Constructing A Classifier by Searching The Ranking Space

Kazuya HARAGUCHI

Department of Information Technology and Electronics, Faculty of Science and Engineering, Ishinomaki Senshu University, Ishinomaki 986-8580

Abstract

In this paper, we consider the classification problem, one of the significant issues in machine learning and artificial intelligence. Motivated by the fact that most of the classifier models inherently utilize the notion of ranking, we propose a framework of constructing a classifier by searching the ranking space. Specifically, we assume the existence of a true ranking and construct such a binary decision tree that induces a ranking close to the true one. We validate our strategy by showing the effectiveness of the decision trees constructed by the algorithm; our decision trees are competitive with those generated by C4.5, in terms of generalization error.

1　Introduction

We consider the *classification problem*. In this problem, we are given a *training set*, which is a set of *examples*. Each example is represented by an n-dimensional vector of numerical and/or nominal values and is given its *class*, which is either *positive* (+) or *negative* (−). The goal of the problem is to construct a function from the example space to the class set {+, −} so that the function can predict the classes of "unseen examples" with high accuracy. The function to be constructed is called a *classifier*. The classification problem is one of the most significant issues in machine learning[1-5] and has many applications in data mining[6] and in pattern recognition[7].

In this paper, we introduce a novel strategy for the classification problem and validate the strategy by showing the effectiveness of a binary decision tree classifier that is constructed by a learning scheme based on it.

The main keyword of our strategy is *ranking*. As pointed out in previous studies[8-11], many existing classifiers are equipped with *ranking functions* either explicitly or implicitly. A ranking function induces a ranking on the training set, from the most "likely" negative example to the most "likely" positive example. For example, we take up hyperplane, the most prevalent classifier model, which is represented by a pair (w, θ) of an n-dimensional vector $w=(w_1, w_2, ..., w_n)\in\mathbb{R}^n$ and a threshold $\theta\in\mathbb{R}$. It classifies a numerical example $x=(x_1, x_2, ..., x_n)$ by which side of the hyperplane $wx+\theta$ the example x exists in, that is, the hyperplane classifies x into the class decided by;

$$\mathrm{sgn}\left(\sum_{i=1}^{n} w_i x_i+\theta\right),$$

where $\mathrm{sgn}(\cdot)$ denotes a function that returns (+) if the value in the parentheses is positive and (−) otherwise. Then the value $wx+\theta$ serves as the ranking function that gives the non-decreasing order of examples with respect to $wx+\theta$. Taking into account that the absolute value $|wx+\theta|$ represents the distance between the hyperplane (w, θ) and the example x, we can regard the head example in the ranking as the most likely negative example, and the tail example as the most likely positive example.

Based on the observation that many classifiers induce a ranking on the training set, we try to construct a classifier by searching the ranking space. Specifically, assuming the existence of the true ranking, we search for such a classifier that induces a ranking close enough to the true one. The classifier model we employ here is *binary decision tree* since it is relatively easy to

establish the connection between a classifier and a ranking. As most of the similar algorithms do, our binary decision tree construction algorithm consists of two processes, *branching* and *pruning*. To decide how we extend the decision tree by branching the leaves, and to decide the degree to which we prune the tree, we introduce novel criteria coming from the view that a binary decision tree induces a ranking on the training set. To validate our strategy, we show that binary decision trees constructed by our algorithms are competitive with those constructed by C4.5[12], a conventional decision tree generator, through computational experiments using benchmark data sets from UCI Repository of Machine Learning[13]. Many classifier construction algorithms have adjustable parameters, by which we may tell the algorithms our "subjective view" on the data. Although our algorithm does not have such parameters and C4.5 has ones, ours works so well as C4.5. This indicates the possibility of our strategy based on ranking to construct a classifier.

The paper is organized as follows. We present the background and related works in more detail in Section 2. After preparing terminologies and notations in Section 3, we explain how to construct a binary decision tree classifier based on our strategy in Section 4. Then in Section 5, we show some experimental results to validate the strategy. Finally, we present concluding remarks in Section 6.

