A Monotonic Measure for Optimal Feature Selection

Huan Liu[1] and Hiroshi Motoda[2] and Manoranjan Dash[3]

[1] Dept of Info Sys & Comp Sci, National University of Singapore, Kent Ridge, Singapore 119260.
[2] Division of Intelligent Sys Sci, Osaka University, Ibaraki, Osaka 567, Japan.
[3] BioInformatics Centre, National University of Singapore, Lower Kent Ridge, Singapore 119074.

Abstract. Feature selection is a problem of choosing a subset of relevant features. Researchers have been searching for optimal feature selection methods. 'Branch and Bound' and Focus are two representatives. In general, only exhaustive search can bring about the optimal subset. However, under certain conditions, exhaustive search can be avoided without sacrificing the subset's optimality. One such condition is that there exists a monotonic measure with which 'Branch and Bound' can guarantee an optimal subset. Unfortunately, most error- or distance-based measures are not monotonic. A new measure is employed in this work that is monotonic and fast to compute. With this measure, the search for relevant features is guaranteed to be complete but not exhaustive. An empirical study is conducted to show that the algorithm indeed lives up to what it claims. Some discussion is given at the end.

1 Introduction

The basic problem of classification is to classify a given pattern (example) to one of m known classes. A pattern of features presumably contains enough information to distinguish among the classes. When a classification problem is defined by features, the number of features (N) can be quite large. A classifier may encounter problems to learn something meaningful because the required amounts of data ($\mathcal{N}$, or the number of patterns) increase exponentially in proportion with N [4]. The task of feature selection is to determine which features to select in order to achieve maximum performance with the minimum measurement effort [3]. Reducing the number of features directly alleviates the measurement effort. Performance for a classifier can be its predictive accuracy, i.e., *1 - error rate*.

As was mentioned in [3], if the goal is to minimize the error rate, and the measurement cost for all the features is equal, then the most appealing function to evaluate the potency of a feature to differentiate between the classes is the Bayes Classifier [20]. Due to the inductive nature of classification problems, no full distribution of data can be obtained[4]. Extensive research effort was devoted

[4] If it is obtainable, we should use the Bayes Classifier under normal circumstances.

to the investigation of other functions (mostly based on distance and information measures, or simply on classifiers) for feature evaluation. If there exist N features, to find an optimal subset of features without knowing how many features are relevant, it requires to explore all the 2^N subsets. When N is large, this exhaustive approach is out of the question. Therefore, various feature selection methods have been designed to avoid exhaustive search while still aiming at the optimal subset. Examples are Branch & Bound [15, 17], Relief [7, 10], Wrapper methods [8], Approximate Markov Blanket [9], and LVF [12]. We will review some of these methods briefly in the next section.

The feature selection problem can be viewed as a search problem [18, 17, 11]. The search process starts with either an empty set or a full set. For the former, it expands the search space by adding one feature at a time (Sequential Forward Selection) [17]; for the latter, it expands the search space by deleting one feature at a time (Sequential Backward Selection) [15]. As we shall see, the best alternative to exhaustive search is Branch & Bound like algorithms if there exists a monotonic function of evaluating features. Assuming we have subsets $\{S_0, S_1, ..., S_n\}$, we have a measure U that evaluates each subset S_i. The monotonicity condition requires that:

$$S_0 \supset S_1 \supset ... \supset S_n \Rightarrow U(S_0) \leq U(S_1) \leq ... \leq U(S_n).$$

In this case, the search can be complete but not exhaustive. That means it need not exhaustively search the whole space but the optimal subset is guaranteed. Many distance and information based measures have been shown to be non-monotonic [18]. Many researchers pointed out that the only remaining alternative is to use the error rate of a classifier as the measure. Among many classifiers, however, only the Bayes Classifier satisfies this monotonicity condition[5] because other classifiers adopt some assumptions and employ certain heuristics [18, 3, 6]. Another disadvantage of using the error rate as a measure in the wrapper models of feature selection is it is slow to compute. For example, to construct a decision tree, it would take at least $O(\mathcal{N} \log \mathcal{N})$.

