E there exists at least one literal $l \in L'$ that covers the p/n pair.

Proof of necessity: Suppose that the negation of the conclusion holds, i.e., that a p/n pair exists that is not covered by any literal $l \in L'$. Then no rule built of literals from L' will be able to distinguish between these two examples. Consequently, a description which is both complete and consistent can not be found.

Proof of sufficiency: Take a positive example p_i. Select from L' the subset of all literals L_i that cover p_i. A constructive proof of sufficiency can now be presented, based on k runs of a covering algorithm, where k is the cardinality of the set of positive examples ($k = |P|$). In the i-th run, the algorithm learns a conjunctive description h_i, $h_i = l_{i,1} \wedge \ldots \wedge l_{i,m}$ for all $l_{i,1}, \ldots l_{i,m} \in L_i$ that are true for p_i. Each h_i will thus be *true* for p_i (h_i covers p_i), and $false$ for all $n \in N$. After having formed all the k descriptions h_i, a resulting complete and consistent hypothesis can be constructed: $H = h_1 \vee \ldots \vee h_k$.

□

The importance of the theorem is manifold. First, it points out that when deciding about the relevance of literals it will be significant to detect which p/n pairs are covered by the literal. Second, the theorem enables us to directly detect useless literals that do not cover any p/n pair. This theorem is the basis of the REDUCE algorithm for literal elimination.

3 Cost-sensitive literal elimination

3.1 Cost-sensitive literal elimination algorithm REDUCE

Algorithm 1 implements the cost-sensitive literal elimination algorithm, initially developed within the ILLM learner [3]. This algorithm is the core of REDUCE [9].

The complexity of Algorithm 1 is $\mathcal{O}(|L|^2 \times |E|)$, where $|L|$ is the number of literals and $|E|$ is the number of examples. This algorithm can be easily transformed into an iterative algorithm that can be used during the process of generation of literals [8].

[2]Hypothesis H is complete if it covers all the positive examples $p \in P$. Hypothesis H is consistent if it does not cover any negative example $n \in N$.

Algorithm 1. Cost-sensitive literal elimination

```
Given: $CL$ – costs of literals in $L$
Input: $P$, $N$ – tables of positive and negative examples, $L$ – set of literals
    $RP \leftarrow P$, $RN \leftarrow N$, $RL \leftarrow L$
    for $\forall\ l_i \in RL$ ($i \in [1, |L|]$) do
        if $l_i$ has value false for all rows of $RP$ then
            eliminate $l_i$ from $RL$
            eliminate column $l_i$ from $RP$ and $RN$ tables
        if $l_i$ has value true for all rows of $RN$ then
            eliminate $l_i$ from $RL$
            eliminate column $l_i$ from $RP$ and $RN$ tables
        if $l_i$ is covered by any $l_j \in RL$ for which $c(l_j) \le c(l_i)$ then
            eliminate $l_i$ from $RL$
            eliminate column $l_i$ from $RP$ and $RN$ tables
    endfor
Output: $RP$, $RN$ – reduced tables of positive and negative examples, $RL$ – reduced set of
    literals
```

The algorithm can be efficiently implemented using simple bitstring manipulation on the table of training examples E. For this purpose, the table E is transformed into E_t as follows:

$\forall p \in P$: replace $true$ by 1 and $false$ by 0

$\forall n \in N$: replace $false$ by 1 and $true$ by 0

In this representation, examples $e \in E_t$ and literals $l \in L$ are bitstrings. Coverage can now be checked by set inclusion. Recall that literal l covers literal l' if $E_t(l') \subseteq E_t(l)$. Thus, if l' has value 1 only in (some of) those rows as l has 1 and in no other rows, l' can be eliminated.

3.2 Relevance of features

The term *feature* is used to denote positive literals such as for example $A_i = v$, $A_j \le w$, and $r(A_i, A_j)$. In the hypothesis language, the existence of one such feature implies the existence of two complementary literals: a positive and a negative literal. Suppose that we consider the feature $Color = black$ and that the attribute $Color$ has three possible values: $black$, $white$, red. Since each feature implies the existence of two literals, the necessary and sufficient condition that a feature can be eliminated as irrelevant is that both of its literals $Color = black$ and $Color \neq black$[3] are irrelevant. This statement directly implies the procedure taken in our experiment. First we convert the starting feature vector to the corresponding literal vector which has twice as many elements. After that, we eliminate the irrelevant literals and, in the third step, we construct the reduced set of features which includes all the features which have at least one of their literals in the reduced literal vector.

