Pruning regression trees with MDL

Marko Robnik-Šikonja and **Igor Kononenko** [1]

Abstract. Pruning is a method for reducing the error and complexity of induced trees. There are several approaches to pruning decision trees, while regression trees have attracted less attention.

We propose a method for pruning regression trees based on the sound foundations of the MDL principle. We develop coding schemes for various constructs and models in the leaves and empirically test the new method against two well know pruning algorithms.

The results are favourable to the new method.

1 INTRODUCTION

Pruning is a generalization technique applied in machine learning to generalize over-specialized models obtained due to noise or the lack of statistical support. This work focuses on tree-based models in regression. Here we build an oversized tree in the learning phase and then prune the unreliable branches. There are three well known regression-tree learning systems, each with its own approach to pruning.

CART [1] uses parameter α to control the balance between the complexity and the accuracy of the tree. The algorithm either sets the value of the optimal parameter by cross validation on the training set or a separate large enough tuning set is used. This method is called error-complexity pruning.

Retis [3] uses a pruning algorithm based on decision tree pruning [10]. The estimated error of each interior node of the tree is compared with the estimated error of the subtrees of the node. If the latter is greater than the former the subtrees are pruned. Retis uses the Bayesian approach (*m*-estimate) to estimate the errors of the tree nodes on unseen examples [4]. The *m* parameter has intuitive meaning but can also be set by a cross-validation.

M5 [11] also compares errors of pruned and unpruned nodes but estimates this error with the mean absolute error on training set multiplied by a factor which takes into account the statistical support and the complexity of the model.

Although these methods have given satisfactory results in various problems they all lack proper theoretical background which would justify their assumptions. However, if we look at decision tree pruning methods there are a number of algorithms [12, 16, 8, 6] based on sound foundations of the Minimum Description Length (MDL) principle [7]. The aim of this work is to present a general pruning algorithm for regression tree models based on the MDL principle.

The paper is organised as follows. Section 2 describes our MDL based pruning algorithm and some encodings needed to encode various types of models and constructs in the tree. Section 3 contains the description of the experimental scenario and two sets of experiments and the last Section concludes and gives guidelines for further work.

[1] University of Ljubljana, Faculty of Computer and Information Science, Tržaška 25, 1001 Ljubljana, Slovenia, *tel./fax: +386-61-1768386, e-mail: {Marko.Robnik,Igor.Kononenko}@fri.uni-lj.si*

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

2 REGRESSION TREES AND MDL

The MDL principle is based on the Occam's statement that it is vain to do with more what can be done with less. It is theoretically justified with the Kolmogorov complexity [7]. The principle has been frequently used in machine learning e.g., [12, 16, 8, 5] and its use can be summarized as: code the possible solutions to the problem and select the instance with the shortest code as the result. We have to emphasize that we are only interested in the length of the code and not the encodings themselves.

The basic problem we have to solve in regression is the coding of the real numbers. For it we follow the approach of Rissanen [13]. Namely, we define a finite precision ε to which the number is interesting and change the real number x to integer:

$$Z(x) = \left\lfloor \frac{x}{\varepsilon} \right\rceil , \tag{1}$$

and then compute the code length of the integer with the following rule:

$$\begin{aligned} L(0) &= 1 \\ L(n) &= 1 + \log_2(n) + \log_2(\log_2(n)) + ... \\ &\quad ... + \log_2(2.865064) \end{aligned} \tag{2}$$

where the sum contains only positive terms. Since one does not need the codes themselves, Rissanen does not gives actual coding algorithm but only proves that the length of the function has the desired properties. The simplified intuition behind the formula (for details see [13, 2]) is that in order to code the integer n with a self contained prefix code we need, besides the obvious $log_2(n)$ bits, also the preamble of the code which contains the information how long the code is (this requires $\log_2(\log_2(n))$ bits), and since this preamble also needs to be coded we need another $\log_2(\log_2(\log_2(n)))$ bits for coding its length and so on. Since this code was developed on the basis of the universal prior probabilities for integers it also requires the term $\log_2(2.865064)$ which guarantees that the sum of probabilities for all positive integers equals 1. Additional 1 bit represents boundaries between the encodings.