This work proposes a measure that is monotonic as well as fast to compute $(O(\mathcal{N}))$. In the following, we review the related work, introduce an automatic Branch & Bound algorithm for feature selection which does not require a predetermined number M of relevant features. We also give a proof outline that the proposed measure is monotonic and use a simple example to show how the algorithm works. Section 4 is about the choice of experimental methods and data sets, and about the results and analysis. In Section 5, we offer some discussion.

2 Related Work

The feature selection problem has long been the research topic in statistics and pattern recognition, but most work in this area has dealt with linear regression [11]. Since early 90's, it has received considerable attention from machine learning researchers. Some divide the feature selection methods into two

[5] But it requires the full distribution of the data.

regimes [11, 8]. One is of Wrapper Models that basically use the classifier's error rate as the measure U. The other is of Filter Models that use computationally less costly evaluation tools based on distance, information, consistency, etc. In this work, we focus on some filter models that explicitly aim at searching for optimal feature subsets. Two reviewed groups are (1) Sequential Feed-forward Selection; and (2) Sequential Backward Selection.

Focus [2] is one of the earliest algorithms within machine learning. Focus starts with an empty set and carries out breadth-first search until it finds a minimal subset that predicts pure classes. If the full set has three features, the root is $(0, 0, 0)$, its children are (0 0 1), (0 1 0), and (1 0 0). It works on binary, noise-free data. It is exhaustive search in nature. A similar approach is taken by [17]. A systematic search is carried out, starting with the empty set and adding features until it finds a subset consistent with the training data (the concept of determination). A reliability measure is used as a heuristic to verifying a determination.

Branch & Bound for feature selection was first proposed in [15] and later reviewed in [18]. It starts with a full set of features, removes one feature at a time. If the full set contains three features, the root is (1 1 1). Its child nodes are (1 1 0), (1 0 1), and (0 1 1), etc. Without restrictions on expanding nodes in the search space, this could lead to exhaustive search. However, if each node is evaluated by a measure U and an upper limit is set for the acceptable values of U, then Branch & Bound backtracks whenever an infeasible node is discovered. If U is monotonic, no feasible node is omitted as a result of early backtracking and, therefore, the gained savings in the search time do not violate the optimality of the selected subset. As was pointed out in [18], the measures used in [15] have disadvantages (non-monotonicity is one). The authors of [18] proposed the concept of *approximate branch and bound* due to Branch & Bound's attractive characteristics as to relax the condition of monotonicity.

The focus of this paper is to avoid exhaustive search while ensuring the search is complete so that an optimal subset is guaranteed. In cases where the optimality of a solution is not paramount, there exist some heuristic and random feature selection algorithms. We briefly introduce some here, many more can be found in [5]. Relief [7] is a feature weight based algorithm inspired by instance-based learning algorithms [1]. Relief assigns a weight to each feature that reflects its ability to distinguish among the classes, and then selects those features with weights that exceed a user-specified threshold. Another algorithm which does not explicitly search exhaustively is LVF [12] that randomly searches the feature space. For each candidate subset, it calculates an inconsistency count based on the intuition that the class label associated with the maximum number of patterns is most probably the correct class, considering only the features in the subset. The two methods in this group employ either re-sampling of data or random generation of subsets. They are different from the methods in the first two groups.

3 A Non-exhaustive yet Complete Search Algorithm

In this section, we present a monotonic measure for feature evaluation; elaborate on an Automatic Branch & Bound algorithm with technical details about its implementation.

3.1 A monotonic measure

For two subsets of features, S_i and S_j, one is preferred to the other based on a measure U of feature-set evaluation. S_i and S_j are indifferent if $U(S_i) = U(S_j)$ and $\#(S_i) = \#(S_j)$ where $\#$ is the cardinality; S_i is preferred to S_j if $U(S_i) = U(S_j)$ but $\#(S_i) < \#(S_j)$, or if $U(S_i) < U(S_j)$ and $\#(S_i) \leq \#(S_j)$. As we know, the condition for Branch & Bound to work optimally is that U is monotonic.