It must be noted that direct detection of irrelevant features (without conversion to and from the literal form) is not possible except in the trivial case where two (or more) features have identical columns in table E_t. Only in this case a feature f exists whose literals f and $\neg f$ cover both literals g and $\neg g$ of some other feature. In a general case if a literal of feature f

[3]We use either the notation $\neg(Color = black)$ or $Color \neq black$ to denote a negative literal.

covers some literal of feature g then the other literal of feature g is not covered by the other literal of feature f. But it can happen that this other literal of feature g is covered by a literal of some other feature h. This means that although there does not exist a feature f that covers both literals of the feature g, feature g can be irrelevant.[4]

4 Utility study: The East-West challenge

Michie et al. [12] issued a "challenge to the international computing community" to discover low size-complexity Prolog programs for classifying trains as Eastbound or Westbound. The challenge was inspired by a problem posed by Michalski and Larson [10].

The original challenge [12] included three separate tasks. Michie later issued a second challenge, involving a fourth task. Our experiments described here involve the first and fourth tasks. The first task was to discover a simple rule for distinguishing 20 trains, 10 Eastbound and 10 Westbound, whereas the fourth task involved 24 trains, 12 Eastbound and 12 Westbound. In these two tasks, the set of trains was classified into East and West using an arbitrary human-generated rule (theory). The challenge was to discover the humen-generated theorys or a simpler theory (rule).

For both tasks, the winner was decided by representing the rule as a Prolog program and measuring its size-complexity. The size-complexity of the Prolog program was calculated as the sum of the number of clause occurrences, the number of term occurrences, and the number of atom occurrences.

4.1 RL-ICET

A cost-sensitive algorithm ICET was developed for generating low-cost decision trees [15]. ICET is a hybrid of a genetic algorithm and a decision tree induction algorithm. The genetic algorithm is Grefenstette's GENESIS [4] and the decision tree induction algorithm is Quinlan's C4.5 [13]. ICET uses a two-tiered search strategy. On the bottom tier, C4.5 uses a TDIDT (Top Down Induction of Decision Trees) strategy to search through the space of decision trees. On the top tier, GENESIS uses a genetic algorithm to search through the space of biases.

ICET takes feature vectors as input and generates decision trees as output. The algorithm is sensitive to both the cost of features and the cost of classification errors. The East-West Challenge involves data in the form of relations, and theories in the form of Prolog programs. For the East-West Challenge, ICET was extended to handle Prolog input. This algorithm is called RL-ICET (Relational Learning with ICET) [16].

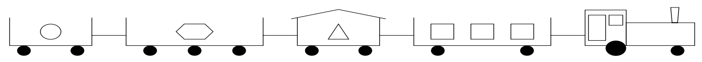
[4]This analysis helps us to see that the standard approach to rule construction which is based on feature selection is sub-optimal. Most rule learners use a two-phase approach: first, the best feature is selected, and second, for a selected feature, one of the two complementary literals (positive or negative) is used to construct a rule. We suggest that learning should be based on tuples of truth-values of positive and negative literals, rather than on feature vectors.

RL-ICET is similar to the LINUS learning system [6,7] since it uses a three-part learning strategy. First, a preprocessor translates the Prolog relations and predicates into a feature vector format. The preprocessor in RL-ICET was designed specially for the East-West Challenge, whereas LINUS has a general-purpose preprocessor. Second, an attribute-value learner applies a decision tree induction algorithm (ICET) to the feature vectors. Each feature is assigned a cost, based on the size of the fragment of Prolog code that represents the corresponding predicate or relation. A decision tree that has a low cost corresponds (roughly) to a Prolog program that has a low size-complexity. When it searches for a low cost decision tree, ICET is in effect searching for a low size-complexity Prolog program. Third, a postprocessor translates the decision tree into a Prolog program. Postprocessing with RL-ICET is done manually, whereas LINUS performs post-processing automatically.

4.2 Feature construction in RL-ICET

Much of the success of RL-ICET in the East-West challenge tasks may be attributed to its preprocessor which translates the Prolog descriptions of the trains into a feature vector representation.