2.1 Pruning algorithm

Our algorithm for pruning regression trees is based on the ideas of [10, 6]. Namely, in each interior node we compare the code of the model in that node with the codes of the subtrees and their respective models. If the latter is larger than the former the subtrees are pruned and the node is turned into a leaf. We recursively repeat this procedure from leaves to the root of the tree.

2.2 Coding the (sub)tree

To code a (sub)tree we proceed as follows: we write the nodes of the tree in a certain predefined order (e.g., preorder: depth and left first). Each node has a 1 bit code: 0 if the node is the leaf and 1 if it is the interior node. For interior node we code the splitting criterion and for the leaf we code the prediction model. The tree in Figure 1 would get the following sequence: 1 [root] 0 [left leaf] 1 [right split] 0 [middle leaf] 0 [right leaf].

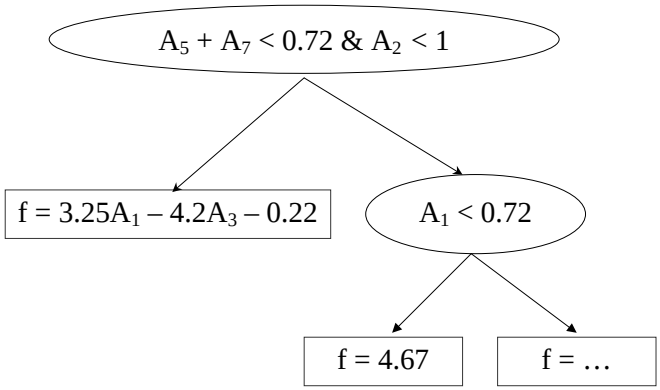

Figure 1. An illustration of the regression tree. Interior nodes contain decision and leaves contain prediction models.

We also have to code the splitting criteria in interior nodes and models in the leaves.

2.3 Coding the splitting expression

As we can see from Figure 1 the splitting criteria can be various especially if we allow lots of constructive induction operators.

We have chosen to separately code the type of the splitting expression (e.g., split on a single attribute, conjunction, sum, product) and then code each type with a simpler (shorter) scheme, rather than code some general formalism (e.g., expression tree, RPN notation).

$$code(split) = code(Type(split)) + code(Expression) \quad (3)$$

To code the type of the split we only need the logarithm of the number of different types:

$$code(Type(split)) = \log_2(\#types) \quad (4)$$

If for example we allow only single attributes and conjunctions we need $\log_2(2) = 1$ bit to encode the type of the node.

Examples of code lengths for specific expressions are given below.

Single attribute is all we need to encode if we use "ordinary" regression tree (univariate). We code discrete and continuous attributes separately.

Discrete attribute is coded as a selection of the attribute A_i from the set of all attributes A and a selection of a subset v_{A_i} of possible values V_{A_i} which defines the split to the left branch:

$$code(A_i) = \log_2(|A|) + |V_{A_i}| \quad (5)$$

For example, let we compute the length of the code for the expression $Color \in \{green, red\}$. First we code the selection of the attribute *Color* from the set of e.g., 16 attributes with $log_2(16) = 4$ bits and then selection of the subset $\{green, red\}$ from the set of altogether 8 possible colors with 8 bits, altogether 12 bits.

Continuous attribute is coded as a selection of attribute A_i from the set of all attributes A and a splitting point. Since all splitting points are equally probable a priori we use the logarithm of the number of possible splitting points. The number of possible splitting points is defined as a quotient between the length of the interval of the attribute's values (known in advance) and the selected precision. If we write the length of the interval of the attribute as $l(A_i) = max(A_i) - min(A_i)$ we get

$$code(A_i) = \log_2(|A|) + \log_2 \frac{l(A_i)}{\varepsilon} \quad (6)$$

To code the expression $Gain \leq 15.2$ we need $log_2(16) = 4$ bits to encode the selection of the attribute *Gain* from the set of 16 attributes, $\log_2 \frac{l(Gain)}{\varepsilon} = \log_2 \frac{100}{0.1} = 9.97$ bits to encode the value interval of the attribute *Gain* spanning from 0 to 100 and the numbers being precise to 0.1. This makes together 13.97 bits.

