A classifier based on a decision tree with verifying cuts

Jan G. Bazan[1], Stanislawa Bazan-Socha[2], Sylwia Buregwa-Czuma[1], Lukasz Dydo[1], Wojciech Rzasa[1], and Andrzej Skowron[3]

[1] Interdisciplinary Centre for Computational Modelling, University of Rzeszow
Pigonia 1, 35-310 Rzeszow, Poland
[2] II Department of Internal Medicine, Jagiellonian University Medical College
Skawinska 8, 31-066 Krakow, Poland
[3] Institute of Mathematics, The University of Warsaw
Banacha 2, 02-097 Warsaw, Poland

Abstract. This article introduces a new method of a decision tree construction. Such decision tree is constructed with the usage of additional cuts that are used for a verification of cuts in tree nodes during the classification of objects. The presented approach allows the use of additional knowledge contained in the attributes which could be eliminated using greedy methods. The paper includes the results of experiments that have been performed on data obtained from biomedical database and machine learning repositories. In order to evaluate the presented method, we compared its outcomes with the results of classification using a local discretization decision tree, well known from literature. The results of comparison of the two approaches show that making decisions is more adequate through the employment of several attributes simultaneously. Our new method allowed us to achieve better quality of classification then the existing method.

Keywords: rough sets, discretization, concept approximation, classifiers.

1 Introduction

One of the most popular method for classifiers construction is based on learning rules from examples (see e.g. [3,6,7,15]). Unfortunately, the decision rules constructed in this way may often be inappropriate to classify unseen cases. For instance, if we have a decision table where the number of values is high for some attributes, then there is a very low chance that a new object is recognized by rules generated directly from this table, because the attribute value vector of a new object will not match any of these rules. Therefore for decision tables with such numeric attributes some discretization strategies are applied to obtain a higher quality classifiers. In this paper we consider the supervised discretization, i.e. the discretization methods that use in their action the values of decision attribute for training cases. There are many methods of supervised discretization, which are based on various heuristics. We use an approach based on the creation of the so-called decision tree of the local discretization (see, e.g., [4,6,13]). It is a binary tree, created by multiple binary partitions of the set into two groups of objects (e.g., cases, states, processes, patients, observations, vehicles) with the value of the selected attribute. The decision tree of local discretization can be used not only for discretization but also can be treated as a classifier (see Section 1.1).

However, there are serious doubts as to the validity of this approach of classifier construction. Therefore, we propose a new method of decision tree construction with the usage of additional cuts making it possible to verify cuts in tree nodes during classifying of objects (see Section 2). To illustrate the method and to verify the effectiveness of presented classifiers, we have performed several experiments on the data sets obtained from Kent Ridge Biomedical Dataset, UC Irvine Machine Learning repository and the website of The Elements of Statistical Learning book (see Section 3).

1.1 Classic discretization tree

In this paper we consider the supervised discretization based on the creation of the so-called decision tree of the local discretization (see, e.g., [4]). It is a binary tree, created by multiple binary partitions of the set into two groups of objects (e.g., patients) with the value of the selected

attribute. Because this method is well known from literature (see, e.g., [6,13]), in this paper we call this method as the *classic method*.

The method of choosing an attribute and its value (for numeric attributes often called the cut), that we use in the partition, is a key element of the discussed local discretization tree construction method and should involve the analysis of decision attribute values for training objects. Thus, one of the most important concepts presented in the strategy is a binary partition of the object set based on the attribute and its value. Formally, a *cut* is a pair (a, v) that are defined for a given *decision table* $\mathbf{A} = (U, A, d)$ in Pawlak's sense (see, e.g., [15]), where $a \in A$ (A is a set of attributes or columns in the data set) and v is the value of the attribute a that defines the partition of a set of attribute's values into two subsets. For numeric attributes, a cut (a, v) defines a partition of a set of objects into two subsets, the first set is a set of objects for which the a attribute value is less than v, and the second set of objects for which the a attribute value is greater than or equal to v. Instead, for symbolic attributes the first one is a set of objects for which the value of the attribute a is equal to v, and the second set is a set of objects for which the a attribute value is different from v.