2 Background

We are pursuiting a novel learning model from a nominal data set such that an example is represented by a vector of only nominal values. We pay attention to the order relation between examples, and attempt to embed its concept in the learning model. There are two typical strategies to construct a classifier from a nominal data set. One strategy transforms the nominal data set into a numerical data set and then constructs a hyperplane based classifier like *support vector machine*. This strategy treats the data set in a distance space and thus inherently utilizes the order relation between examples. The other strategy partitions the example space (i.e., the Cartesian product of the attribute domains) into subspaces and assigns either (+) or (−) to each subspace. This does not utilize the order relation between examples in general. We claim that our new learning model should be between the two typical strategies since ours does not utilize the concept of distance but only employs the order relation. In this paper, we give realization of such learning model in more general settings in which we can treat not only nominal data but also numerical data.

In the last decade, various attempts have been made that utilize the ranking function to improve the classifier performance[8-11]. The ranking function of a classifier can be evaluated by the *area under ROC curve* (*AUC*). AUC indicates the probability by which a random positive example is ranked after a random negative example. A classifier with a high AUC is expected to perform well. It is known that AUC is a more robust criterion against the change of the class distribution than error rate. The previous studies employ the notion of AUC maximization to improve the classifier performance. As we will see later, the notion of AUC maximization is also employed in our decision tree construction algorithm.

Recently, due to its various applications in the Internet context (e.g., search engines, recommendation), ranking itself is often treated as the target of machine learning. For example, Freund, Iyer, Schapire and Singer[14] formulated the problem of learning a ranking from a given preference list and applied the *boosting* to the problem. This is not the machine learning problem that we treat in this paper.

Decision tree construction has been studied by researchers from artificial intelligence and related fields over 20 years. For detail, see the survey in the paper[15]. We find many algorithms implemented in Weka[16], the well-known open source machine learning software.

3 Preliminaries

In this section, we prepare notations and terminologies. We denote a training set by $X=\{x_1, x_2, ..., x_m\}$, where m denotes the number of examples in the training set. Each example $x_i \in X$ has n values for n *attributes* and is represented by an n-dimensional vector of numerical and/or nominal values, that is, $x_i=(x_{i,1}, x_{i,2}, ..., x_{i,n})$. The domain of an attribute $j=1, 2, ..., n$ is denoted by D_j. For example, if the attribute j is numerical, then $D_j=\mathbb{R}$, and if the attribute j' is nominal, then $D_{j'}$ is a finite countable set of unordered elements. Let X^+ (resp., X^-) denote the set of positive (resp., negative) examples in X. So we have $X=X^+ \cup X^-$ and $X^+ \cap X^-=\emptyset$.

We denote a binary decision tree by $T=(V, L; \{l_v, r_v, (j_v, b_v)\}_{v\in V}, \{c_w\}_{w\in L})$, where V denotes the set of the inner nodes, and L denotes the set of the leaves. The binary decision tree T has the root node, and each inner node has exactly two children. For each inner node $v \in V$, the l_v, $r_v \in V \cup L$ represent the left child and the right child of v, respectively. Each v is associated with a *branching rule* that is represented by a pair of j_v and b_v; the integer $j_v \in \{1, 2, ..., n\}$ represents the index of an attribute, and b_v is a mapping such that $b_v : D_{j_v} \to \{0, 1\}$. In other words, the function b_v gives either 0 or 1 to an example x according to the value x_{j_v}. For b_v of a numerical attribute j_v, we restrict ourselves to the class of *threshold functions*, that is, with a real threshold $\theta \in \mathbb{R}$, the function gives $b_v(x_{j_v})=1$ (resp., 0) if $x_{j_v} > \theta$ (resp., $x_{j_v} \leq \theta$). For b_v of a nominal attribute j_v, we concentrate on such a function that is defined by a subset $D \subseteq D_j$ and that gives $b_v(x_{j_v})=1$ (resp., 0) if $x_{j_v} \in D$ (resp., $x_{j_v} \in (D_j \backslash D)$). Each leaf $w \in L$ is labeled either positive (+) or negative (−). The value $c_w \in \{+, -\}$ represents the class of a leaf w.

A classifier is a function $D_1 \times D_2 \times \cdots \times D_n \to \{+, -\}$, that is, it outputs either (+) or (−) in response to an input example x. The binary decision tree T works as a classifier as follows. Starting from taking the root as v, we evaluate the value $b_v(x_{j_v})$ for the current inner node v. If $b_v(x_{j_v})=0$ (resp., 1), then we update v to the left child l_v (resp., right child r_v). This process is repeated until we visit a leaf. Then T outputs the class c_w of the reached leaf w.

