Consistency Based Feature Selection

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

[1] School of Computing, National University of Singapore, Singapore.
[2] Division of Intelligent Sys Sci, Osaka University, Ibaraki, Osaka 567, Japan.

Abstract. Feature selection is an effective technique in dealing with dimensionality reduction for classification task, a main component of data mining. It searches for an "optimal" subset of features. The search strategies under consideration are one of the three: complete, heuristic, and probabilistic. Existing algorithms adopt various measures to evaluate the goodness of feature subsets. This work focuses on one measure called *consistency*. We study its properties in comparison with other major measures and different ways of using this measure in search of feature subsets. We conduct an empirical study to examine the pros and cons of these different search methods using consistency. Through this extensive exercise, we aim to provide a comprehensive view of this measure and its relations with other measures and a guideline of the use of this measure with different search strategies facing a new application.

1 Introduction

Classification is an important data mining task. 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. Pattern classification is inherently connected to information reduction. Features can also be redundant or irrelevant. An irrelevant feature does not affect the underlying structure of the data in any way. A redundant feature does not provide anything new in describing the underlying structure. Because redundant and irrelevant information is cached inside the totality of the features, a classifier that uses all features will perform worse than a classifier that uses relevant features that maximize interclass differences and minimize intraclass differences [4]. Feature selection is a task of searching for "optimal" subset of features from all available features. Its motivation is three-fold: *simplifying* the classifier; *improving* the accuracy of the classifier; and *reducing* data dimensionality for the classifier. The last point is particularly relevant when a classifier is unable to handle large volumes of data.

Features may not be all relevant. In order to measure the usefulness (or goodness) of selected features, we need selection criteria. The class separability is often used as one of the basic selection criteria. When a set of features maximizes the class separability, it is considered well suited for classification. From a statistics viewpoint, five different measurements for class separability are analyzed in [8]: error probability, interclass distance, probabilistic distance, probabilistic dependence and entropy. Information-theoretic considerations [20] suggested something similar: using a good feature of discrimination provides compact descriptions of each class, and these descriptions are maximally distinct. Geometrically, this constraint can be interpreted to mean that (i) such a feature takes on nearly identical values for all examples of the same class, and (ii) it takes on some different values for all examples of the other class. In this work, we use a selection criterion that does not attempt to maximize the class separability but tries to retain the discriminating power of the data defined by original features. Feature selection is formalized as finding the smallest set of features that can distinguish classes as if with the full set. In other words, with a subset S of features, no two examples with the same values on S have different class labels [1]. We study the pros and cons of this measure in comparison with other measures. Another aspect of feature selection is related to the study of search strategies. Extensive research efforts have been devoted to this study [19, 7, 3]. Examples are Branch &Bound [16], Relief [11], Wrapper methods [12], and Las Vegas algorithms [14]. 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 (Forward Selection) - an example is Focus [1]; for the latter, it expands the search space by deleting one feature at a time (Backward Selection) - an example is 'Branch & Bound' [16].

The contributions of this paper are: (a) studying a monotonic criterion for feature selection w.r.t. other selection criteria; (b) exploring its properties and use in exhaustive (complete), heuristic, and probabilist search; (c) comparing its different uses with a number of data sets; and (d) suggesting a framework of when to use what. In the rest of the paper P is the number of patterns, N is the number of features, M is the size of relevant features, and m is the number of class labels.

2 Consistency Measure

Consistency can be interpreted as zero inconsistency. If we attain zero inconsistency, we achieve 100% consistency. Throughout this paper we use consistency and inconsistency interchangeably.

2.1 The Measure

The suggested measure U is an *inconsistency rate* over the data set for a given feature set. In the following description *pattern* means a set of values for the features in a candidate subset. The inconsistency rate is calculated as follows: (1) two patterns are considered inconsistent if they match all but their class labels, for example, an inconsistency is caused by two instances (0 1 a) and (0 1 $\bar{a}$) with different classes (a and $\bar{a}$); and (2) the inconsistency count for a pattern is the number of times it appears in the data minus the largest number among different class labels: for example, let us assume there are n matching patterns, among which 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 for all possible patterns of a feature subset divided by the total number of patterns (P). By employing a hashing mechanism, we can compute the inconsistency rate approximately with a time complexity of $O(P)$. Unlike the commonly used univariate measures in literature [18], this is a multivariate measure which checks a subset of features at a time.