Moreover, any cut (a, v) defines two templates, where by a template we understand a description of some set of objects. In case of numerical attributes, the first template defined by a cut (a, v) is described by a formula $T_{(a,v)} = (a(u) < v)$ and the second pattern defined by a cut (a, v) by formula $\neg T_{(a,v)} = (a(u) \geq v)$. In case of symbolic attributes, the first template defined by a cut (a, v) is designated by a formula $T_{(a,v)} = (a(u) = v)$ and the second pattern defined by a cut (a, v) by $\neg T_{(a,v)} = (a(u) \neq v)$.

As a measure of the binary partition quality, for example, the number of pairs of objects distinguishable by partition and having different values of the decision attribute can be used. For example, if a partition (a, v) divides objects into two sets of sizes M and N, and the first of these collections have M_0 and M_1 objects from the class C_0 and C_1 respectively, and in the second one we have N_0 and N_1 objects from the decision class C_0 and C_1, then the number of pairs of objects dicerned by the partition is given by: $N_1 \cdot M_0 + M_1 \cdot N_0$. If we determine the value of this measure for all possible cuts, then we can greedily choose one of the cuts and divide the entire set of objects into two parts on its basis. Of course, this approach can be easily generalized to the case of more than two decision classes.

It should be noted that this measure of the quality of the binary partition of objects set can be calculated for the given cut in time $O(n)$, where n is the number of objects in decision table (see, e.g., [6]). But determining the optimal cut requires the calculation of quality measures for all potential cuts. For this purpose it is necessary to check all potential cuts, including all conditional attributes in a specific order. This can be done using various methods. One of such methods for numerical attributes firstly sorts the objects of the given attribute for which we seek the optimal partition. This allows to determine the optimal cut in a linear manner.

Sorting a collection of objects results in the fact that the calculation of the optimal partition is done in time $O(n \cdot log\ n \cdot m)$, where n is the number of objects, and m is the number of conditional attributes. That is the method we have implemented in our own computational library RS-lib.

The quality of cuts may be computed for any subset of a given set of objects. In the local strategy of discretization, after finding the best cut and dividing the objects set into two subsets of objects (matching both templates mentioned above for a given cut), this procedure is repeated for each objects set separately until a stop condition holds.

At the beginning of the procedure we have the whole set of objects at the root of the tree. Then, we recursively apply the same splitting procedure to the emerging parts that we assign to tree nodes at higher and higher levels. Stop condition of partition is designed so that the given part is not divided (becomes a tree leaf) if it contains only objects from one decision class (optionally the objects from the given class constitute a certain percentage, which is treated as a parameter of the method) or the considered cut does not have any effect, i.e., there are no new pairs of objects from different decision classes separated by the cut.

In this paper, we assume that the partition stops when all objects from the current set of objects belong to the same decision class. Hence, the local strategy can be realized by using *decision tree* (see Figure 1).

The decision tree computed during local discretization can be treated as a classifier for the concept C represented by decision attribute from a given decision table $\mathbf{A}$. Let u be a new object

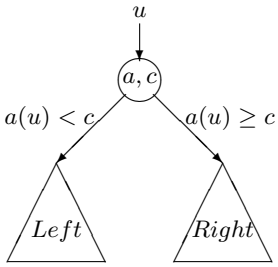

Fig. 1. The decision tree used in local discretization

and $\mathbf{A}(T)$ be a subtable containing all objects matching the template T defined by the cut from the current node of a given decision tree (at the beginning of an algorithm run T is the template defined by the cut from the root). We classify object u starting from the root of the tree as follows:

Algorithm *Classification by decision tree* (see [6])
Step 1 If u matches template T found for $\mathbf{A}$
then: go to subtree related to $\mathbf{A}(T)$
else: go to subtree related to $\mathbf{A}(\neg T)$.
Step 2 If u is at the leaf of the tree then go to 3
else: repeat 1-2 substituting $\mathbf{A}(T)$
(or $\mathbf{A}(\neg T)$) for $\mathbf{A}$.
Step 3 Classify u using the decision value attached to the leaf

A simple method for constructing the tree described above can be configured in many ways. For example, one can change the cut quality, which may also be made by introducing the domain knowledge. This often leads to the improvement of the classifier performance (see [4]).

