



Consistency-based search in feature selection

Manoranjan Dash^{a,*}, Huan Liu^b

^a *Electrical and Computer Engineering, Northwestern University,
2145 Sheridan Rd, Evanston, IL 60208-3118, USA*

^b *Department of Computer Science and Engineering, Arizona State University,
PO Box 875406, Tempe, AZ 85287-5406, USA*

Received 7 March 2002; received in revised form 27 March 2003

Abstract

Feature selection is an effective technique in dealing with dimensionality reduction. For classification, it is used to find an “optimal” subset of relevant features such that the overall accuracy of classification is increased while the data size is reduced and the comprehensibility is improved. Feature selection methods contain two important aspects: evaluation of a candidate feature subset and search through the feature space. Existing algorithms adopt various measures to evaluate the goodness of feature subsets. This work focuses on *inconsistency* measure according to which a feature subset is inconsistent if there exist at least two instances with same feature values but with different class labels. We compare inconsistency measure with other measures and study different search strategies such as exhaustive, complete, heuristic and random search, that can be applied to this measure. We conduct an empirical study to examine the pros and cons of these search methods, give some guidelines on choosing a search method, and compare the classifier error rates before and after feature selection.

© 2003 Elsevier B.V. All rights reserved.

Keywords: Classification; Feature selection; Evaluation measures; Search strategies; Random search; Branch and bound

1. Introduction

The basic problem of classification is to classify a given instance (or example) to one of m known classes. A set of features presumably contains enough information to distinguish among the classes. When a classification problem is defined by features, the number of

* Corresponding author.
E-mail address: manoranj@lilac.ece.nwu.edu (M. Dash).

features can be quite large, many of which can be irrelevant or redundant. A relevant feature is defined in [5] as one removal of which deteriorates the performance or accuracy of the classifier; an irrelevant or redundant feature is not relevant. Because irrelevant information is cached inside the totality of the features, these irrelevant features could deteriorate the performance of a classifier that uses all features [6]. The fundamental function of a feature selector is to extract the most useful information from the data, and reduce the dimensionality in such a way that the most significant aspects of the data are represented by the selected features [14]. Its motivation is three-fold: *simplifying* the classifier by retaining only the relevant features; *improving or not significantly reducing* the accuracy of the classifier; and *reducing* the dimensionality of the data thus reducing the size of the data. The last point is particularly relevant when a classifier is unable to handle large volumes of data. Research on feature selection has been done for last several decades and is still in focus. Reviews and books on feature selection can be found in [11,26,27]. Recent papers such as [2,9,16,17,22,43] address some of the existing issues of feature selection.

In order to select relevant features one needs to measure the goodness of selected features using a selection criterion. The class separability is often used as one of the basic selection criteria, i.e., when a set of features maximizes the class separability, it is considered well suited for classification [37]. From a statistics view point, five different measurements for class separability are analyzed in [14]: error probability, inter-class distance, probabilistic distance, probabilistic dependence and entropy. Information-theoretic considerations [45] 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 called *consistency measure* that does not attempt to maximize the class separability but tries to retain the discriminating power of the data defined by original features. Using this measure, feature selection is formalized as finding the smallest set of features that can distinguish classes as if with the full set. In other words, if S is a consistent set of features, no two instances with the same values on S have different class labels [1].

Another aspect of feature selection is related to the study of search strategies to which extensive research efforts have been devoted [5,11,41]. 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 shrinks the search space by deleting one feature at a time (Backward Selection)—an example is ‘Branch & Bound’ [34]. Except these two starting points, the search process can start from a random subset and continue either probabilistically (LVF [30]) or deterministically (QBB [12]) from there.

The contributions of this paper include:

- a detailed study of consistency measure vis-a-vis other evaluation measures;
- pros and cons of various search strategies (exhaustive, complete, heuristic, and probabilistic) based on consistency measure;
- experimental comparison of different search methods; and
- guidelines about choosing a search method.

In the rest of the paper, we describe the feature selection process in Section 2. Section 3 discusses consistency measure and compares with other measures. Section 4 describes different search strategies for consistency measure and their pros and cons. Section 5 shows some experimental results and Section 6 concludes the paper.

2. Feature selection process

In the rest of the paper we use the following notations. P is the total number of instances, N denotes the total number of features, M stands for the number of relevant/selected features, S denotes a subset of features, f_1, \dots, f_M are the M features, m denotes the number of different class labels, and C stands for the class variable.

Ideally feature selection methods search through the subsets of features and try to find the best subset among the competing 2^N candidate subsets according to some evaluation measure. But this procedure is exhaustive as it tries to find only the best one, and may be too costly and practically prohibitive even for a medium-sized N . Other methods based on heuristic or random search methods attempt to reduce computational complexity by compromising optimality. These methods need a stopping criterion to prevent an exhaustive search of subsets. There are four basic steps in a typical feature selection method (see Fig. 1):

- (1) a *generation procedure* to generate the next candidate subset for evaluation,
- (2) an *evaluation function* to evaluate the candidate subset,
- (3) a *stopping criterion* to decide when to stop, and
- (4) a *validation procedure* to check whether the subset is valid.