2.2 Other Evaluation Measures

An optimal subset is always relative to a certain evaluation function. An optimal subset chosen using one evaluation function may not be the same as that using another evaluation function. Typically, an evaluation function tries to measure the discriminating ability of a feature or a subset to distinguish the different class labels. Blum and Langley [3] grouped different feature selection methods into two broad groups (i.e., filter and wrapper) based on their use of an inductive algorithm in feature selection or not. *Filter* methods are independent of an inductive algorithm, whereas *wrapper* methods are not. Ben-Bassat [2] grouped the evaluation functions until 1982 into three categories: information or uncertainty measures, distance measures, and dependence measures. Considering these divisions and latest developments, we divide the evaluation functions into five categories: *distance*, *information (or uncertainty)*, *dependence*, *consistency*, and *classifier error rate*. **Distance Measures** It is also known as separability, divergence, or discrimination measure. For a two-class problem, a feature X is preferred to another feature Y if X induces a greater difference between the two-class conditional probabilities than Y; if the difference is zero then X and Y are indistinguishable. An example is Euclidean distance. **Information Measures** These measures typically determine the information gain from a feature. The information gain from a feature X is defined as the difference between the prior uncertainty and expected posterior uncertainty using X. Feature X is preferred to feature Y if the information gain from feature X is greater than that from feature Y [2]. An example is entropy. **Dependence Measures** Dependence measures or correlation measures quantify the ability to predict the value of one variable from the value of another variable. Correlation coefficient is a classical dependence measure and can be used to find the correlation between a feature and a class. If the correlation of feature X with class C is higher than the correlation of feature Y with C, then feature X is preferred to Y. A slight variation of this is to determine the dependence of a feature on other features; this value indicates the degree of redundancy of the feature. All evaluation functions based on dependence measures can be divided between distance and information measures. But, these are still kept as a separate category because, conceptually, they represent a different viewpoint [2]. **Consistency Measures** This type of measures has been in focus recently. They are characteristically different from other measures because of their heavy reliance on the training data and use of Min-Features bias in selecting a subset of features [1]. Min-Features bias prefers consistent hypotheses definable over features as few as possible. This measure is similar to the consistency measure U we described in the beginning of this section with the difference that U can handle noise (e.g. misclassification). **Error Rate Measures** The methods using this type of evaluation function are called "wrapper methods", *i.e.*, the classifier is the evaluation function. As the features are selected using the classifier that later uses these selected features in predicting the class labels of unseen instances, the accuracy level is very high although computationally rather costly [9].

2.3 Consistency Measure vis-a-vis other measures

We compare consistency measure with other measures. First, consistency measure is monotonic and others are not. Assuming we have subsets $\{S_0, S_1, ..., S_n\}$ of features, we have a measure U that evaluates each subset S_i. The monotonicity condition requires the following: $S_0 \supset S_1 \supset ... \supset S_n \Rightarrow U(S_0) \leq U(S_1) \leq ... \leq U(S_n)$. Second, for the consistency measure, a feature subset can be evaluated in $O(P)$. It is usually costlier for other measures. For example, to construct a decision tree in order to have predictive accuracy, it requires at least $O(P \log P)$; to calculate the distances, it requires $O(P^2)$. Third, consistency measure can help remove both redundant and irrelevant features; other measures may not do so. Last, consistency measure is capable of handling some noise in the data reflected as a percentage of inconsistencies. This percentage can be obtained by going through the data once. In short, consistency measure is monotonic, fast, able to remove redundant and/or irrelevant features, and capable of handling some noise[1]

3 Ways of Using Consistency Measure

Different search strategies pose further constraints on a selection criterion. We demonstrate that the consistency measure can be employed in common forms of search without modification. Five different algorithms represent standard search strategies: *exhaustive* - Focus [1], *complete* - ABB [13], *heuristic* - SetCover [6], *probabilistic* - LVF [14], and *hybrid* of ABB and LVF - QBB. We examine their advantages and disadvantages.

Focus: exhaustive search: Focus [1] starts with an empty set and carries out breadth-first search until it finds a minimal subset that predicts pure classes. With some modification of Focus, we have FocusM that can work on non-binary data with noise. As FocusM is exhaustive search it guarantees an optimal solution. However, FocusM's time performance can deteriorate if M is not small with respect to N. The search space of FocusM is closely related to the number of relevant features. In general, the less the number of relevant features, the smaller the search space.