The quality of a given cut computed as a number of object pairs discerned by this cut and belonging to different decision classes was used in [5] and the classifier constructed with usage of this method we call here as the *RS-C* classifier.

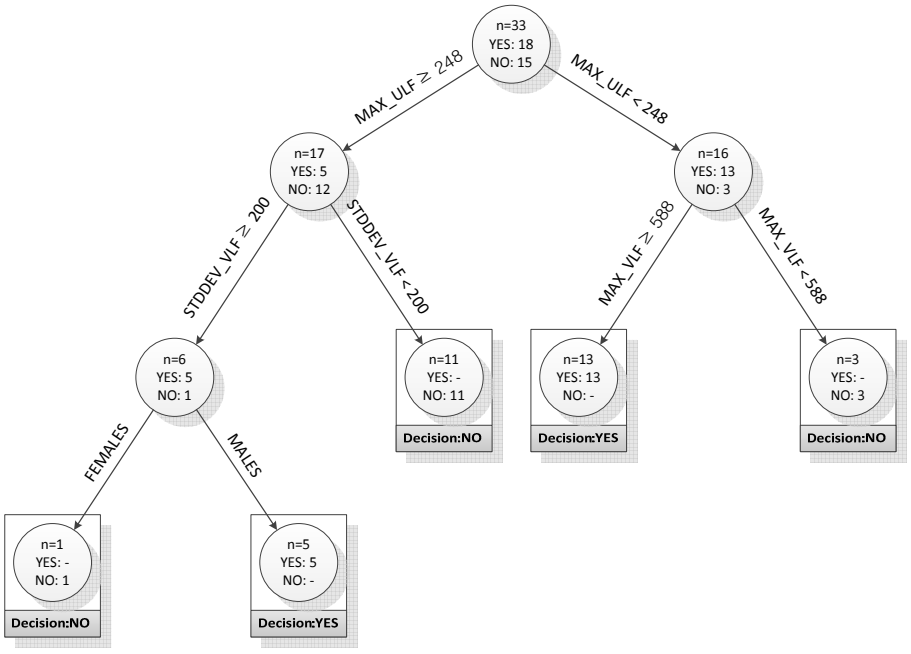

Fig. 2. An exemplary RS-C classifier

Figure 2 presents an exemplary *RS-C* classifier for the problem of forecasting coronary stenosis presence on the basis of medical data set (see [4]). Note that the above decision tree can be treated directly as a classifier, as test objects can be classified by stating to which leaf of the tree they belong. This is possible because, thanks to the designated partitions, one can trace membership of an object in the path from the root to the leaf, and then classify the object to the decision class whose objects dominate in the leaf.

Sample application of the tree is classification of real life objects. For example, for a patient with maximal ULF, i.e. power in ultra low frequency equal to 112 ms^2 and maximal VLF (very low frequency) equal 256, we follow from the root of the tree, down to the right subtree, as the patient suits the template $MAX\_ULF < 248$. Then, in the next step we decide again to go to the right subtree, which consists of one node, called leaf, where we stop. The fitting path indicates that the coronary arteries of that patient are not significantly narrowed by atherosclerosis. For a man with maximal ULF equal to 605 and standard deviation of VLF equal 509.6, we anticipate the atherosclerotic coronary arteries stenosis presence.