Conjunction consists of a sequence of terms where each term is either a value of the discrete attribute or an interval of values of the continuous attribute. We do not need to code the subset of values going left in the tree since logical expressions have only two possible outcomes and we can presume that e.g., true goes left.

If N is the maximal number of terms and T the actual number of terms used we can write

$$code(Conj) = \log_2(N) + \sum_{i=1}^{T} code(Term_i) \quad (7)$$

To code the expression $(Color = green) \wedge (20.1 \leq Gain < 40.0)$ in the setting where we allow 3 terms at most (and we actually have two terms) we need $log_2(3) = 1.58$ bits to encode the number of terms plus two additional pieces of code, one for each of the terms. For term which presents *a value of the discrete attribute* we code the selection of the attribute and selection of the value:

$$code(Term_d) = \log_2(|A|) + \log_2(|V_{A_i}|) \quad (8)$$

The difference to the Equation (5) is due to the fact that we select only a single value of the attribute here and not a subset of the values. So, to code the term $Color = green$ we need $log_2(16) = 4$ bits for the selection of the attribute *Color* and $\log_2(8) = 3$ bits to code the color *green* from the set of 8 possible colors, making it all together 7 bits.

For term which presents *an interval of the values of the continuous attribute* we code the selection of the attribute and two boundaries (lower and upper) of the interval of the values:

$$code(Term_c) = \log_2(|A|) + 2 \cdot \log_2 \frac{l(A_i)}{\varepsilon} \quad (9)$$

The term $20.1 \leq Gain < 40.0$ would be encoded with $\log_2(16) = 4$ bits for selection of the attribute *Gain* and $2 \cdot \log_2 \frac{100}{0.1} = 19.94$ bits for the two boundaries (20.1 and 40.0), which sums to 23.94 bits.

Sum and product: expression consist of a sequence of continuous attributes $\sum_{i \in S} A_i \leq B$ or $\prod_{i \in S} A_i \leq B$. We first code the actual length of the expression $|S|$ and then selection of the attributes to be included in the sum (product). The boundary value has to be

coded with Equation (2) and one bit is added for the sign. If C is a set of all continuous attributes, S a set of selected attributes, and N maximal allowed length of the expression we get

$$code(Sum) = \log_2(N) + \log_2\left(\frac{|C|}{|S|}\right) + 1 + L\left(\frac{|B|}{\varepsilon}\right) \qquad (10)$$

The sum $Gain + Gift \leq 9.3$ would require $\log_2(3) = 1.58$ bits for determining that it consists from 2 out of 3 maximally allowed terms, $\log_2\binom{8}{2} = \log_2(28) = 4.81$ bits for selection of the two attributes $Gain$ and $Gift$ from 8 existing continuous attributes and $1 + L(\frac{9.3}{0.1}) = 14.73$ bits for the boundary (9.3) which makes together 21.12 bits.

2.4 Coding the leaves

If the receiver wants to reconstruct the prediction (to a predefined precision) we have to send also the models in the leaves describing the data and the errors made by the models on the data. If N_t is the number of examples falling into the leaf, and e_i is the error on i^{th} example we can write

$$code(leaf) = code(model) + \sum_{i=1}^{N_t} code(e_i), \qquad (11)$$

The error is coded by Equation (2) which penalizes larger errors.

$$code(e_i) = 1 + L\left(\frac{|c_i - \tau(x_i)|}{\varepsilon_e}\right) \qquad (12)$$

where c_i is the true predicted value of example i, $\tau(x_i)$ is the value of example i predicted by the model and ε_e is a predefined precision to which the error is measured. 1 bit is added for the sign of the error.

Various models can be used in the leaves (linear formulas, nearest neighbour, kernel densities). We give an example of linear formulas which are used in both M5 and Retis.