In this work, U is an *inconsistency rate* over the data set given S_i. The inconsistency rate is calculated as follows: (1) two patterns are considered inconsistent if they match all but their class labels, for example, patterns (0 1 1) and (0 1 0) match with respective to the first two attributes, but are different in the last attribute (class label); (2) the inconsistency count is the number of all the matching patterns minus the largest number of patterns of different class labels: for example, there are n matching patterns, among them, c_1 patterns belong to label_1, c_2 to label_2, and c_3 to label_3 where $c_1 + c_2 + c_3 = n$. If c_3 is the largest among the three, the inconsistency count is $(n - c_3)$; and (3) the inconsistency rate is the sum of all the inconsistency counts divided by the total number of patterns ($\mathcal{N}$). By employing a hashing mechanism, we can compute the inconsistency rate approximately with a time complexity of $O(\mathcal{N})$.

Now we give a proof outline to show that this inconsistency rate measure is monotonic, i.e., if $S_i \subset S_j$, then $U(S_i) \geq U(S_j)$. Since $S_i \subset S_j$, the discriminating power of S_i can be no greater than that of S_j. As we know, the discriminating power is reversely proportional to the inconsistency rate. Hence, the inconsistency rate of S_i is greater than or equal to that of S_j, or $U(S_i) \geq U(S_j)$. The monotonicity of the measure can also be proved as follows. Consider three simplest cases of $S_k (= S_j - S_i)$ without loss of generality: (i) features in S_k are irrelevant, (ii) features in S_k are redundant, and (iii) features in S_k are relevant. (We consider here data without noise and discuss noisy data later.) If features in S_k are irrelevant, based on the definition of irrelevancy, these extra features do not change the inconsistency rate of S_j since S_j is $S_i \cup S_k$, so $U(S_j) = U(S_i)$. Likewise for case (ii) based on the definition of redundancy. If features in S_k are relevant, that means S_i does not have as many relevant features as S_j. Obviously, $U(S_i) \geq U(S_j)$ in the case of $S_i < S_j$. It is clear that the above results remain true for cases that S_k contains irrelevant, redundant as well as relevant features.

3.2 Automatic Branch & Bound (ABB)

ABB is a Branch & Bound algorithm with its bound set to the inconsistency rate δ of the data set with the full set of features. It starts with the full set

of features S^0, removes one feature from S_j^{l-1} in turn to generate subsets S_j^l where l is the current level and j specifies different subsets at the lth level. If $U(S_j^l) > U(S_j^{l-1})$, S_j^l stops growing (the branch is pruned), otherwise, it grows to level $l+1$, in other words, one more feature will be removed. In short, ABB seeks the smallest S_j whose inconsistency rate is δ. S is the full feature set and D the data set.

```
δ = inConCal(S, D);
ABB (S, D) {
    /* subset generation */
    For all feature f in S {
        S1 = S − f; /* remove one feature at a time */
        enQueue(Q, S1);} /* add at the end */
    while notEmpty(Q) {
        S2 = deQueue(Q); /* remove at the start */
        if (S2 is legitimate ∧ inConCal(S2, D) ≤ δ)
            /* recursion */
            ABB (S2, D); }}
```

The essence of the algorithm is shown above. inConCal() calculates the consistency rate of data given a feature subset. Care has been taken in implementing the algorithm such that (1) no duplicate subset will be generated via proper enumeration; and (2) no child node of a pruned node will be generated by ensuring that the Hamming distance between a new subset at the current level and any pruned subset at the parent level is greater than 1[6] (this is the legitimacy test in ABB).

It is not required anymore to specify the size of a desired subset, M,or a bound for the measure as normally required by Branch & Bound. At the end of search, we just need to report the legitimate subsets with the smallest cardinality as the optimal subsets. This is because of using the inconsistency measure. Any subset with its inconsistency rate larger than δ is out for sure. The algorithm is named "ABB" since M is automatically determined.

3.3 An example

The working of ABB is best explained in detail through an example. Let's consider a simple example in which a data set is described by four features, assuming only the first two are relevant. The root $S_0 = (1\ 1\ 1\ 1)$ of the search tree is a binary array with four '1's. Refer to Figure 1. Following ABB, we expand the root to four child nodes by turning one of the four '1's into '0' (L2 in Figure 1). All these four are legitimate child nodes because the root is a valid node. The child nodes expanded from the current parent may be illegitimate if they are also children of some pruned nodes. The four nodes are $S_1 = (1\ 1\ 1\ 0)$, $S_2 = (1\ 1\ 0\ 1)$, $S_3 = (1\ 0\ 1\ 1)$, and $S_4 = (0\ 1\ 1\ 1)$. Since one of the relevant features is missing,

[6] A full set of N attributes entails an N-bit binary array in which ith value 1 means ith attribute is chosen to include in the subset.