The data about each train in the East-West challenge were represented using Prolog. For example, the first train, shown below, is represented by the following Prolog clause:



```
eastbound([c(1, rectangle, short, not_double, none, 2, l(circle,1)),
   c(2, rectangle, long, not_double, none, 3, l(hexagon, 1)),
   c(3, rectangle, short, not_double, peaked, 2, l(triangle, 1)),
   c(4, rectangle, long, not_double, none, 2, l(rectangle, 3))]).
```

The relatively compact Prolog description was converted by a simple Prolog program into a feature vector format (tuples of truth-values of features) to be used for decision tree induction. This resulted in rather large feature vectors of 1199 elements. The large vectors were required to ensure that all the features that are potentially interesting for the final solution are made available for ICET.

What follows is a brief outline of the feature construction procedure that occurs in the preprocessing of the training set. We started with 28 predicates that apply to the cars in a train, such as `ellipse(C)`, which is true when the car `C` has an elliptical shape. For each of these 28 predicates, we defined a corresponding feature. All of the features were defined for whole trains, rather than single cars, since the problem is to classify trains. The feature `ellipse`, for example, has the value *true* when a given train has a car with an elliptical shape. Otherwise `ellipse` has the value *false*. We then defined features by forming all possible unordered pairs of the original 28 predicates. For example, the feature `ellipse_triangle_load` has the value *true* when a given train has a car with an elliptical shape that is carrying a triangle load, and *false* otherwise. Note that the features `ellipse` and `triangle_load` may have the value *true* for a given train while the feature `ellipse_triangle_load` has the value *false*, since `ellipse_triangle_load` only has the value *true* when the train has a car that is both elliptical and carrying a triangle load. Next we defined features by forming all possible ordered pairs of the original 28 predicates, using the relation `infront(T, C1, C2)`. For example, the feature `u_shaped_infront_peaked_roof` has the value *true* when the train has a U-shaped car in front of a car with a peaked roof, and *false* otherwise. Finally, we added 9 more predicates that apply to the train as a whole, such as `train_4`, which has the value *true* when the train has exactly four cars. Thus a train is represented by a feature vector, where every feature has either the value *true* or the value *false*.

Each feature was assigned a cost, based on the complexity of the fragment of Prolog code required to represent the given feature. The complexity of a Prolog program is defined as a sum of the number of clause occurrences, the number of term occurrences and the number of atom occurrences. Table 2 shows some examples of constructed features and their costs.

Feature	*Prolog Fragment*	*Cost (Complexity)*
ellipse	$has\_car(T,C), ellipse(C).$	5
short_closed	$has\_car(T,C), short(C), closed(C).$	7
train_4	$len1(T,4).$	3
train_hexagon	$has\_load1(T, hexagon).$	3
ellipse_peaked_roof	$has\_car(T,C), ellipse(C), arg(5,C,peaked).$	9
u_shaped_no_load	$has\_car(T,C), u\_shaped(C), has\_load(C,0).$	8
rectangle_load_infront _jagged_roof	$infront(T,C1,C2), has\_load0(C1, rectangle), arg(5,C2,jagged).$	11

Table 2: Examples of features and their costs.

The feature vector for a train does not capture all the information that is in the original Prolog representation. For example, we could also define features by combining all possible unordered triples of the 28 predicates. However, these features would likely be less useful, since they are so specific that they will only rarely have the value *true*. If the target concept should happen to be a triple of predicates, it could be closely approximated by the conjunction of the three pairs of predicates that are subsets of the triple.[5]

4.3 Previous results of RL-ICET

RL-ICET was the winning algorithm for the second task in the first East-West Challenge, and it performed very well in the other three tasks.

[5]This kind of translation to feature vector representation could be applied to many other types of structured objects. For example, consider the problem of classifying a set of documents. The keywords in a document are analogous to the cars in a train. The distance between keywords or the order of keywords in a document may be useful when classifying the document, just as the *infront* relation may be useful when classifying trains.