For a leaf w of a binary decision tree T, let us denote by $X(w)$ the set of the examples in X that visit the leaf w when given to T as input. We also denote by $X^+(w)$ and $X^-(w)$ the sets of the positive and negative examples in $X(w)$, respectively.

A *ranking on the training set* X is a function $\pi : X \to \{1, 2, ..., m\}$. For any $x \in X$, the value $\pi(x)$ represents the rank of x. We assume any ranking to be *partial*, that is, we may have a tie $\pi(x)=\pi(x')$ for some $x, x' \in X$ $(x \neq x')$. The *degree* of π is defined as the size of the range. We denote the degree of π by $\deg(\pi)$. So we have $\deg(\pi) \in \{1, 2, ..., m\}$. We say that π *separates* X if $\pi(x^-) < \pi(x^+)$ holds for any negative example $x^- \in X^-$ and positive example $x^+ \in X^+$. In other words, any negative example is ahead of all the positive examples with respect to π.

For a partial ranking π and any ordered pair $(x, x') \in X \times X$ of examples, we define $\delta_\pi(x, x') = -1$ if $\pi(x) < \pi(x')$, $\delta_\pi(x, x')=1$ if $\pi(x) > \pi(x')$, and $\delta_\pi(x, x')=0$ otherwise. Note that δ_π satisfies $\delta_\pi(x, x') = -\delta_\pi(x', x)$. To measure the closeness between two partial rankings π and π', we introduce a generalization of Kendall-tau distance[17,18]. Let us define;

$$\Delta_{\pi,\pi'}(x, x') = \frac{1}{2} |\delta_\pi(x, x') - \delta_{\pi'}(x, x')|$$

Thus we have $\Delta_{\pi,\pi'}(x, x') \in \left\{0, \frac{1}{2}, 1\right\}$. Denoted

by $d(\pi, \pi')$, the Kendall-tau distance between π and π' is defined as follows;

$$d(\pi, \pi') = \sum_{(x, x') \in X \times X} \Delta_{\pi, \pi'}(x, x'). \quad (1)$$

Bansal and Fernández-Baca[19] presented two algorithms to compute $d(\pi, \pi')$, where one runs in $O(m \log m / \log\log m)$ time and the other runs in $O(m \log(\min\{\deg(\pi), \deg(\pi')\}))$ time.

A decision tree induces a ranking on the training set X such that the examples visiting the same leaf have the same rank, those visiting different leaves have different ranks, and the leaves are ordered appropriately. We denote the number of the leaves by $k=|L|$. Then there are $k!$ rankings possible. Among these, we employ the ranking that is obtained by ordering the leaves in the non-increasing order of $|X^-(w)|/|X^+(w)|$. More precisely, we order the k leaves $w_1, w_2, \ldots, w_k$ so that;

$$\frac{|X^-(w_1)|}{|X^+(w_1)|} \geq \frac{|X^-(w_2)|}{|X^+(w_2)|} \geq \cdots \geq \frac{|X^-(w_k)|}{|X^+(w_k)|}. \quad (2)$$

We describe the reason why we utilize this one. In our study, we evaluate a decision tree by the Kendall-tau distance between the induced ranking π and a certain separating ranking π'. The Kendall-tau distance of (1) is decomposed as follows.

$$\begin{aligned} d(\pi, \pi') = & \sum_{(x, x') \in X^+ \times X^+} \Delta_{\pi, \pi'}(x, x') \\ & + \sum_{(x, x') \in X^- \times X^-} \Delta_{\pi, \pi'}(x, x') \\ & + 2 \sum_{(x, x') \in X^+ \times X^-} \Delta_{\pi, \pi'}(x, x'). \end{aligned}$$

When π' is separating, the summation in the last term is equal to $W^<(\pi) + \frac{1}{2} W^=(\pi)$, where;

$$W^<(\pi) = \sum_{l^+=1}^{k-1} |X^+(w_{l^+})| \sum_{l^-=l^++1}^{k} |X^-(w_{l^-})|,$$

$$W^=(\pi) = \sum_{l=1}^{k} |X^+(w_l)||X^-(w_l)|.$$

Besides, let us define $W^>(\pi)$ as follows;

$$W^>(\pi) = \sum_{l^+=2}^{k} |X^+(w_{l^+})| \sum_{l^-=1}^{l^+-1} |X^-(w_{l^-})|. \quad (3)$$