Linear formula is of the form $f = \sum_{i \in S} a_i A_i$, where S is the subset of the set which consists of all the continuous attributes C and a constant 1. We have to encode the power of the set S, the selection of S from its superset, and also $|S|$ coefficients a_i. We write

$$\begin{aligned} code(Lin) \quad = \quad & log_2(|C|+1) + \log_2\left(\begin{matrix}|C|+1\\|S|\end{matrix}\right) + \\ & \sum_{i \in S}(1 + L\left(\frac{|a_i|}{\varepsilon}\right)) \end{aligned} \qquad (13)$$

To encode the formula $f = -1.3Gain + 4.2Gift + 0.7$ we code the fact that we have 3 terms out of possible 9 (8 continuous attributes and a constant) with $log_2(9) = 3.17$ bits, selection of the terms ($Gain$, $Gift$, and 1) with $\log_2\binom{9}{3} = 6.39$ bits and finally the coefficients (-1.3,4.2,0.7) with $1 + L(\frac{1.3}{0.1}) + 1 + L(\frac{4.2}{0.1}) + 1 + L(\frac{0.7}{0.1}) = 8.50 + 11.46 + 6.87 = 26.83$ bits, together with 36.39 bits.

2.5 Setting the parameters

The only parameter we have to set in our pruning algorithm is the precision of real numbers. Actually, we have decided to use two separate precision parameters: one for the arithmetic purposes (coefficients in linear models and splitting points) and another one for error of the prediction. They both have an intuitive meaning and can be set with the help of the user for each domain we want to learn. The user knows to what precision the values of the attributes are given and to what precision the results should be predicted, so he/she can set the precision parameters to sensible values.

With the help of these two parameters we can also control the extent of pruning which we demonstrate in the next section.

3 EXPERIMENTS

We have built the MDL pruning method into the CORE system [14] for building regression trees. CORE uses RReliefF [15] and mean squared error (MSE) as the estimators of attribute quality, enables constructive induction (conjunction, sum and product) in the interior nodes and multiple models in the leaves (point, linear, kNN).

To test the performance of our pruning method we have compared it against the m-pruning method of Retis and CART's error-complexity pruning.

Retis [3] uses similar bottom-up recursive algorithm as our approach. In each node it estimates and compares errors the (sub)tree would make, if it were pruned at that point and if it were not. If the estimated error of the pruned node is lower than the estimated error of the unpruned node then the subtrees of that node are pruned. The crucial point in this form of pruning is the estimation of errors. Retis uses a Bayesian approach to error estimation, where the probability of the outcome X is estimated as:

$$p(X) = \frac{n}{n+m}p_X + \frac{m}{n+m}p_a \qquad (14)$$

where p_X is the relative frequency of the outcome X in n trials, and p_a is the prior probability of X. Parameter m weights p_a. Larger m implies greater influence of prior probabilities, so the tree must be smaller. We have used general version of this approach which can be used with arbitrary models in the leaves (Retis supports only linear models). As we estimate errors, we can write the Bayesian error of the model τ as:

$$e(\tau) = \frac{n}{n+m}e_\tau + \frac{m}{n+m}e_a(\tau) \qquad (15)$$

where n is the number of examples in the node. Prior error e_a of the model τ is computed on all training examples (not only the examples in the node).

CART's error-complexity pruning [1] uses parameter α to balance error and complexity of the tree. With increasing α we get decreasing (in size) sequence of trees and then select the one which minimizes the estimated error on yet unseen examples. This error estimation demands that we build several trees (with cross-validation) or we need a separate tuning set of training data.

M5's pruning [11] works only for the linear models in the leaves and does not have any parameters to control the size of the tree so it would not be fair to compare it against our pruning with tuned parameters. MDL pruning with the precision parameters set to defaults gave comparable results to M5's pruning (there were no significant differences neither in the error of the prediction nor in the complexity of the obtained trees).