Since the initial population of biases in ICET is set randomly, ICET may produce a different result each time it runs. Therefore, when solving a problem, ICET needs to be run several times. The best (lowest cost) decision tree that was generated for the first competition [16] is shown below. The total cost of the tree equals 18 units (obtained as a sum of `short_closed = 7`, `train_4 = 3`, `u_shaped = 5`, and `train_circle = 3`).

```
short_closed = 1: 1 (8.0)
short_closed = 0:
|   train_4 = 0: 0 (7.0)
|   train_4 = 1:
|   |   u_shaped = 0: 0 (2.0)
|   |   u_shaped = 1:
|   |   |   train_circle = 0: 0 (1.0)
|   |   |   train_circle = 1: 1 (2.0)
```

The induced decision trees were converted into Prolog programs by hand. For example, the above decision tree was converted to the following Prolog program.

```
eastbound(T) :-
   has_car(T, C),
   ((short(C), closed(C));
   (len1(T, 4), u_shaped(C), has_load1(T, circle))).
```

The above Prolog program was the entry for the first competition. This program has a complexity of 19 units, which shows that the cost of the decision tree (18 units) is only an approximation of the cost of the corresponding Prolog program, since some Prolog code needs to be added to assemble the Prolog fragments into a working whole. This extra code means that the sum of the sizes of the fragments is less than the size of the whole program. It is also sometimes possible to subtract some code from the whole, because there may be some overlap in the code in the fragments. The ideal solution to this problem would be to add a post-processing module to RL-ICET that automatically converts the decision trees into Prolog programs. The complexity could then be calculated directly from the output Prolog program, instead of the decision tree. Although post-processing with RL-ICET was done manually, it could be automated, as demonstrated by LINUS, which has a general-purpose post-processor.

4.4 Feature elimination by REDUCE

The objective of the experiments was to show the utility of the literal elimination algorithm REDUCE. Two experiments were performed separately for the 20 and 24 trains problems. In both experiments, the RL-ICET preprocessor was used to generate the appropriate features and to transform the training examples into a feature vector format. This resulted in two training sets of 20 and 24 examples each, described by 1199 features.

In order to apply the REDUCE algorithm we first converted the starting feature vector of 1199 elements to the corresponding literal vector which has twice as many elements, containing 1199 features generated by the RL-ICET preprocessor (positive literals) as well as their negated counterparts (1199 negative literals). After that, we eliminated the irrelevant literals and, in the third phase, constructed the reduced set of features which includes all the features which have at least one of their literals in the reduced literal set.

The experimental setup, designed to test the utility of REDUCE, was as follows. First, 10 runs of the ICET algorithm were performed on the set of training examples described with 1199 features. Second, 10 runs of ICET were performed on the training examples described with the reduced set of features selected by REDUCE.

Ten runs were needed because of the stochastic nature of the ICET algorithm: each time it runs, it yields a different result (assuming that the random number seed is changed). If we compared one single run of ICET on 1199 features to one run of ICET on the reduced feature set, the outcome of the comparison could be due to chance.

The results were compared with respect to execution times, costs of decision trees induced by ICET, and the complexity of Prolog programs after the RL-ICET transformation of decision trees into the Prolog program form (notice that the transformation into the Prolog form is currently manual and sub-optimal, which means that a tree with lowest cost found by ICET is not necessarily transformed into a Prolog program with lowest complexity).

The results of the experiment are summarized in Tables 3 and 4. The average results of 10 runs of RL-ICET were compared with respect to the costs of decision trees and execution times. Notice that all the experiments are independent of each other, e.g., results of experiment 4 should not be compared to the results of experiment 14. Only average results are relevant for the comparison.

4.4.1 Results of the 20 trains experiment

With the 20 train data, REDUCE cut the original set of 1199 features down to 86 features. In this way, the complexity of the learning problem was reduced to about 7% (86/1199) of the initial learning problem. Results of 10 runs of ICET on the 1199 feature set are the results reported in [16], whereas results of 10 runs of ICET on the training examples described with 86 features are new.

The results show that the efficiency of learning significantly increased. In the initial problem with 1199 features, the average time per experiment was about 2 hours and 17 minutes, whereas in the reduced problem setting with 86 features the average time per experiment was about 12 minutes. The difference between times t_1 and t_2 is significant at the 99.99% confidence level. This shows the utility of literal reduction for genetic algorithms which are typically expensive in terms of CPU time.