ABB: complete search: Branch & Bound (B&B) [16] starts with a full set

[1] There are many types of noise. Consistency measure can handle misclassifications.

of features, and removes one feature at a time. When there is no restriction on expanding nodes in the search space, this could lead to an 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 B&B backtracks whenever an infeasible node is discovered. If U is monotonic, no feasible node is omitted and savings of search time do not sacrifice optimality. As pointed out in [19], the measures used in [16] such as accuracy have disadvantages (e.g., non-monotonicity); the authors of [19] proposed the concept of approximate monotonicity. ABB [13] is an automated B&B algorithm having its bound as the inconsistency rate of the data when the full set of features is used. 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 (its branch is pruned); otherwise, it grows to level $l+1$, *i.e.* one more feature could be removed.

Since inconsistency is a monotonic measure, ABB guarantees an optimal solution. However, a brief analysis suggests that ABB's time performance can deteriorate as the difference $N - M$ increases. This issue is related to how many nodes (subsets) have been generated. The search space of ABB is closely related to the number of relevant features. In general, the more the number of relevant features, the smaller the search space due to early pruning of the illegitimate nodes. Our analysis of Focus and ABB reveals that Focus is efficient when M is small, and ABB is efficient when $N - M$ is small. In other cases, we can use inconsistency measure in heuristic search.

SetCover: heuristic search: SetCover [6] uses the observation that the problem of finding the smallest set of consistent features is equivalent to 'covering' each pair of examples that have different class labels with some feature on which they have different values. This enables us to apply Johnson's algorithm [10] for Set Cover for this problem, which implies that the resulting algorithm outputs a consistent feature set of size $O(M \log P)$ in polynomial time. Variants of Set Cover have previously been used for learning conjunctions of boolean features. Consistency criterion can be restated as: a feature set S is consistent if for any pair of instances with different class labels, there is a feature in S that takes different values. Thus including a feature f in S "takes care of" all those example pairs with different class labels on which f takes different values. Once all pairs are "taken care of" the resulting set S is consistent. The advantages of SetCover is that it is fast, close to optimal, and deterministic. This works well for data where features are rather independent of each other. It may, however, have problems where features have inter-dependencies. This is because it selects the best feature in each iteration based on the number of instance-pairs covered. A new solution is needed that avoids the problems of exhaustive and heuristic search. Probabilistic search is a natural choice.

LVF: probabilistic search: Las Vegas algorithms [5] for feature subset selection can make probabilistic choices of subsets in search of an optimal set. Another similar type of algorithms is the Monte Carlo algorithm in which it is often possible to reduce the error probability arbitrarily at the cost of a slight increase in computing time [5]. LVF is more suitable since the probability of generating a certain subset is the same. LVF adopts the inconsistency rate as the evaluation measure. Due to its monotonicity, a superset of a subset of relevant features is also good. Hence, there are more chances for good subsets to be selected. LVF keeps the smallest subset of features randomly generated so far that satisfies a threshold (by default it is the inconsistency rate of the data with all features). It is fast in reducing the number of features. We conducted experiments to observe how the number of valid features (M') drops as the number of randomly generated feature sets increases. A total of 10 data, both artificial and real, are chosen for the experiments from the UC Irvine data repository [15] (Table 1). Two typical graphs are shown in Figure 1 in a longer time span (partial results shown in Table 1) in order to observe the trend.

Data	LED24	Lung	Lymph	Mush	Par3+3	Promo	Soy	Splice	Vote	Zoo
P	1200	32	148	7125	64	106	47	3190	435	74
m	10	3	4	2	2	2	4	3	2	7
N	24	56	18	22	12	57	35	60	16	16
$M'(M)$	12(5)	19(4)	8(6)	8(4)	5(3)	15(4)	12(2)	19(9)	13(8)	9(5)
#Eval	230	155	215	22	25	187	42	284	215	25
#Max	2^{24}	2^{56}	2^{18}	2^{22}	2^{12}	2^{57}	2^{35}	2^{60}	2^{16}	2^{16}

Table 1. The number of valid features (M') drops sharply in the first few hundred runs for all data. P, N, M and m are defined earlier. #Eval is number of subsets generated and evaluated. #Max is maximum possible subsets.