2 Decision tree with verifying cuts

When applying the approach described in the previous sections to the construction of classifiers for the case of data with a large number of attributes, there are serious doubts as to the validity of this approach. The point is that the method chooses only one split with the best quality of the established measure, at the given step of searching for optimal binary partitions. In other words, just one from among perhaps many partitions with high quality is chosen, while the others are ignored. Obviously, in the next step, the subsequent optimal binary split is selected, but not for the entire input set of objects, only for each previously obtained part separately. Such an approach is often effective leading to efficient classifiers, if the number of attributes is small and the attributes carry diverse information (e.g., in terms of a diverse positive area with respect to decision attribute). However, data with a large number of attributes can contain a lot of attributes that bear similar but significantly different information about objects. These attributes will be here called additional or redundant attributes. The domain experts (such as medical doctors) who in their daily work observe the redundancy of attributes, realize that and use it, e.g., making diagnosis more secure through the use of several attributes simultaneously. Meanwhile, the above-mentioned greedy method, out of the plurality of redundant attributes selects only one, eliminating others. However, in practice, the classification of the test objects may reveal the object that for some reason should not be classified according to the partition specified by the greedy algorithm (e.g., an outlier, from the point of view of an attribute selected by the greedy algorithm in the current node of the decision tree)

Therefore, the possibility of the object classification by greedily designated binary partition would require a confirmation by other attributes, which in the above method is not done. As a consequence a situation may occur where, e.g., for the microarray data with very many attributes and few objects the method finds only a few partitions (on several attributes), that are sufficient to create a tree allowing us for an unambiguous classification of objects from the training sample. This situation is very difficult to accept for the domain experts who are not able to come to terms with such a large reduction of the knowledge contained in the attributes and not using the rejected redundant attributes when creating the tree.

Serious doubts arise also from a statistical point of view. Typically, a set of objects is too small to call it representative. Therefore, the disposal of the information contained in the rejected attributes may lead to overfitting of the constructed tree to the objects from the training sample.

Some might say that the problem could be solved by a method calculating greedily in one step k-partitions, which at given stage of the algorithm best distinguish pairs of objects in terms of the fixed criterion (for one or more attributes). Unfortunately, no such algorithm is known with the time complexity of less than a square one (with regard to the number of objects). Such an algorithm would have to optimize k at a given stage of its operation, examining different subsets of potential partitions, which would increase its complexity. Due to efficiency, in the work we are interested in the complexity of the algorithms of the order at most $n \cdot log\ n$ with regard to the number of objects n.

The basic idea described in this paper lies in the fact that at a given stage of searching for partitions of a set of attributes, after determining the optimal binary partition of a set of objects, the family of k-binary verifying partitions is determined (for other attributes than the attributes used in the optimal partition), which as similar as possible to the optimal partition, distinguish between pairs of objects of different decision classes.

The idea behind it is as follows. As discussed above, the optimal partition is a tool to classify partially the test object, i.e., it provides information on where the test object should be sent to classify: to the right subtree or to the left one. Each partition of the family of verifying k-partitions divides objects from the training table like the optimal partition. Thus, if a test object to be classified will be submitted by the optimal partition to the left tree, there is a presumption that it should be directed similary by the verifying partition. If so, this increases our confidence that the optimal partition correctly classified the test object (as it is possible in a given node).

However, if it is not so, i.e., for instance, the optimal partition directs the test object to the left tree, and the verifying partition to the right, then there may be uncertainty about the performance of our classifier. Therefore, in this situation, a caution is recommended in planning further classifier actions. This precaution in the case of our method is revealed by the fact that the object is oriented to the classification by both the left and right subtree. After receiving the results of the classification, the possible conflict between the received decisions is resolved. Of course, there may be more then one verifying partition, and therefore the methodology outlined above must take this into account.

Another issue is the question of how verifying partitions interfere with the formation of the tree for a training table. In the classical method of creating a tree (see Section 1.1), on a given stage of the tree construction, the optimal partition of a set of objects into two disjoint sets is determined, for which subtrees are separately created in subsequent stages. However, in the present method, the split of a set of objects may not be a partition. On the one hand, as before, the optimal partition divides a set of training objects into two disjoint sets, but there may be a number of objects that match the pattern defined on the basis of the optimal partition, but does not match any of the patterns defined for verifying partitions. Similarly, there may be objects that do not match the pattern defined on the basis of the optimal partition, but match any of the patterns defined for verifying partitions.