The average cost of descriptions induced from the 86 feature set has decreased (from 20 to 18.6), but the difference between decision tree costs c_1 and c_2 is not significant. The variance (or the standard deviation) of the costs was also reduced, i.e., the costs of the decision trees

$86\ features$				$1199\ features$			
$Trial$	$Time$ t_1	$Cost$ c_1	$Compl.$ cm_1	$Trial$	$Time$ t_2	$Cost$ c_2	$Compl.$ cm_2
1	11 : 05	18	22	11	2 : 21 : 32	24	25
2	11 : 19	21	27	12	2 : 21 : 34	21	22
3	12 : 55	18	22	13	2 : 19 : 15	20	22
4	11 : 35	18	22	14	2 : 19 : 32	20	22
5	15 : 16	18	22	15	2 : 16 : 20	18	19
6	11 : 35	18	19	16	2 : 23 : 52	22	23
7	11 : 32	18	22	17	2 : 24 : 09	21	22
8	11 : 38	18	22	18	2 : 18 : 41	16	20
9	11 : 28	18	22	19	2 : 16 : 58	18	22
10	11 : 18	21	23	20	2 : 23 : 09	20	22
Sum	119 : 41	186	223	Sum	23 : 25 : 02	200	219
$Mean$	11 : 57	18.6	22.3	$Mean$	2 : 16 : 54	20	21.9

Table 3: Summary of results in the 20 trains East-West challenge.

generated from 1199 features vary more than the costs of the trees generated from 86 features: $var(c_1) = 1.6$ ($sd(c_1) = 1.3$) and $var(c_2) = 5.1$ ($sd(c_2) = 2.3$).

The lowest cost decision tree (16 units) reported in Table 3 was generated in trial 18, with the full set of 1199 features. However, all of the features that appear in the tree in trial 18 are also in the reduced set of 86 features, so REDUCE does not prevent RL-ICET from possibly discovering this tree (if the genetic search algorithm is lucky). Notice that the sub-optimal transformation into the Prolog form transforms the lowest cost decision tree into a Prolog program with complexity 20, which is higher than the minimal size-complexity non-recursive Prolog program (size 19) for the 20 train problem, generated from decision trees generated in trials 6 and 15.

It is important to note that, using the reduced set of 86 features, in Trial 6, the same best tree as reported in [16] and obtained by Trial 15 from 1199 features, was induced (cost = 18, complexity = 19, see Section 4.3). The fact that the same optimal non-recursive Prolog program was induced and the substantial efficiency increase confirm the usefulness of our approach for learning with genetic algorithms.

It is also interesting to observe that the trees induced by RL-ICET from the set of 1199 features use mostly the features of the reduced 86 feature set; however, some other features are used as well (in Trial 11: `triangle_load_one_load`, in Trials 13, 14 and 20: `no_roof_infront_short`, in Trial 17: `triangle_load`).

4.4.2 Results of the 24 trains experiment

In this experiment, REDUCE decreased the number of features from 1199 to 116. In this way, the complexity of the learning problem was reduced to about 10% (116/1199) of the initial learning problem. The results show that the efficiency of learning significantly increased. In the initial problem with 1199 features, the average time per experiment was nearly two hours, whereas in the reduced problem setting with 116 features the average time per experiment was about 14 minutes. The difference between times t_1 and t_2 is significant at the 99.99% confidence level.

116 features				*1199 features*			
Trial	*Time* t_1	*Cost* c_1	*Compl.* cm_1	*Trial*	*Time* t_2	*Cost* c_2	*Compl.* cm_2
1	14 : 35	20	33	11	1 : 54 : 15	27	33
2	14 : 26	18	30	12	1 : 55 : 29	21	28
3	14 : 59	18	30	13	2 : 00 : 25	26	31
4	14 : 17	21	34	14	1 : 56 : 31	25	28
5	13 : 32	18	28	15	1 : 56 : 47	25	30
6	13 : 31	22	27	16	1 : 57 : 14	24	27
7	14 : 29	18	28	17	1 : 56 : 52	28	31
8	13 : 54	23	28	18	1 : 56 : 33	23	28
9	13 : 51	23	33	19	1 : 49 : 08	27	30
10	14 : 30	18	29	20	1 : 47 : 46	28	41
Sum	2 : 22 : 04	199	300	*Sum*	19 : 11 : 00	254	307
Mean	14 : 12	19.9	30	*Mean*	1 : 55 : 05	25.4	30.7

Table 4: Summary of results in 24 trains East-West challenge.