The trend found in all the experiments is that M' drops sharply from N in the first few hundred runs (one run means one feature set is randomly generated and evaluated). Afterwards, it takes quite a long time to further decrease M'. Some analysis can confirm this finding. A particular set has a probability of $1/2^N$ to be generated. At the beginning, the full set is valid. Many subsets can satisfy the inconsistency criterion. As M' decreases from N to M, fewer and fewer subsets can satisfy the criterion. However, the probability of a distinct set being generated is still $1/2^N$. That explains why the curves have a sharp drop in the beginning and then become flat in Figure 1. LVF reduces the number of features quickly during the initial stage (the first few hundred loops); after that LVF still searches in the same way (i.e., blindly), the computing resource is spent on generating many subsets that are obviously not good.

QBB: hybrid search: As ABB and LVF complement each other, QBB is a natural offspring of ABB and LVF, which uses inconsistency as its evaluation measure. QBB runs LVF in the first phase and ABB in the second phase so that the search is more focused after the sharp decrease in the number of valid subsets. A key issue remains: what is the crossing point in QBB at which ABB takes over from LVF. If we allow only certain amount of time to run QBB, the point at which ABB takes over from LVF is crucial for the efficiency of QBB.

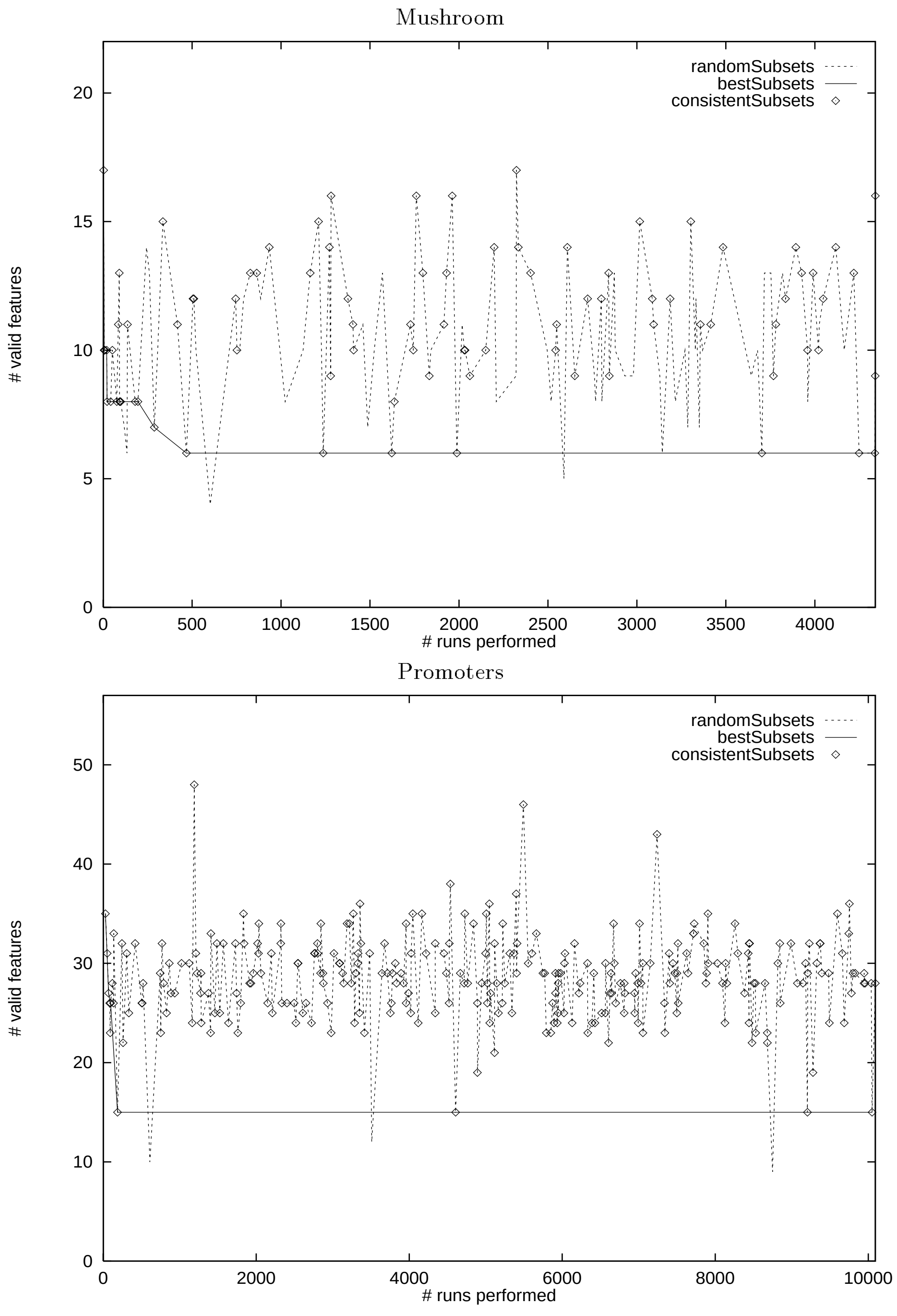

Fig. 1. The typical trends of the decreasing number of valid features versus the number of runs performed. Points include both valid and invalid feature subsets. Valid subsets are connected by solid lines.

Extensive experiments suggested that dividing the total time equally between LVF and ABB is a robust solution and is more likely to yield the best results. If the crossing point is too early, LVF might not have reduced the valid subset size substantially for ABB to perform well under time constraint; but if the crossing point is too late, the small sized subsets generated by LVF at the crossing point might not contain any minimal size subset, and so ABB becomes ineffective.

3.1 Summary: when to use what