One can tell about such objects that already at the stage of the tree construction, it is doubtful that the pattern based on the determined optimal partition is appropriate to classify this type of objects. Therefore, when creating a tree, the objects will be included into both sets of objects, the one intended to create a left subtree, as well as the set of objects intended to create a right subtree. This corresponds to the intuition that learning to classify this type of object is somehow postponed and transferred to both subtrees, where for their classification new partitions will be counted (perhaps better suited to these objects than optimal partition counted in the current tree node).

Below is presented the algorithm for construction of a decision tree, which formalizes the above considerations. Due to the fact that the algorithm uses verifying binary partitions, the decision tree produced by this algorithm will be called V-decision tree.

Assume that a decision table $\mathbf{A} = (U, A, d)$ and a parameter k belonging to natural numbers appear at the input of the algorithm.

Algorithm *V-decision tree construction*

Step 1 Find the optimal binary partition p in the table $\mathbf{A}$

Step 2 Find a collection of binary partitions p_1, ..., p_k in the table $\mathbf{A}$ which verify the partition p, such that $\frac{Dis(p,p_i)}{Dis(p)} > t$, where $Dis(p)$ is a number of pairs of objects from different decision classes discerned by partition p, whilst $Dis(p, p_i)$ is a number of pairs of objects from different decision classes discerned by both partitions p and partition p_i (for $i = 1, ..., k$) and t is a fixed threshold (t was equal 0.9 in our experiments).

Step 3 Split the table $\mathbf{A}$ into two subtables $\mathbf{A}(T_p)$ and $\mathbf{A}(\neg T_p)$ such that $\mathbf{A}(T_p)$ contains objects matching a pattern T_p, and $\mathbf{A}(\neg T_p)$ contains objects matching a pattern $\neg T_p$.

Step 4 Assign $\mathbf{A}_l = \mathbf{A}(T_p)$ and $\mathbf{A}_r = \mathbf{A}(\neg T_p)$.

Step 5 Determine all objects in the table $\mathbf{A}$, which match the pattern T_p and do not match a pattern T_{p_i} (for $i \in \{1, ..., k\}$) or match the pattern $\neg T_p$ and do not match a pattern $\neg T_{p_i}$ (for $i \in \{1, ..., k\}$), and attach these objects both

to the table $\mathbf{A}_l$ and $\mathbf{A}_r$ (if they are not there yet)

Step 6 If the tables $\mathbf{A}_l$ and $\mathbf{A}_r$ satisfy the stop condition, then finish tree construction
else repeat steps 1-4 for all subtables which do not satisfy the stop condition.

The stop condition mentioned in the above algorithm is the same as in the algorithm discussed in Subsection 1.1. Note that the only element of the above algorithm, which would increase the order of magnitude in complexity of computing time compared to the classical algorithm of Section 1.1 is step 2, in which the collection of k binary partitions verifying the partition p is computed. We show that this step can be accomplished in time $O(n \cdot log\ n \cdot m)$, where n is the number of objects and m the number of conditional attributes and therefore does not increase the computational time complexity of the algorithm in relation to the algorithm of Section 1.1.

It's not hard to see that for symbolic attributes, which typically have a small number of values, the determination of the best verification split can be done in time $O(n \cdot l)$, where l is the number of symbolic attribute values. Somewhat more difficult situation occurs when the verification partition for numerical attribute is calculated. Below we present the algorithm for selection of verification partition for the designated earlier binary partition p, assuming that the verification split is determined by a numerical attribute. For ease of discussion, assume that there are only two decision classes C_0 and C_1 in the data. This approach can of course be easily generalized to the case of more than two classes of decision-making.