The average cost of decision trees induced from the 116 feature set has also decreased. The difference between decision tree costs c_1 and c_2 is significant at the 99.99% confidence level. Our hypothesis that variance (standard deviation) of the output of RL-ICET can be reduced is only weakly supported since the inequality of variance is insignificant: $var(c_1) = 4.8$ ($sd(c_1) = 2.2$) and $var(c_2) = 5.2$ ($sd(c_2) = 2.3$).

Turney's original entry in Michie's 24 train challenge was the following tree whose total cost equals 23 units:

```
peaked_roof = 1: 1 (6.0)
peaked_roof = 0:
|   double = 1:
|   |   train_diamond = 0: 1 (5.0)
|   |   train_diamond = 1: 0 (1.0)
|   double = 0:
|   |   closed_hexagon_load = 1: 1 (1.0)
|   |   closed_hexagon_load = 0: 0 (11.0)
```

The tree is transcribed into a Prolog program of size 23:

```
eastbound(T) :-
   has_car(T, C),
   (arg(5, C, peaked);
   (double(C), not has_load0(C, diamond));
   (has_load0(C, hexagon), closed(C))).
```

Notice that the cost of this decision tree is only slightly below average for 1199 features, is above average for 116 features, and is higher than the cost of the minimal decision tree induced from the 116 feature set (minimal cost is 18). However, this Prolog program has a lower complexity than any of the 20 Prolog programs generated in this experiment. This tree/Prolog program was generated with all of the `_infront_` features disabled.[6] When the `_infront_` features are removed, the number of features drops from 1199 to 415.

When comparing the features of the above best tree with the set of 116 features, all features appearing in the tree appear also in the reduced 116 feature set, except `closed_hexagon_load`. However, the feature `closed_infront_closed` substitutes for `closed_hexagon_load` in the 116 feature set. This means that the reduction algorithm could have removed from the feature set either `closed_hexagon_load` or `closed_infront_closed`, but has, due to its ignorance about the transformation procedure into Prolog encodings, randomly decided to eliminate `closed_hexagon_load`. The resulting substition does not change the cost of the tree (cost = 23), but the corresponding Prolog program is larger (complexity = 26):

```
peaked_roof = 1: 1 (6.0)
peaked_roof = 0:
|   double = 1:
|   |   train_diamond = 0: 1 (5.0)
|   |   train_diamond = 1: 0 (1.0)
|   double = 0:
|   |   closed_infront_closed = 1: 1 (1.0)
|   |   closed_infront_closed = 0: 0 (11.0)

eastbound(T) :-
   has_car(T, C),
   (arg(5, C, peaked);
   (double(C), not has_load0(C, diamond));
   (infront(T, C1, C2), closed(C1), closed(C2))).
```

4.5 Costs versus complexity: Results for 20 and 24 trains

Due to the imperfect transformation of decision trees into Prolog rules, it is hard to achieve the optimal result in terms of the complexity of induced Prolog rules (the goal of the competition

[6]Disabling this feature was due to the suspicion that the "infront" features weren't very useful, since S. Muggleton had decided to redefine the "append" predicate by adding cut to it.

was to minimize Prolog code complexity); that is, RL-ICET optimizes the cost of decision trees and not the complexity of Prolog encodings.

To test the correlation between the costs of decision trees and the complexity of Prolog rules we have tried to find the correlation between costs and complexity.

To this end, we have computed the correlation between cost and complexity for the 20 trains experiment: $corr(c_1, cm_1) = 0.73$ and $corr(c_2, cm_2) = 0.86$, and for the 24 trains experiment: $corr(c_1, cm_1) = 0.22$ and $corr(c_2, cm_2) = 0.66$. From these correlations we speculate that:

- the correlation decreases as the number of trains increases, and
- the correlation decreases as the number of features decreases.

Similar conclusions hold also if the correlations between cost and complexity are computed on vectors of 20 elements (and not 10 elements as above), merging the results of the 20 and the 24 trains experiments.