$U(S_3)$ and $U(S_4)$ will be greater than $U(S_0)$ where U is the inconsistency rate on the given data. Hence, the branches rooted by S_3 and S_4 are pruned and will not grow further. Since ABB is a breadth-first search algorithm, it expands S_1 first. Following our enumeration procedure[7], we now have three more new nodes (L3 in Figure 1). That is $S_5 = (1\ 1\ 0\ 0)$, $S_6 = (1\ 0\ 1\ 0)$, and $S_7 = (0\ 1\ 1\ 0)$. However, S_6 and S_7 are illegitimate since they are also children of pruned nodes S_3 and S_4 respectively. This can be determined by the Hamming Distance (HD) of two nodes, e.g., S_6 and S_3. Since their HD is 1, S_6 is also a child of S_3. Only when a new node passes the legitimacy test will its inconsistency rate be calculated. Doing so improves the efficiency of ABB because $\mathcal{N}$ (number of patterns) is normally much larger than N (number of attributes). The rest of the nodes $(S_8, ..., S_{11})$ are generated and tested in the same spirit.

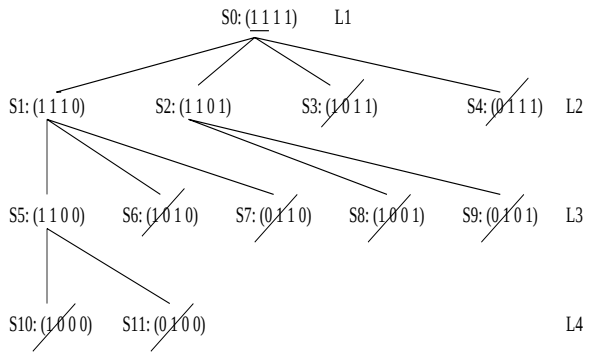

Fig. 1. A simple example with four attributes. The first two are relevant.

3.4 Handling noise

The existence of noise affects the measure of feature evaluation. There are two types of noise in general. Type I noise is about inconsistencies in the data. That is, two patterns are the same but do not have the same class labels. This type of noise is naturally handled by ABB since it calculates the bound (δ) before searching for subsets. Type II noise is some patterns with their class labels consistently wrongly labeled. Obviously type II noise may not cause inconsistencies in the data. *A priori* knowledge about type II noise is required in order to handle it. If the knowledge is available, ABB can handle type II noise by modifying δ based on how much type II noise is in the data. We will come back to this issue

[7] The procedure is: find the first '0' from the left end as the marker; create child nodes by changing one '1' at a time to '0' from right to left until each '1' to the left side of the marker is changed in turn.

in discussion after the empirical study through an example (Monk3) in which we explicitly allowerd some inconsistency although there was no inconsistency in the training data.

4 Empirical Study

The objectives of this empirical study are to verify:

1. ABB indeed finds optimal subsets for various data sets, and
2. features selected are good for various learning algorithms.

To verify whether ABB indeed finds optimal subsets we select two groups of data sets: one with known relevant features and the other with unknown relevant features as described next. All data sets are from the UC Irvine data repository [13] unless specified otherwise.

For the first group we compare the output subsets of ABB with the known subsets. For the second group we compare the outputs of ABB with that of Focus, a popular method in literature that guarantees optimal subsets. For the second objective we choose two different learning algorithms: a decision tree method (C4.5 [16]) and a standard back-propagation neural network (SNNS [21]).
1st group of data sets, with known relevant attributes, consists of *CorrAL* data designed in [6], *Monks* data in [19] (CorrAL and Monks have separate data sets for training and testing), and *Par3+3+3* data which is a parity-3 problem and has 9 attributes: the first three are relevant, the middle three are irrelevant, and the last three are redundant. We should expect as many as 8 equally good subsets of three features. The whole data set has 512 patterns.
2nd group of data sets, with unknown relevant attributes, consists of *WBC* - the Wisconsin Breast Cancer data set, *LED-7* - data with 7 Boolean attributes and 10 classes, the set of decimal digits (0..9), *Letter* - the letter image recognition data, *LYM* - the lymphography data, and *Vote* - the U.S. House of Representatives Congress-persons on the 16 key votes.