As we have five algorithms to choose from, we are also interested to know how we should use them. Theoretical analysis and experimental experience suggest the following. If M - the size of relevant features is small, FocusM should be chosen; however if M is even moderately large, FocusM will take a long time. If there are a small number of irrelevant and redundant features, ABB should be chosen; but ABB will take a long time for a moderate number of irrelevant features. For data with large numbers of features, FocusM and ABB should not be expected to terminate in realistic time. For the Letter data with 20,000 instances ($N = 16$ and $M = 11$) FocusM took more than 2 days to terminate whereas ABB took more than 7 hours to generate optimal subsets. Hence, in such cases one should resort to heuristic or probabilistic search for faster results. Although these algorithms may not guarantee optimal subsets but will be efficient in generating near optimal subsets in much less time. SetCover is heuristic, fast, and deterministic. It may face problems with data having highly interdependent features. LVF is probabilistic, not prone to the problem faced by SetCover, but slow to converge in later stages. As we have shown, it can reduce the feature subset size very fast in the beginning but then it slows down in reducing features. QBB is a welcome modification as it captures the best of LVF and ABB. It is reasonably fast (slower than SetCover), robust, and can handle features with high interdependency.

4 Further Experiments

The points that remain inconclusive are: (1) features selected using inconsistency can achieve the objective of dimensionality reduction without sacrificing predictive accuracy; and (2) how the different algorithms fare in terms of time and optimality. The experimental procedure is to (1) choose data frequently used by the community; (2) run ABB to get the minimal size as reference; (3) compare the performance (average time and number of selected features) of different algorithms; and (4) compare the accuracy of two different classifiers (C4.5 [17] and Back-propagation neural network [21]) over data before and after feature selection by QBB. Ten data, both artificial and real, are chosen for the experiments from the UC Irvine data repository [15]. A summary of these data is given in Table 1. Par3+3 contains 12 features (3 relevant, 3 redundant, 6 irrelevant).

Figure 2 shows a comparison of the performance (both average time and number of selected features) of different algorithms. First ABB is run over the 10 data to find the M (minimal size) values. For comparison purpose we have