The complexity of the Prolog programs is only weakly correlated with the cost of the decision trees. The ideal correlation (1.0) was not achieved due to the imperfect transformation of decision trees into Prolog rules. This transformation is performed in two steps. In the first step, a Prolog program is created whose structure is nearly identical to the structure of the decision tree. The second step employs ways to compress the Prolog program, by removing redundant sections of code and altering the structure of the program; this step actually disturbs the correlation between the cost of the decision trees and the complexity of the Prolog programs. If the second step were eliminated, there would be a much higher correlation between the cost and the complexity, but there would also be a large increase in the complexity of Prolog programs.

The ideal solution to this problem would be to fully automate the transformation of the decision trees to Prolog programs and then modify RL-ICET to search for the least complex Prolog program, instead of the least costly decision tree.

4.6 Applying C4.5 in the East-West Challenge

We have compared the results of RL-ICET with the ones achieved using C4.5 [13]. This experiment was made in order to check whether our claims of the usefulness of feature reduction can be made more general.

To do so, C4.5 was first applied to the 24 trains problem, using the default settings. C4.5 generated a tree that makes one error on the training data; it misclassifies one of the 24 trains. This is because of one of the default C4.5 parameters settings: the parameter that sets the minimum number of objects that can be at one leaf in the tree. The default value is two. In the experiments on 20 trains and 24 trains, C4.5 was therefore run with the parameter setting which sets the minimum number of objects at one. In this way, C4.5 generates trees that make no errors on the training data.

4.6.1 Results of C4.5

Results of using C4.5 are given in Table 5. Feature reduction does not help C4.5 to find a better solution in terms of costs: using feature reduction, C4.5 becomes slower and gains nothing, since feature reduction itself takes about 5 minutes (real 4:49.00, user 3:32.50, system 0:40.34, times measured on a HP Workstation).

	20 *Trains* 86 *features*	20 *trains* 1199 *features*	24 *trains* 116 *features*	24 *trains* 1199 *features*
Cost	22	22	23	23
Time (seconds)	1	1	1	2

Table 5: Results of C4.5.

4.6.2 Comparing RL-ICET and C4.5

From the comparison of Table 4 and Table 5 we see that for the 116 feature dataset RL-ICET has lower average cost (19.9) than C4.5 (cost 23), but not for the 1199 feature data (average cost 25.4 for RL-ICET and cost 23 for C4.5). From the comparison of Tables 3 and 5 we see that RL-ICET has lower average cost than C4.5 both for the 86 feature dataset (average cost of 18.6 units for RL-ICET and 22 units for C4.5) and for the 1199 feature dataset (average cost of 20 units for RL-ICET and 22 units for C4.5).

In order to see how much literal reduction contributes to the favourable results achieved by RL-ICET, let us separately analyse the results for the reduced and original datasets. For reduced feature sets, the results are favourable for RL-ICET when compared to C4.5.

For the reduced datasets the results are as follows:

86 features, 20 trains: both the minimal cost (18) and the average cost (18.6) are lower than the cost of the tree induced by C4.5 (22).

116 features, 24 trains: both the minimal cost (18) and the average cost (20) are lower than the cost of the tree induced by C4.5 (23).

For the original 1199 feature datasets the results are as follows:

1199 features, 20 trains: both the minimal cost (16) and the average cost (20) of the trees induced by RL-ICET are better than the cost of the tree induced by C4.5 (22).

1199 features, 24 trains: the minimal cost (21) of the tree induced by RL-ICET is lower than the cost of the tree induced by C4.5 (23). However, the average cost of trees induced by RL-ICET (25.4) is higher.

In summary, we may claim that feature reduction helped RL-ICET to achieve very favourable results. In both experiments (20 and 24 trains) it helped RL-ICET to outperform C4.5, when comparing costs of decision trees, both in terms of minimal and average costs. In both experiments it helped RL-ICET to substantially reduce the execution time; however, even on reduced feature sets, RL-ICET of course needs much more time than C4.5.

5 Summary

This work is a study of the problem of relevance for inductive learning system, applicable both in attribute-value (feature vector) learning and in the LINUS transformation approach to inductive logic programming. Irrelevant literal elimination, such as performed by REDUCE, assumes that the goal of the learning algorithm is to find a simple (low size or low cost) hypothesis. This assumption applies to most existing learning systems.