Algorithm *Selection of verifying cut*

Step 1 Sort the values of the numerical attribute a

Step 2 Browsing the a attribute values from the smallest to the largest
determine for each appearing cut c the following numbers and place them
in a memory about cuts M:
$L(a, c, C_0)$ - number of objects from decision class C_0 with values of attribute a
smaller than c and at the same time matching the pattern T_p,
$L(a, c, C_1)$ - number of objects from decision class C_1 with values of attribute a
smaller than c and at the same time matching the pattern T_p.

Step 3 Browsing the a attribute values from the highest to the lowest
determine for each appearing cut c the following numbers and place them
in a memory of information about cuts M:
$H(a, c, C_0)$ - number of objects from decision class C_0 with values of attribute a
greater than or equal to c and at the same time matching the pattern $\neg T_p$.
$H(a, c, C_1)$ - number of objects from decision class C_1 with values of attribute a
greater than or equal to c and at the same time matching the pattern $\neg T_p$.

Step 4 Reviewing the information memory about cuts M, for each cut c, using information
about the cutting from the memory M, determine the optimum cutting such that
together with partition p distinguishes the largest number of pairs of objects from
different decision classes, and for each cut c calculate the number of such pairs
by formula: $L(a, c, C_0) \cdot H(a, c, C_1) + L(a, c, C_1) \cdot H(a, c, C_0)$.

Assuming that the memory about cuts M is accessible in constant time, the above algorithm runs in time $O(n \cdot log\ n)$, where n is the number of objects (due to the sorting of objects on the basis of the a attribute).

Now we provide the algorithm for an object classification, using a V-tree with verifying partitions. Suppose we classify the object u at a node where the optimal cut c and the family of verifying cuts c_1, ..., c_k were found. Let T denote the pattern for c, and T_1, ..., T_k the patterns for c_1, ..., c_k. The classification is performed according to the following algorithm.

Algorithm *Classification by V-decision tree*

Step 1 If a node satisfies the stop condition, return the decision fixed in the tree node and terminate

Step 2 Assign k_1: = the number of patterns from the collection T_1, ..., T_k,
to which the object u fits

Step 3 Assign $k_2 = k - k_1$

Step 4 If object u matches the pattern T and $k_1 = k$, then send it for classification
by the subtree constructed for the table $\mathbf{A}(T)$, to obtain the value of the decision d_1, and return d_1
otherwise
if the object u does not match the pattern T, and $k_2 = k$, then send u for classification
by subtree designed for a table $\mathbf{A}(\neg T)$, to obtain the value of the decision d_2, and return d_2
otherwise
classify object u by node $\mathbf{A}(T)$ to obtain the value of the decision d_1.
classify object u by node $\mathbf{A}(\neg T)$ to obtain the value of the decision d_2.
If $d_1 = d_2$ then return d_1.
otherwise
if $k_1 > k_2$ return d_1
otherwise return d_2.

The classifier constructed with the use of V-decision tree will be called here the *RS-V* classifier. Note that the above algorithm to classify the object in the node utilizes a single tree only when all verifying splits classify the object just as a main partition p. In other cases, the classification is done by both subtrees. Then the following two cases are considered. The first case refers to the situation when the two subtrees returned the same decision value. Then the value of the node is returned as the decision. The second case refers to a situation where one of the subtrees returned one decision value, and the second subtree the other one. Then that node returns a decision coming from the subtree, which is associated with a greater number of such verifying patterns that classify a test object for this tree.

3 Experiments and results

To verify the quality of classifiers based on verifying cuts, we have implemented the algorithms from the library RS-lib, which is an extension of the RSES-lib library forming the kernel of the RSES system [7].

The experiments have been performed on the data sets obtained from Kent Ridge Biomedical Dataset ([12]), UC Irvine (UCI) Machine Learning repository (see [22]) and website of The Elements of Statistical Learning book (Statweb)(see [9]).

From the first source there comes 6 collections which cover the microarray experiments in lung cancer [11]), lymphoma ([1]), ALL-AML leukemia ([10]), colon tumors ([2]), ovarian cancers ([18]) and prostate tumors ([19]). The experiments were conducted on the merged original training and testing data sets.