The experiments are designed to observe:

1. optimal subsets for various data sets found by ABB. The optimality of subsets is verified by Focus and prior knowledge,
2. reduction of error rate for C4.5 inductive classifier with optimal subsets, and
3. reduction of mean square error (MSE) for SNNS [21] neural network classifier with optimal subsets.

In Table 1, two thirds of the data is taken for selecting features using ABB and Focus. The rest of the data is used for testing purpose if not otherwise specified.

Table 1 shows that ABB indeed finds optimal subsets as validated by Focus and *a priori* knowledge. Focus does breadth first search starting from the empty set and stops after reaching the first consistent subset. In fact, the subset found by Focus is one of the solutions of ABB.

Data	D_{Tr}	D_{To}	C	N	M	Focus/Prior Knowledge	ABB (One Set)
CorrAL	32	64	2	6	4	A_1, A_2, A_3, A_4	A_1, A_2, A_3, A_4
Monk1	124	432	2	6	3	A_1, A_2, A_5	A_1, A_2, A_5
Monk2	169	432	2	6	6	A_1 - A_6	A_1 - A_6
Monk3	122	432	2	6	3	A_2, A_4, A_5	A_2, A_4, A_5
Par3+	341	512	2	9	3	A_1, A_2, A_3	A_7, A_8, A_9
WBC	463	699	2	9	4	A_1, A_2, A_6, A_7	A_1, A_2, A_6, A_7
LED-7	400	600	10	7	5	A_1, A_2, A_3, A_4, A_5	A_1, A_2, A_3, A_4, A_5
Letter	5980	8968	26	16	9	A_1, A_2, A_7 - A_{13}	A_1, A_2, A_7 - A_{13}
LYM	100	148	4	18	6	A_2, A_{13} - A_{16}, A_{18}	A_2, A_{13} - A_{16}, A_{18}
Vote	300	435	2	16	8	A_1 - $A_4, A_9, A_{11}, A_{13}, A_{16}$	A_1 - $A_4, A_9, A_{11}, A_{13}, A_{16}$

Table 1. Reduction in number of features. D_{Tr} - training set, D_{To} - tiotal set, C - number of classes, N - number of original features; M - number of selected features.

While comparing the results of ABB and Focus, we found an interesting fact that *"ABB and Focus complement each other with respect to time taken to reach optimal subset"*. To verify this we conducted another set of experiments to compare the portions of the search space evaluated by ABB and Focus for various data sets considering the sequential backward exhaustive search as reference (see Table 2). Notice that for those data sets for which ABB is quite efficient (Monk2, LED-7, Letter, Vote) Focus searches a comparatively large number of subsets. For example, for Monk2 data set ABB evaluates 7 subsets whereas Focus evaluates 63 subsets out of a maximum of 64. In contrast, there are data for which Focus is particularly good when compared with ABB; for example, Par3+3+3, WBC, and Lymphography (as shown bold faced in Table 2). The reason is that Focus starts the search from an empty set whereas ABB starts from the complete set. So, if the size of the optimal subset is not small ABB is a better choice, otherwise Focus is better than ABB. Hence, our finding is that ABB and Focus complement each other. To take adavantage of both algorithms one may run both simultaneously till any one of the two algorithms stops.

Based on the subsets found for each data set, we obtain the results shown in Tables 3 and 4. In Table 3, C4.5 gave better accuracy (except for Monk2 since no feature should be removed for this data set) of 10-fold cross validation. For the second data group, we notice that numbers of features were all reduced, error rates were decreased. But the results of 10-fold cross validation for tree size are mixed for both data groups, some showing larger tree sizes as pointed out by $\leftarrow$ in Table 3. As was observed by [14], smallest trees do not necessarily give the best predictive accuracy. What is observed here is that better accuracy may not mean a smaller tree size. We also noticed that in the "after" (feature selection) setting, in most cases, C4.5 used all features selected by ABB, which indicates that features selected by ABB are relevant in decision tree induction. However,