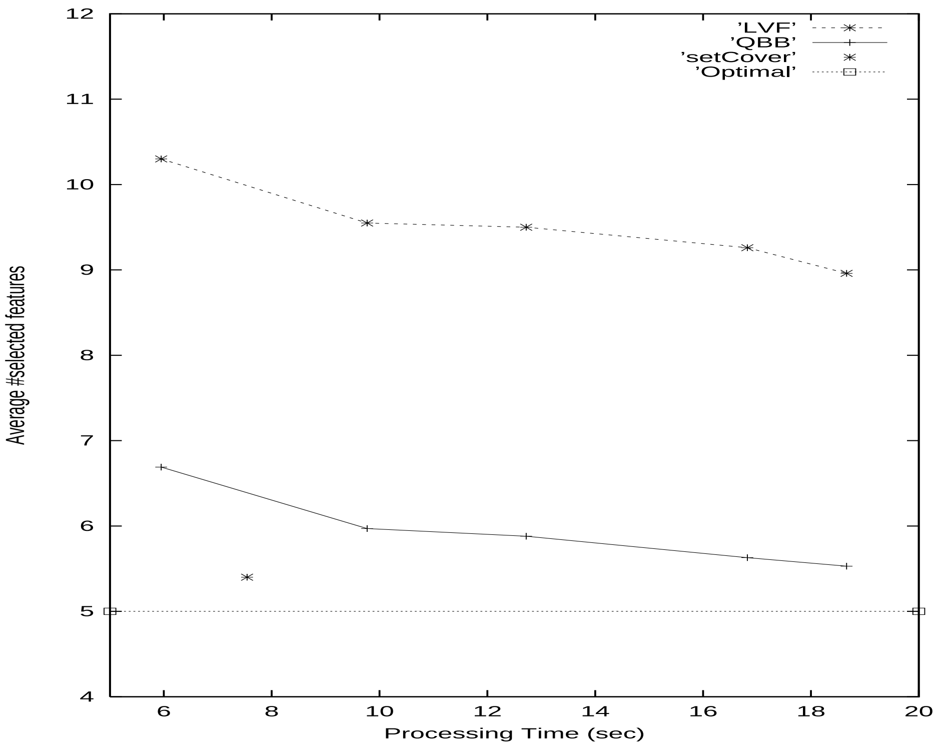

Fig. 2. Experiments to show how differently algorithms fare in terms of time and optimality. Results of *Focus* and *ABB* are out of bounds in x-axis (time).

calculated the average minimal value, M_{Avg}, over all data which is found to be 5. This value is used as a reference line in Figure 2. Out of the 5 competing algorithms, FocusM, ABB and SetCover are deterministic, whereas LVF and QBB are non-deterministic due to their probabilistic nature. QBB spends half of the time running LVF and the other half running ABB. For LVF and QBB we show results for 5 different processing time in terms of total numbers of subsets evaluated (1000...5000). Each experiment was repeated 50 times. Notice that Focus and ABB are not shown in the graph as their average times fall outside the range of the 'processing time' in the x-axis of the graph, although minimal sized subsets are guaranteed in each case. For data having large differences between N and M values such as Lung Cancer, Promoters, Soybean, Splice data ABB takes very long time (a number of hours) to terminate. For data having large N values and not very small M values such as Splice data ($N = 60$, $M = 9$) FocusM takes many hours to terminate. The comparison in Figure 2 shows that QBB is more efficient both in average time and number of selected features compared to LVF, FocusM, and ABB. The average size of the subsets produced by QBB is close to M_{Avg} and it approaches to M_{Avg} with time. SetCover produces near optimal subsets in much less time. Between QBB and SetCover we would say QBB is more robust while SetCover, although very fast and accurate, may fail to deliver efficient subsets if there is dependency among the features.

The error probability is often used as a validation criterion. Among the different algorithms discussed in the paper we take QBB due to its robustness. We choose C4.5 decision tree and Back-propagation neural network as two classifiers for validation. For back-propagation each data was divided into a training set (two-third of the original size) and the rest one-third as testing. For C4.5, we use the default settings, apply it to data before and after feature selection, and obtain the results of 10-fold cross-validation. This is repeated 10 times for each data and the average error rate and tree size are reported in Table 2. That is, QBB has been run 10 times and C4.5 100 times. The experiment shows the improvement/no reduction for most data (9 out of 10) in C4.5's accuracy after feature selection.

Running Back-propagation 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 QBB, we try to minimize the tuning of the parameters for each data. We fix the learning rate at 0.1, the momentum at 0.5, one hidden layer, the number of hidden units as half of the original input units for all data. The experiment is carried out in two steps: (1) a trial run to find a proper number of CYCLES for each data which is determined by a sustained trend of no decrease of error; and (2) two runs on data with and without feature selection respectively using the number of CYCLES found in step 1. Other parameters remain fixed for the two runs in step 2. The results are shown in Table 2 with an emphasis on the difference before and after feature selection. In most cases, error rates decrease (6 out of 10) or do not change (3 out of 10) after feature selection.