Table 1. Experimental data set details

Microarray data	Objects	Attributes	Classes
Colon	62	2000	2
Leukemia	72	7129	2
Lymphoma	47	4026	2
Lung cancer	181	12533	2
Ovarian cancer	253	15154	2
Prostate	136	12600	2

UCI and Statweb data	Objects	Attributes	Classes
biodeg	1055	41	2
CHD	462	9	2
Parkinson	185	23	2
spam	4601	58	2

The next sets of data derived from UCI (the first two) and Statweb are related to such topics as: Parkinson disease, the biodegradation of molecules, coronary heart disease (CHD) and spam e-mail. Table 1, shows the summary of the characteristics of the data sets.

The aim of conducted experiments was to check the quality of the classification algorithms described in this paper. Here we present the experimental results of presented methods. For testing quality of classifiers we applied 10 fold cross-validation (CV) technique. In this metod, a data set is divided into ten equinumerous subsets (folds). In each test, nine folds of data are used for training and the remaining one for testing. The procedure is repeated 10 times. The final result of the algorithm is the average of the 10 trials.

As a measure of classification success (or failure) we use the following parameters well known from literature: the accuracy (ACC) and the coverage (COV).

In Table 2 we give the results of experiments in applying presented classification methods to chosen data.

Table 2. Average ACC and COV of data sets with classification algorithms based on 10 fold CV

Method	*RS-C* classifier		*RS-V* classifier	
	Accuracy	Coverage	Accuracy	Coverage
Colon	0.712	1	0.773	1
Leukemia	0.830	1	0.907	1
Lymphoma	0.802	0.938	0.890	1
Lung cancer	0.920	1	0.926	1
Ovarian cancer	0.979	1	0.988	0.999
Prostate	0.794	1	0.796	1
biodeg	0.810	1	0.814	1
CHD	0.630	1	0.647	1
Parkinson	0.887	1	0.888	1
spam	0.893	1	0.896	1

4 Conclusion

In the paper we presented the new method of a decision tree building which makes use of the plurality of redundant attributes. The method simulates the behavior of domain experts in decision making, e.g. doctors who increase the certainty of making diagnosis through the use of several attributes simultaneously. The assesement of the approach was conducted using ten datasets. The novelty of the paper is important because the experimental results show that the employment of the knowledge contained in the redundant attributes increases the quality of the classifiers. For all datasets the accuracy of the proposed classifier was better compared to the classical method, from an insignificant increase (0.1%) to almost 9%. We observed that better results (ACC increase of 6.1%, 7.7% and 8.8%) were achieved for microarray data (for half of the data sets) which are characterized by a large number of attributes and a small number of objects.

The conducted experiments constitute the initial steps of investigating of the new method. We expect that the method may be used in various fields. We plan further experimental work involving the improvement of the method, e.g. the optimization of its parameters.

Acknowledgement

This work was partially supported by the Polish National Science Centre grant DEC-2013/09/B/ST6/01568 and by the Centre for Innovation and Transfer of Natural Sciences and Engineering Knowledge of University of Rzeszów, Poland.

References

1. Alizadeh, A., Eisen, M., Davis, R., Ma, C. et. al.: Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. Nature 403, 503–511 (2000)
2. Alon, U. et al.: Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. PNAS 96, 6745–6750 (1999)
3. Bazan, J.G.: Hierarchical classifiers for complex spatio-temporal concepts. Transactions on Rough Sets, IX, LNCS 5390, 2008, pp. 474–750.