3.1 Experimental scenario

We ran our system on the artificial data sets and on data sets with continuous prediction value from UCI [9]. Artificial data sets used were:

Fraction is floating point generalization of the parity concepts of order $I = \{2,3,4\}$. Each domain contains continuous attributes

with values from 0 to 1. The predicted value is the fractional part of the sum of I important attributes: $C = \sum_{j=1}^{I} A_j - \lfloor \sum_{j=1}^{I} A_j \rfloor$

Modulo-8 is integer generalization of the parity of order $I = \{2,3,4\}$. Value of each attribute is an integer value in the range 0-7. Half of the attributes are treated as discrete and half as continuous; each continuous attribute is exact match of one of the discrete attributes. The predicted value is the sum of the I important attributes by modulo 8: $C = (\sum_{j=1}^{I} A_j) \bmod 8$.

Parity is continuously randomized parity of order $I = \{2,3,4\}$. Each data set consists of discrete, Boolean attributes. The I informative attributes define parity concept: if their parity bit is 0, the predicted value is set to a random number between 0 and 0.5, otherwise the predicted value is randomly chosen to be between 0.5 and 1.

Linear presents simple linear dependency with 4 important attributes: $f = A_1 - 2A_2 + 3A_3 - 3A_4$. The attributes are continuous with values chosen randomly between 0 and 1.

Cosinus is non-linear dependency with cosinus multiplied by the linear combination of two attributes: $f = \cos(4\pi A_1) \cdot (-2A_2 + 3A_3)$. The attributes are continuous with values from 0 to 1.

Altogether there were 11 artificial data sets, each consisting of 10 attributes - 2, 3 or 4 important, the rest are random, and containing 1000 examples. UCI datasets used were:

Abalone: predicting the age of the abalone, 1 nominal and 7 continuous attributes, 4177 instances.

Autompg: city-cycle fuel consumption, 1 nominal, 6 continuous attributes 398 instances.

Autoprice: prices of the vehicles, 10 nominal , 15 continuous attributes, 201 instances.

Cpu: relative CPU performance, 6 continuous attributes, 209 instances.

Housing: housing values in Boston area, 1 nominal, 12 continuous attributes, 506 instances.

Servo: rise time of a servomechanism, 2 nominal, 2 integer attributes, 167 instances.

Wisconsin: time to recur in Wisconsin breast cancer database, 32 continuous attributes, 198 instances.

For each experiment we collected the results as the average of 10 fold cross-validation and the significance of the differences were computed by a two-sided Student's t-test at a 0.05 level.

3.2 Upper bounds of performance

With the first set of experiments we tried to measure the upper bound of performance of pruning, namely how good can we actually perform with the pruning algorithm. For it we have built full trees in several different settings and for each setting performed the pruning with several different values of the pruning parameters. The best pruning result for each method was chosen to represent what can actually be done in the best case with that pruning method. For MDL pruning we were varying the precision on the logarithmic scale: for arithmetic precision from 0.0002 to 100 (30 values) and for the error precision relatively to the prediction values interval from 0.0002 to 1 (20 values), altogether 600 values. For *m*-pruning we were varying the *m* parameter on the logarithmic scale from 0.0001 to 10000 (80 values). For α in error-complexity pruning one can compute the values of α so that the obtained sequence of the trees is smoothly decreasing from full sized tree to one leaf. The number of trees in the sequence and (therefore also the number of different α values) is problem dependend and was varying from a few hundred (abalone) to only a few. Although it may seem that MDL pruning is given advantage here since it was tested with more different settings of the parameters this is not the case, namely increasing sequences of *m* and α caused the size of the tree to decrease smoothly which means that with denser sampling of the parameters we would not get different pruned trees and different results.

We were changing the following options of our learning system: estimation of the attributes with RReliefF or MSE, constructive induction (conjunction, sum and product) on or off, linear models in the leaves or just point predictions (with the mean of the prediction values). These options gave together 8 different major settings of our system and the pruning algorithms were tested in each of them.