Data set	# All	ABB		Focus	
		# Evaluated	Ratio	# Evaluated	Ratio
CorrAL	64	14	0.22	42	0.66
Monk1	64	12	0.19	24	0.38
Monk2	64	7	0.11	63	1.00
Monk3	64	19	0.30	35	0.54
Par3+3+3	512	265	**0.51**	46	0.09
WBC	2^9	188	**0.37**	145	0.28
LED-7	2^7	9	0.07	99	0.77
Letter	2^{16}	1971	0.03	42,634	0.65
LYM	2^{18}	82,156	**0.31**	23,167	0.08
Vote	2^{16}	301	0.005	39,967	0.66

Table 2. Search space reduction. # All - number of all nodes in the search space for a data set, # Evaluated - number of nodes generated and evaluated, and Ratio is # Evaluated divided by #All.

C4.5 did choose features not selected by ABB in the "before" setting, e.g., in the case of CorrAL data.

	Tree Size		Error Rate %		
Data	Before	After	Before	After	
CorrAL	14.6	13.0	6.0	0.0	
Monk1	43.0	41.0	0.7	0.0	
Monk2	16.3	16.3	21.1	21.1	
Monk3	19.0	19.0	1.1	1.1	
Par3+3+3	13.0	**15.0**	17.2	0.0	$\leftarrow$
WBC	38.0	36.0	6.6	6.0	
LED-7	19.0	19.0	0.0	0.0	
Letter	6660.0	6113.0	28.1	27.9	
LYM	26.9	**29.6**	21.8	21.0	$\leftarrow$
Vote	16.0	**19.0**	2.8	2.3	$\leftarrow$

Table 3. The 10-fold cross validation results (tree size and error rate) of C4.5 before and after feature selection.

Running back-propagation neural network involves the setting of some parameters, such as the network structure (number of layers, number of hidden units), learning rate, momentum, number of CYCLES (epochs), etc. In order to focus our attention on the effect of feature selection by ABB, we try to minimize the tuning of the parameters for each data set. We fix the learning rate as 0.1,

the momentum as 0.5, one hidden layer, the number of hidden units as half of the original input units for all data sets. The experiment is carried out in two steps: (1) a trial run to find a proper number of CYCLES for each data set which is determined by a sustained trend of no decrease of error; and (2) two runs on data sets with and without feature selection via ABB respectively using the number of CYCLES found in step 1. Other parameters remain fixed for the two runs in step 2. As the output of SNNS is a real number, the class label in each data is converted to a binary pattern. If the data has 3 (0,1,2) classes then label '0' becomes output pattern '1 0 0' and so on. Finally the rror is measured by Mean Squared Error (MSE) by finding the difference in the predicted and the actual values. The results are shown in Table 4. In all cases MSE either decreases or is very close after feature selection.

			Before FS		After FS	
Data	CYCLES	#HU	N	MSE	M	MSE
CorrAL	1000	3	6	0.023	4	0.046
Monk1	1000	3	6	0.262	3	0.212
Monk2	1000	3	6	0.16	6	0.16
Monk3	1000	3	6	0.003	3	0.0
Par3+3+3	1000	5	9	0.311	3	0.0
WBC	1000	5	9	0.04	4	0.06
LED-7	1000	4	7	0.0	5	0.0
Letter	5000	8	16	0.374	9	0.273
LYM	7000	9	18	0.16	6	0.19
Vote	4000	8	16	0.038	8	0.024

Table 4. Back-propagation results of SNNS before and after feature selection, where #HU denotes number of Hidden Unit, N is number of original features, M is number of features selected by ABB.

5 Discussion and Conclusion

Time complexity This issue is directly related to how large the search space is. In other words, how many nodes (subsets) have been generated. The other factors about time complexity of ABB are (i) time complexity of inconsistency checking, which is $O(\mathcal{N})$; and (2) time complexity of legitimacy test, which is $O(N)$. But they are relatively the same for each node. So the key factor is the search space. Through our experiments and analysis, we realize that the search space of ABB is closely related to the number of relevant features. For instance, taking the simple example in Section 3.3, when two features out of four are relevant, the search space is 12 nodes; if all four features are relevant, it is 5 nodes, i.e., the root plus 4 child nodes. In general, the larger the size of optimal

subset, the smaller the search space due to early pruning of the illegitimate nodes.