4. Bazan, J.G., Bazan-Socha, S., Buregwa-Czuma, S., Pardel, P.W., Sokolowska, B.: Predicting the presence of serious coronary artery disease based on 24 hour Holter ECG monitoring. In: M. Ganzha, L. Maciaszek, M. Paprzycki (eds.), *Proceedings of the Federated Conference on Computer Science and Information Systems*, 2012, pp. 279-286, IEEE Xplore - digital library.
5. Bazan, J.G., Bazan-Socha, S., Buregwa-Czuma, S., Pardel, P.W., Sokolowska, B.: Prediction of coronary arteriosclerosis in stable coronary heart disease, Proceedings of the Fourteen Conference of Information Processing and Management of Uncertainty in Knowledge-Based Systems, 2012, Springer-Verlag, Communications in Computer and Information Science, vol. 298, part 9, Springer, 2012, pp. 550-559.
6. Bazan, J. G., Nguyen, H. S., Nguyen, S. H., Synak, P., Wróblewski, J.: Rough set algorithms in classification problems. In: L. Polkowski, T. Y. Lin, S. Tsumoto (eds.), "Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems," *Studies in Fuzziness and Soft Computing*, Springer-Verlag/Physica-Verlag, vol. 56, 2000, pp. 49–88.
7. Bazan, J. G., Szczuka, M.: The Rough Set Exploration System. Transactions on Rough Sets, III, LNCS 3400, 2005, pp. 37–56.
8. Doherty, P., Łukaszewicz, W., Skowron, A., Szałas, A.: Knowledge Engineering: A Rough Set Approach. Springer, Heidelberg, Germany, 2006.
9. The Elements of Statistical Learning repository, `http://statweb.stanford.edu/ tibs/ElemStatLearn/datasets/`
10. Golub, T., Slonim, D., Tamayo, P. et al.: Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. Science 286, 531–537 (1999)
11. Gordon, G., Jensen, R., Hsiao, L., Gullans, S. et al.: Translation of microarray data into clinically relevant cancer diagnostic tests using gene expression ratios in lung cancer and mesothelioma. Cancer Research 62, 4963–4967 (2002)
12. Kent Ridge Biomedical Dataset repository, `http://datam.i2r.a-star.edu.sg/datasets/krbd/`
13. Nguyen, H. S.: Approximate Boolean Reasoning: Foundations and Applications in Data Mining, Transactions on Rough Sets, **V**, LNCS 4100, 2006, pp. 334–506.
14. Nguyen, Hung, S., Skowron, A., Stepaniuk, J.: Granular computing: A rough set approach. In *Computational Intelligence*, **17** (3):514–544, 2001.
15. Pawlak, Z., Skowron, A.: Rudiments of rough sets. Information Sciences, 177, 2007, pp. 3–27.
16. Pedrycz, W., Skowron, A., Kreinovich, V. (eds.) Handbook of Granular Computing, John Wiley & Sons, The Atrium, Southern Gate, Chichester, England. 2008.
17. Peters, J. F.: Classification of objects by means of features. In: D. Fogel, J. Mendel, X. Yao, T. Omori (Eds.), *Proc. 1st IEEE Symp. Foundations of Computational Intelligence*, (FOCI'2007), Honolulu, Hawaii, 2007, pp. 1–8.
18. Petricoin, E., Ardekani, A., Hitt, B., Levine, P. et al.: Use of proteomic patterns in serum to identify ovarian cancer. The Lancet 359, 572–577 (2002)
19. Singh, D. et al.: Gene expression correlates of clinical prostate cancer behavior. Cancer Cell 1, 203–209 (2002)
20. Skowron, A.: Toward intelligent systems: Calculi of information granules. In T. Terano, T. Nishida, A. Namatame, S. Tsumoto, Y. Ohsawa, and T. Washio, editors, *New Frontiers in Artificial Intelligence, Joint JSAI 2001 Workshop Post-Proceedings*, volume 2253 of *Lecture Notes in Artificial Intelligence*, pages 251–260, Matsue, Japan, May 20–25 2001. Springer-Verlag.
21. Skowron, A.: Towards granular multi-agent systems. In Sankar K. Pal and Ashish Ghosh, editors, *Soft Computing Approach to Pattern Recognition and Image Processing*, pages 215–234. World Scientific, 2002.
22. UC Irvine Machine Learning Repository, `http://archive.ics.uci.edu/ml/`