Since the results (and conclusions) were similar in each of the setting we present only results for RReliefF, constructive induction, and linear models in Table 1. We present the relative error (*RE*), complexity of the tree (*C*), and significance of the differences (sign.) in complexity. If MDL-pruning produced significantly smaller models than *m*-estimate pruning or error-complexity pruning we marked this with letters m and e in significance column, respectively. If the differences were insignificant we mark this with '-'. Error-complexity or *m*-estimate pruning never produced significantly less complex trees than MDL pruning. We did not detect any significant difference in error so we do not report significance of the error differences in the Tables 1 and 2.

Table 1. Upper bounds of performance of the pruning methods. Numbers give the relative mean squared error (*RE*), the complexity of the pruned tree (*C*), and the significance (sign.) of the differences between the MDL and the two other methods.

	m		err. comp.		MDL		
problem	*RE*	*C*	*RE*	*C*	*RE*	*C*	sign.
Fraction-2	.02	45	.02	45	.02	45	-
Fraction-3	.01	138	.01	111	.03	121	-
Fraction-4	.75	175	.74	201	.76	188	-
Modulo-8-2	.00	118	.00	118	.00	116	-
Modulo-8-3	.00	139	.00	139	.00	148	-
Modulo-8-4	1.07	50	1.17	31	1.20	25	m
Parity-2	.29	12	.29	12	.31	7	-
Parity-3	.27	15	.27	15	.27	15	-
Parity-4	.25	31	.25	31	.25	31	-
Linear	.01	21	.01	16	.02	11	m,e
Cosinus	.23	482	.21	530	.21	331	m,e
Abalone	.47	56	.48	140	.46	41	m,e
Autompg	.14	33	.17	76	.15	32	e
Autoprice	.19	102	.20	118	.21	25	m,e
CPU	.12	95	.15	107	.14	27	m,e
Housing	.23	164	.21	167	.24	161	-
Servo	.17	84	.17	68	.20	41	m,e
Wisconsin	.94	36	.95	118	.94	33	e

All methods give approximately equal results concerning the error but the MDL based pruning is superior concerning the complexity of the trees, especially in the real world problems, giving significantly less complex trees than both competing methods in 4 out of 7 problems.

3.3 Tuning parameters with cross validation

With the next set of experiments we assumed that we do not know sensible values of the parameters in advance. Therefore we did 5-fold cross validation on the training data for the tuning of the pruning

parameters. We built the model on 4 splits and tried the pruning with the above described set of parameters on the fifth one and selected the best parameters. We repeated the procedure 5 times, each time with the different split for pruning and averaged the best parameter values over 5 trials. Then we once again built the tree on the whole training data and did the pruning with only the selected parameters. We compare the performance of these trees (giving averages of 10-fold cross validation) in the Table 2. Since the conclusions are similar across different settings of the learning system we give, as before, the results for RReliefF with constructive induction and linear models in the leaves.

Table 2. Tuning the parameters for the pruning methods with 5-fold cross-validation. Numbers give the relative mean squared error (*RE*), complexity of the pruned tree (*C*) and significance of the differences between the MDL and the other two methods.

	m		err. comp.		MDL		
problem	*RE*	*C*	*RE*	*C*	*RE*	*C*	sign.
Fraction-2	.04	453	.05	482	.05	364	m,e
Fraction-3	.02	452	.02	468	.03	433	e
Fraction-4	1.05	249	.90	250	1.02	266	-
Modulo-8-2	.00	118	.00	118	.00	118	-
Modulo-8-3	.00	165	.00	166	.00	174	-
Modulo-8-4	1.07	50	1.06	51	1.03	47	-
Parity-2	.29	12	.29	12	.29	8	m,e
Parity-3	.32	40	.32	53	.34	20	m,e
Parity-4	.28	33	.28	33	.28	31	-
Linear	.14	510	.14	499	.15	333	m,e
Cosinus	.29	486	.29	481	.30	322	m,e
Abalone	.64	1202	.65	1051	.69	959	m,e
Autompg	.22	200	.23	209	.22	139	m,e
Autoprice	.23	97	.23	102	.30	49	m,e
CPU	.14	88	.12	114	.27	46	m,e
Housing	.37	252	.36	251	.41	172	m,e
Servo	.19	72	.19	66	.20	37	m,e
Wisconsin	1.13	30	1.56	104	1.10	21	m,e

The interpretation of the results is similar as above. MDL pruning mostly produces significantly smaller trees with approximately the same error. Especially impressive is this result for the UCI data sets, where MDL-pruning significantly outperformed both competitors on all 7 problems.