	C4.5				Back-Propagation			
	Tree Size		Error Rate				Error Rate	
Data	Bef	Aft	Bef	Aft	Cycles	#HU	Bef	Aft
LED-24	19.0	19.0	0.0	0.0	1000	12	0.06	0.0
Lung	19.0	10.9	50.0	41.8	1000	28	75.0	75.0
Lymphography	26.9	22.1	21.8	21.4	7000	9	25.0	25.0
Mushroom	36.3	34.2	0.0	0.0	5000	11	0.0	0.0
Par3+3	12.0	15.0	41.9	0.0	1000	6	22.2	0.0
Promoters	21.4	8.2	26.3	22.1	2000	29	46.8	25.0
Soybean	7.1	9.4	2.5	2.0	1000	18	10.0	0.0
Splice	173.0	187.0	5.9	14.0	6000	30	25.64	42.33
Vote	14.5	14.2	5.3	5.3	4000	8	6.7	4.0
Zoo	17.8	17.7	7.8	6.6	4000	8	10.3	3.4

Table 2. Results of Hybrid Search. #HU is number of Hidden Units.

5 Concluding Remarks

The fact that the consistency criterion does not incorporate any search bias relating to a particular classifier enables it to be used with a variety of different learning algorithms. As shown in the experiments, for the two different types of classifiers, selected features improve the performance in terms of lower error rates in most cases. Features selected without search bias bring us efficiency in later stage as the evaluation of a feature subset becomes simpler than that of a full set. On the other hand, since a set of features is deemed consistent if *any* function maps from the values of the features to the class labels, any algorithm optimizing this criterion may choose a small set of features that has a complicated function, while overlooking larger sets of features admitting simple rules. Although intuitively this should be relatively rare, it can happen in practice, as apparently this was the case for the Splice data where both C4.5 and Back-propagation's performance deteriorate after feature selection.

The inconsistency measure has received a comprehensive examination that reveals its many merits for feature selection. The outstanding one is its monotonicity. It is also fast to compute, can detect redundant as well as irrelevant features. It has been used with a variety of search strategies in feature selection and no modification is required. The salient contribution of this work is that a guideline is suggested as to when to use what after detailed evaluation of different search algorithms. We believe the guideline will be very helpful to practitioners in need to reduce dimensionality of huge data, and to researchers who want to further the work of feature selection.

References

1. H. Almuallim and T. G. Dietterich. Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305, November 1994.

2. M. Ben-Bassat. Pattern recognition and reduction of dimensionality. In P. R. Krishnaiah and L. N. Kanal, editors, *Handbook of Statistics*, pages 773–791. North Holland, 1982.
3. A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271, 1997.
4. A Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Readings in Machine Learning*, pages 201–204, 1990.
5. G. Brassard and P. Bratley. *Fundamentals of Algorithms*. Prentice Hall, New Jersy, 1996.
6. M. Dash. Feature selection via set cover. In *Proceedings of* IEEE *Knowledge and Data Engineering Exchange Eorkshop*, pages 165–171, Newport, California, November 1997. IEEE Computer Society.
7. M. Dash and H. Liu. Feature selection methods for classification. *Intelligent Data Analysis: An Interbational Journal*, 1(3), 1997.
8. P. A. Devijver and J. Kittler. *Pattern Recognition : A Statistical Approach*. Prentice Hall, 1982.
9. G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 121–129, 1994.
10. D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.
11. K. Kira and L. A. Rendell. The feature selection problem : Traditional methods and a new algorithm. In *Proceedings of Ninth National Conference on AI*, pages 129–134, 1992.
12. R. Kohavi. *Wrappers for performance enhancement and oblivious decision graphs*. PhD thesis, Department of Computer Science, Stanford University, CA, 1995.
13. H. Liu, H. Motoda, and M. Dash. A monotonic measure for optimal feature selection. In *Proceedings of European Conference on Machine Learning*, pages 101–106, 1998.
14. H. Liu and R. Setiono. Feature selection and classification - a probabilistic wrapper approach. In *Proceedings of Ninth International Conference on Industrial and Engineering Applications of AI and ES*, 1996.
15. C. J. Merz and P. M. Murphy. UCI repository of machine learning databases, 1996. FTP from ics.uci.edu in the directory pub/machine-learning-databases.
16. P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature selection. *IEEE Transactions on Computers*, C-26(9):917–922, September 1977.
17. J. R. Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.
18. T. W. Rauber. *Inductive Pattern Classification Methods - Features - Sensors*. PhD thesis, Department of Electrical Engineering, Universidale Nova de Lisboa, 1994.
19. W. Siedlecki and J Sklansky. On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2:197–220, 1988.
20. S. Watanabe. *Pattern Recognition: Human and Mechanical*. Wiley Intersceince, 1985.
21. A. Zell and et al. Stuttgart Neural Network Simulator (SNNS), user manual, version 4.1. Technical report, 1995.