Prior knowledge can come in many forms. One is about the noise in the data. As we know, this can be used to modify δ in ABB. δ is defined by the inconsistency found in the data with full features plus a certain percentage (prior knowledge) of noise. For the Monk3 data set, this percentage is 5% as δ for ABB.

We briefly reviewed the work of optimal feature selectors, and demonstrated that with a monotonic measure, Branch & Bound is the best deterministic algorithm (the search is not exhaustive, yet complete). We showed that the inconsistency rate is such a measure and it is fast to compute. The new method ABB is simple to implement and guarantees optimal subsets of features. Empirical study demonstrates that (1) ABB removes irrelevant, redundant, and/or correlated features even with the presence of noise (as in Monk3 with 5% noise); and (2) the performance of a classifier with the features selected by ABB also improves. Another finding from this study is Focus and ABB complement each other.

References

1. D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.
2. H. Almuallim and T. G. Dietterich. Learning with many irrelevant features. In *Proceedings of Ninth National Conference on AI*, pages 547–552, 1991.
3. M. Ben-Bassat. Pattern recognition and reduction of dimensionality. In P. R. Krishnaiah and L. N. Kanal, editors, *Handbook of statistics-II*, pages 773–791. North Holland, 1982.
4. A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Occam's razor. In J.W. Shavlik and T.G. Dietterich, editors, *Readings in Machine Learning*, pages 201–204. Morgan Kaufmann, 1990.
5. M. Dash and H. Liu. Feature selection methods for classifications. *Intelligent Data Analysis: An International Journal*, 1(3), 1997.
6. G.H. John, R. Kohavi, and K. Pfleger. Irrelevant feature and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann Publisher, 1994.
7. K. Kira and L.A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *AAAI-92, Proceedings Ninth National Conference on Artificial Intelligence*, pages 129–134. AAAI Press/The MIT Press, 1992.
8. R. Kohavi. *Wrappers for performance enhancement and oblivious decision graphs*. PhD thesis, Department of Computer Science, Standford University, Stanford, CA, 1995.
9. D. Koller and M. Sahami. Toward optimal feature selection. In L. Saitta, editor, *Machine Learning: Proceedings of the 13th International Conference*. Morgan Kaufmann Publishers, 1996.
10. I. Kononenko. Estimating attributes : Analysis and extension of RELIEF. In *Proceedings of European Conference on Machine Learning*, pages 171–182, 1994.
11. P. Langley. Selection of relevant features in machine learning. In *Proceedings of the AAAI Fall Symposium on Relevance*. AAAI Press, 1994.

12. H. Liu and R. Setiono. A probabilistic approach to feature selection - a filter solution. In L. Saitta, editor, *Machine Learning: Proceedings of the 13th International Conference*. Morgan Kaufmann Publishers, 1996.
13. C.J. Merz and P.M. Murphy. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html` . Irvine, CA: University of California, Department of Information and Computer Science, 1996.
14. P.M. Murphy and M.J. Pazzani. Exploring the decision forest: An empirical investigation of occam's razor in decision tree induction. *Journal of Art. Intel. Res.*, 1:257–319, March 1994.
15. P.M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Trans. on Computer*, C-26(9):917–922, September 1977.
16. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
17. J. C. Schlimmer. Efficiently inducing determinations : a complete and systematic search algorithm that uses optimal pruning. In *Proceedings of Tenth International Conference on Machine Learning*, pages 284–290, 1993.
18. W. Siedlecki and J Sklansky. On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2:197–220, 1988.
19. S.B. Thrun and et al. The monk's problems: A performance comarison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.
20. Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems That Learn*. Morgan Kaufmann Publishers, San Mateo, California, 1991.
21. Andreas Zell and et al. Stuttgart neural network simulator (snns), user manual, version 4.1. Technical Report 6/95, Institute for Parallel and Distributed High Performance Systems (IPVR), University of Stuttgart, FTP: ftp.informatik.uni-stuttgart.de/pub/SNNS, 1995.

This article was processed using the LaTeX macro package with LLNCS style