4 CONCLUSIONS

We have developed a theoretically sound, MDL based procedure for pruning of regression trees. The parameters of this procedure are intuitively comprehensible and easily set by the user who knows the domain, since he/she knows the precision of the values of the attributes and precision of the predicted values and these values are sensible defaults.

We have empirically compared our pruning method with the well-known *m*-pruning and error-complexity pruning and it turned out that our pruning method produces potentially better trees (significantly less complex with comparable accuracy). We have also empirically shown that automatic setting of the parameters is feasible for the MDL pruning and produces significantly less complex trees with comparable error.

In further work we are planning to develop new coding schemes for more advanced constructs in interior nodes (e.g., X-of-N) and also for the general expressions (e.g., RPN notation). In the leaves of the trees we are planning to code other predictors such as kernel density functions.

REFERENCES

[1] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone, *Classification and regression trees*, Wadsworth Inc., Belmont, California, 1984.

[2] Peter Elias, 'Universal codeword sets and representations of the integers', *IEEE Transactions on Information Theory*, **21**(2), 194–203, (1975).

[3] Aram Karalič, 'Employing linear regression in regression tree leaves', in *Proceedings of European Conference on Artificial Intelligence '92*, ed., Bernd Neumann, pp. 440–441. John Wiley & Sons, (1992).

[4] Aram Karalič and Bojan Cestnik, 'The bayesian approach to tree-structured regression', in *Proceedings of Information Technology Interfaces '91*, pp. 155–160, Cavtat, Croatia, (1991).

[5] Igor Kononenko, 'On biases in estimating multi-valued attributes', in *Proceedings of the International Joint Conference on Aartificial Intelligence '95*, pp. 1034–1040. Morgan Kaufmann, (1995).

[6] Igor Kononenko, 'The minimum description length based decision tree pruning', Technical report, University of Ljubljana, Faculty of Information and Computer Science, (1997). (submitted).

[7] Ming Li and Paul Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, New York, 1993.

[8] Manish Mehta, Jorma Rissanen, and Rakesh Agrawal, 'MDL-based decision tree pruning', in *Proceedings of Knowledge Discovery in Databases '95*, pp. 216–221, (1995).

[9] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases, 1995. (http://www.ics.uci.edu/ mlearn/MLRepository.html).

[10] Tim Niblett and Ivan Bratko, 'Learning decision rules in noisy domains', in *Developement in Expert Systems*, ed., M. Bramer, Cambridge University Press, (1986).

[11] J. Ross Quinlan, 'Combining instance-based and model-based learning', in *Proceedings of the X. International Conference on Machine Learning*, pp. 236–243. Morgan Kaufmann, (1993).

[12] J. Ross Quinlan and Ronald L. Rivest, 'Inferring decision trees using the minimum description length principle', *Information and Computation*, **80**, 227–248, (1989).

[13] Jorma Rissanen, 'A universal prior for integers and estimation by minimum description length', *The Annals of Statistics*, **11**(2), 416–431, (1983).

[14] Marko Robnik Šikonja, 'Core - a system that predicts continuous variables', in *Proceedings of Electrotechnical and Computer Science Conference*, pp. 145–148, Portorož, Slovenia, (1997).

[15] Marko Robnik Šikonja and Igor Kononenko, 'An adaptation of Relief for attribute estimation in regression', in *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, ed., Dough Fisher, pp. 296–304. Morgan Kaufmann Publishers, (1997).

[16] C.S. Wallace and J.D. Patrick, 'Coding decision trees', *Machine Learning*, **11**(1), (1993).