The presented case study of data preprocessing shows that cost-sensitive elimination of irrelevant features can substantially improve the efficiency of learning and can reduce the costs of induced hypotheses. This study, using the hybrid genetic decision tree induction algorithm RL-ICET on two East-West Challenge problems, together with other presented results in feature reduction confirm the usefulness of feature reduction in preprocessing.

In order to evaluate the effects of feature reduction, we have also compared the results of ICET (with and without feature reduction) with the results achieved using C4.5 [13]. In both experiments, feature reduction (reduction to 86 and 116 features, respectively) helped ICET to outperform C4.5 when comparing costs of decision trees, both in terms of minimal and average costs. On the other hand, the application of REDUCE did not help C4.5 itself to induce a lower cost solution from examples described with fewer features.

More detailed information on this study is available in two technical reports of J. Stefan Institute, Ljubljana, that can be obtained upon request from the authors.

Acknowledgements

This research was supported by the Slovenian Ministry of Science and Technology, the Croatian Ministry of Science, the National Reseach Council of Canada and the ESPRIT IV project 20237 Inductive Logic Programming 2. The authors are grateful to Donald Michie for his stimulative interest in this work, and to Sašo Džeroski for his involvement in earlier experiments.

References

1. Caruana, R. and D. Freitag: Greedy Attribute Selection, in: Proceedings of the 11th International Conference on Machine Learning, Morgan Kaufmann, 1994, 28–36.

2. Fayyad, U.M. and K.B. Irani: On the handling of continuous-valued attributes in decision tree generation, Machine Learning, 8 (1992), 87–102.

3. Gamberger, D.: A Minimization Approach to Propositional Inductive Learning, in: Proceedings of the 8th European Conference on Machine Learning, Springer, 1995, 151–160.

4. Grefenstette, J.J.: Optimization of control parameters for genetic algorithms, IEEE Transactions on Systems, Man, and Cybernetics, 16 (1986), 122–128.

5. John, G.H., R. Kohavi and K. Pfleger: Irrelevant Features and the Subset Selection Problem, in: Proceedings of the 11th International Conference on Machine Learning, Morgan Kaufmann, 1994, 190–198.

6. Lavrač, N., S. Džeroski and M. Grobelnik:. Learning Nonrecursive Definitions of Relations with LINUS, in: Proceedings of the 5th European Working Session on Learning, Springer, 1991, 265–281.

7. Lavrač, N. and S. Džeroski: Inductive Logic Programming: Techniques and Applications, Ellis Horwood, 1994.

8. Lavrač, N., D. Gamberger and S. Džeroski: An Approach to Dimensionality Reduction in Learning from Deductive Databases, in: Proceedings of the 5th International Workshop on Inductive Logic Programming, Scientific Report, Katholieke Universiteit Leuven, 1995, 337–354.

9. Lavrač, N., D. Gamberger and P. Turney: Cost-Sensitive Feature Reduction Applied to a Hybrid Genetic Algorithm, in: Proceedings of the 7th International Workshop on Algorithmic Learning Theory, Springer, 1996, 127–134.

10. Michalski, R.S. and J.B. Larson: Inductive Inference of VL Decision Rules, ACM SIGART Newsletter, 63 (1977), 38–44.

11. Michalski, R.S.: A Theory and Methodology of Inductive Learning, in: Machine Learning: An Artificial Intelligence Approach (Eds. R. Michalski, J. Carbonell and T. Mitchell), Tioga, 1983, 83–134.

12. Michie, D., S. Muggleton, D. Page and A. Srinivasan: To the International Computing Community: A new East-West Challenge. Oxford University Computing Laboratory, Oxford, 1994. [Available at URL ftp://ftp.comlab.ox.ac.uk/pub/Packages/ILP/trains.tar.Z.]

13. Quinlan, J.R.: C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.

14. Skalak, D.: Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms, in: Proceedings of the 11th International Conference on Machine Learning, Morgan Kaufmann, 1994, 293–301.

15. Turney, P.: Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm, Journal of Artificial Intelligence Research, 2 (1995), 369–409. [Available at URL http://www.cs.washington.edu/research/ jair/home.html.]

16. Turney, P.: Low Size-Complexity Inductive Logic Programming: The East-West Challenge as a Problem in Cost-Sensitive Classification, in: Advances in Inductive Logic Programming (Ed. L. De Raedt), IOS Press, 1996, 308–321.