We easily see that $W^<(\pi) + W^=(\pi) + W^>(\pi) = |X^+||X^-|$ holds. Since $W^=(\pi)$ is a constant regardless of the choice of the ordering on the leaves, we should maximize $W^>(\pi)$ in (3) in order to minimize $W^<(\pi) + \frac{1}{2} W^=(\pi)$. The ratio $W^>(\pi)/(|X^+||X^-|)$ is equivalent to AUC, which was mentioned in Section 2. Then we have a good reason to maximize $W^>(\pi)$. We can easily show that, among all the possible $k!$ rankings, the one in (2) achieves the maximum value of $W^>(\pi)$. Based on these, we employ the ranking defined by (2).

4 Learning Algorithm

In this section, we present the algorithm to construct a binary decision tree classifier based on our strategy for the classification problem. First we describe what kind of decision tree we would like to have. We should construct a decision tree so that it classifies the classes of unseen examples as accurately as possible. We assume the existence of the true model.

Assumption 1 *We assume that there is a classifier that classifies the class of any example correctly.*

We call this classifier the *true classifier*. We would like such a decision tree that is close to the true classifier in a sense, but of course, we do not know its exact form. From the observation so far on the relationship between a classifier and the induced ranking, we make an assumption on the true classifier.

Assumption 2 *We assume that the true classifier induces a separating ranking.*

We call this ranking the *true ranking*. We attempt to construct a decision tree such that the ranking induced in the way of (2) is close enough to the true ranking. Of course, however, we do not know the exact form of the true ranking. Then we make an additional assumption on the true ranking based on *Occam's Razor*[20-22], the belief employed in the scientific

research of various fields, also in machine learning, stating that entities should not be multiplied unnecessarily.

Assumption 3 *The degree of the true ranking is "relatively small."*

We do not go into the detail of what the term "relatively small" exactly means. The degree of the true ranking can be from 2 to m, and our intention in Assumption 3 is that it should not necessarily be as large as m.

Now we describe our algorithm to construct a decision tree. The algorithm consists of two processes, branching and pruning. In the branching process, we search for a decision tree that induces a separating ranking. This is because we assume in Assumption 2 that the true ranking is separating and it would not be natural to complete the construction task without finding such a decision tree that induces a separating ranking. The branching process starts from a single node tree and branches the leaves iteratively (i.e., giving new two children to each leaf) until the resulting decision tree induces a separating ranking. Let v denote any leaf of an intermediate decision tree. We give a branching rule (j_v, b_v) to v in order to attach two children to the leaf v, denoted by l_v and r_v. To the leaf v, we choose the branching rule (j_v, b_v) that minimizes the distance between the induced ranking and the nearest separating ranking, where the ranking is focused on the subset $X(v)$ of the examples. Divided by (j_v, b_v), $X(l_v)$ (resp., $X(r_v)$) becomes the subset of $X(v)$ such that any example $\boldsymbol{x}$ in the subset satisfies $b_v(\boldsymbol{x}_{j_v})=0$ (resp., 1). Also we have;

$$X^+(v) = X^+(l_v) \cup X^+(r_v),$$

$$X^-(v) = X^-(l_v) \cup X^-(r_v).$$

We assume that the following inequality holds without loss of generality;

$$\frac{|X^-(l_v)|}{|X^+(l_v)|} \geq \frac{|X^-(r_v)|}{|X^+(r_v)|}. \tag{4}$$

We denote the distance between the induced ranking and the nearest separating ranking by $f_v(j_v, b_v)$. One can readily see that $f_v(j_v, b_v)$ is defined as follows.

$$\begin{aligned} f_v(j_v, b_v) = & 2|X^+(l_v)||X^-(r_v)| \\ & + |X^-(l_v)||X^+(r_v)| \\ & + |X^-(r_v)||X^+(l_v)|. \end{aligned} \tag{5}$$

The branching process terminates if the decision tree induces a separating ranking.

Then, in the pruning process, we prune the leaves of the decision tree iteratively as long as the distance between the induced ranking and the *binary separating ranking* gets smaller. Denoted by β, the binary separating ranking gives a rank to an example $\boldsymbol{x} \in X$ as follows.

$$\beta(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in X^-, \\ 1 & \text{otherwise.} \end{cases}$$

We denote the k leaves of the decision tree by $w_1, w_2, \ldots, w_k$ which are ordered in the way of (2). The distance $d(\pi, \beta)$ is defined as follows;

$$\begin{aligned} d(\pi, \beta) = & -2W^>(\pi) - \frac{1}{2}\sum_{l=1}^{k} |X(w_l)|^2 \\ & + \text{const.} \end{aligned} \tag{6}$$

If the decision tree has a leaf such that removing the leaf results a smaller $d(\pi, \beta)$ than the current tree, then we remove the leaf that attains the smallest $d(\pi, \beta)$. Otherwise, we terminate the pruning process. Clearly, β is the separating ranking that has the smallest degree. From Assumption 3, we would like the decision tree to get close to β. After the pruning process, the resulting decision tree may not induce a separating ranking but is expected to have a smaller size. It is interesting to see that, to minimize the distance of (6), we should maximize $W^>(\pi)$ in the 1st term which is proportional to AUC, and at the same time, the summation in the 2nd term which may represent the "simplicity" of the decision tree.

Finally, we summarize the algorithm that we described so far in Algorithm 1.

Algorithm 1 Algorithm to construct a decision tree

Branching process
1: Take a decision tree that consists of a single node, denoted by v
2: $L \leftarrow \{v\}$ ▷ L is the set of leaves
3: **while** there is $v \in L$ such that $X^+(v) \neq \emptyset$ and $X^-(v) \neq \emptyset$ **do**
4: Compute the branching rule (j_v, b_v) that minimizes the distance $f_v(j_v, b_v)$ in (5) under the assumption of (4)
5: $L \leftarrow (L \cup \{l_v, r_v\}) \setminus \{v\}$
6: **end while**
Pruning process
7: $d^* \leftarrow d(\pi_T, \beta)$, where π_T denotes the ranking induced by the decision tree T
8: **while** the decision tree has a leaf w such that $d(\pi_{T-w}, \beta) < d^*$ **do**
9: Compute the leaf w that minimizes $d(\pi_{T-w}, \beta)$
10: $d^* \leftarrow d(\pi_{T-w}, \beta)$, $T \leftarrow T - w$
11: **end while**

5 Computational Experiments

In this section, we present some experimental results to validate our strategy. In the experiments, we compare the "generalization ability" between the decision tree constructed by our algorithm and the one constructed by C4.5[12]. The generalization ability is measured by the error rate of the classifier.

We show the results in Table 1. We took 18 benchmark data sets from UCI Repository of Machine Learning[13]. The 2nd column n indicates the number of attributes of each data set. The 3rd and 4th columns indicate the numbers of examples in the training set and the test set, respectively. The 5 data sets named ADULT, IONO, MONKS-1, MONKS-2 and MONKS-3 have their own test sets. For these data sets, we construct a classifier from the training set and measure the error rate on the test set. For the other data sets, we estimate the error rate by the average over 50 iterations of 10-fold cross validation[1, 23]. Some data sets contain examples with missing values, and we removed such examples in advance.

Our algorithm does not have any adjustable parameter, and we show the error rates of two decision trees in the table, unpruned one (i.e., one that is constructed by the branching process and does not undergo the pruning process) and pruned one. On the other hand, C4.5 has various adjustable parameters. Among these, the *confidence level* is one of the most influential parameter since it decides the degree of pruning. We set the confidence level to 1%, 10%, 25%, 50%, 75% and 99% while we set the other parameters to the default values. In the column for C4.5, we show the smallest error rate among those observed by different values of the confidence level. Although we give C4.5 such advantages, our algorithm and C4.5 are competitive; ours outperforms (resp., ties and loses to) C4.5 in 8 (resp., 2 and 8) data sets. Our algorithm is rather straightforward and has much room for improvement. Nevertheless, it is competitive with C4.5, a conventional decision tree generator. This should validate our strategy of constructing a classifier by searching the ranking space.

6 Concluding Remarks

In this paper, we introduced a novel strategy for the classification problem; we assume the true ranking and search for such a classifier that induces a ranking close enough to the true one. We assert that our experimental results support the validity of this strategy. For future works, we should improve the decision tree construction algorithm further. To realize this, we need a sophisticated model of the ranking space that admits us to conduct an effective search. Such a model may require new parameters to control the search.

Acknowledgments

This work is supported by JSPS KAKENHI Grant Number 22700015.

References

(1) Weiss SM, Kulikowski CA (1991) Computer systems that learn: classification and prediction methods from statistics, neural nets, machine

Table 1: Experimental results on 18 data sets from UCI Repository [13]

Data set	n	#Train	#Test	Error rates (%)		
				Ours		C4.5
				(Unpruned)	(Pruned)	
ADULT	14	32561	16281	20.26	24.57	**13.78**
BCW	9	699	-	4.85	6.83	**4.75**
CHESS	36	3196	-	**0.46**	4.72	0.57
CREDIT	15	690	-	18.46	**13.76**	13.88
GLASS	9	214	-	8.41	7.58	**6.75**
HABERMAN	3	306	-	34.98	**26.85**	28.05
HEART	13	270	-	24.74	24.32	**22.53**
HEPATITIS	19	155	-	18.04	**15.86**	16.14
IONO	34	200	151	16.56	**5.30**	8.61
MONKS-1	6	124	432	**3.70**	16.67	21.99
MONKS-2	6	169	432	**21.30**	32.87	32.87
MONKS-3	6	122	432	10.19	**2.78**	**2.78**
MUSHROOM	22	8124	-	**0.00**	1.48	**0.00**
PIMA	8	768	-	29.81	26.77	**25.63**
SPAMBASE	57	4601	-	8.91	10.46	**7.16**
TICTACTOE	9	958	-	**5.44**	17.59	14.31
VOTING	16	435	-	6.00	4.39	**3.65**
WDBC	30	569	-	6.61	7.3	**6.26**

learning, and expert systems. Morgan Kaufmann.

(2) Anthony M, Biggs N (1992) Computational learning theory. Cambridge University Press.

(3) Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT Press.

(4) Vapnik VN (2000) The nature of statistical learning theory (2nd edition). Springer.

(5) Schoelkopf B, Smola AJ (2001) Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT Press.

(6) Liu H, Motoda H (1998) Feature extraction, construction and selection: a data mining perspective. Kluwer Academic Publishers.

(7) Ripley BD (1996) Pattern recognition and neural networks. Cambridge University Press.

(8) Macskassy S, Provost F (2004) Confidence bands for ROC curves: methods and an empirical study. Proceedings of ROCAI-2004 in ECAI-2004.

(9) Wu S, Flach P (2005) A scored AUC metric for classifier evaluation and selection. Proceedings of ICML 2005 workshop on ROC analysis in machine learning.

(10) Ataman K, Street WN (2006) Learning to rank by maximizing AUC with linear programming. Proceedings of IJCNN 2006: 123-129

(11) Flach P, Matsubara ET (2007) A simple lexicographic ranker and probability estimator. Proceedings of ECML 2007: 575-582

(12) Quinlan JL (1993) C4.5: programs for machine learning. Morgan Kaufmann.

(13) Frank A, Asuncion A (2010) UCI Machine Learning Repository (http://archive.ics.uci.edu/ml). University of California, Irvine, School of Information and Computer Sciences.

(14) Freund Y, Iyer R, Schapire RE, Singer Y (2003) An efficient boosting algorithm for combining preferences. Journal of machine learning research 4: 933-969

(15) Garofalakis M, Hyun D, Rastogi R, Shim K (2003) Building decision trees with constraints. Data mining and knowledge discovery 7:187-214

(16) Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. SIGKDD explorations 11-1

(17) Fagin R, Kumar R, Mahdian M, Sivakumar D, Vee E (2006) Comparing partial rankings. SIAM journal on discrete mathematics 20-3: 628-648

(18) Kemeny JG, Snell JL (1973) Mathematical models in the social sciences (2nd edition). MIT Press.

(19) Bansal MS and Fernández-Baca D (2009) Computing distances between partial rankings. Information processing letters 109: 238-241

(20) Gamberger D, Lavrač N (1997) Conditions for Occam's razor applicability and noise elimination. Proceedings of ECML '97: 108-123

(21) Nolan D (1997) Quantitative parsimony. The British journal for the philosophy of science 48-2: 329-343

(22) Zahálka J, Železný F (2011) An experimental test of Occam's Razor in classification. Machine learning 82: 475-481

(23) Li DC, Fang YH, Fang YMF (2010) The data complexity index to construct an efficient cross-validation method. Decision support systems 50: 93-102