The *generation procedure* uses a search strategy to generate subsets of features for evaluation. It starts (i) with no features, (ii) with all features, or (iii) with a random subset of features. In the first two cases features are iteratively added/removed, whereas in the last

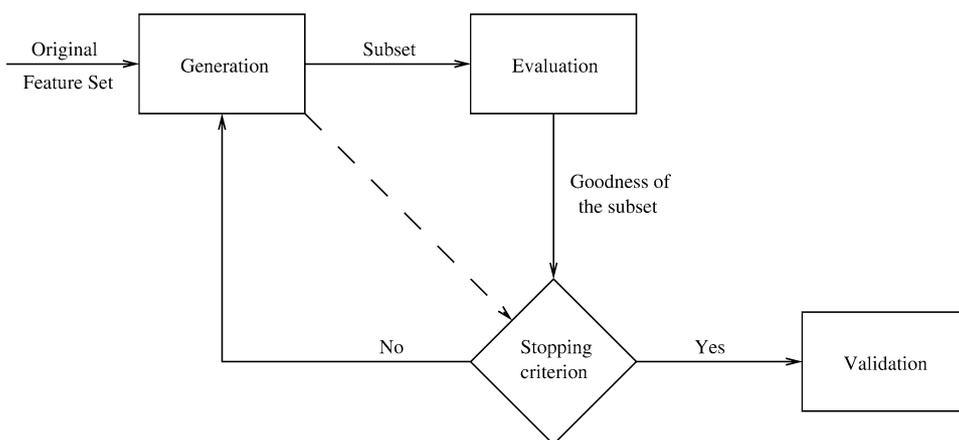


Fig. 1. Feature selection process with validation.

case, features are either iteratively added/removed or produced randomly thereafter during search.

An *evaluation function* measures the goodness of a subset produced by some generation procedure, and this value is compared with the previous best. If it is found to be better, then it replaces the previous best subset. An optimal subset is always relative to a certain evaluation function (i.e., an optimal subset chosen using one evaluation function may not be the same as that using another evaluation function).

Without a suitable *stopping criterion* the feature selection process may run unnecessarily long or possibly forever depending on search strategy. Generation procedures and evaluation functions can influence the choice for a stopping criterion. Examples of stopping criteria based on a generation procedure include: (i) whether a predefined number of features are selected, and (ii) whether a predefined number of iterations reached. Examples of stopping criteria based on an evaluation function include: (i) whether further addition (or deletion) of any feature produces a better subset, and (ii) whether an optimal subset (according to some evaluation function) is obtained. The feature selection process halts by outputting the selected subset of features which is then validated.

There are many variations to this feature selection process but the basic steps of generation, evaluation and stopping criterion are present in almost all methods.

The validation procedure is not a part of the feature selection process itself. It tries to test the validity of the selected subset by testing and comparing the results with previously established results or with the results of competing feature selection methods using artificial datasets, and/or real-world datasets.

3. Consistency measure

3.1. The measure

The suggested measure U is an *inconsistency rate* over the dataset for a given feature set. In the following description a *pattern* is a part of an instance without class label. It is a set of values of feature subset. For a feature subset S with $n_{f_1}, n_{f_2}, \dots, n_{f_{|S|}}$ number of values for features $f_1, f_2, \dots, f_{|S|}$ respectively, there are at most $n_{f_1} * n_{f_2} * \dots * n_{f_{|S|}}$ patterns.

Definition. Consistency measure is defined by inconsistency rate which is calculated as follows.

- (1) A pattern is considered *inconsistent* if there exists at least two instances such that they match all but their class labels; for example, an inconsistency is caused by instances $(0, 1, 1)$ and $(0, 1, 0)$ where the two features take the same values in the two instances while the class attribute varies which is the last value in the instance.
- (2) The *inconsistency count* for a pattern of a feature subset is the number of times it appears in the data minus the largest number among different class labels. For example, let us assume for a feature subset S a pattern p appears in n_p instances out of which c_1 instances has class label 1 , c_2 has label 2 , and c_3 has label 3 where $c_1 + c_2 + c_3 = n_p$.

If c_3 is the largest among the three, the inconsistency count is $(n - c_3)$. Notice that the sum of all n_p s over different patterns p that appear in the data of the feature subset S is the total number of instances (P) in the dataset, i.e., $\sum_p n_p = P$.

- (3) The *inconsistency rate* of a feature subset S ($I_R(S)$) is the sum of all the inconsistency counts over all patterns of the feature subset that appears in the data divided by P .

The consistency measure is applied to the feature selection task as follows. Given a candidate feature subset S we calculate its inconsistency rate $I_R(S)$. If $I_R(S) \leq \delta$ where δ is a user given inconsistency rate threshold, the subset S is said to be *consistent*. Notice that this definition is an extension of the earlier definition used in [1]: the new definition tolerates a given threshold error rate. This definition suits the characteristic of consistency measure because real-world data is usually noisy and if δ is set to 0% then it may so happen that no feature subset can satisfy the stringent condition.

By employing a hashing mechanism, we can compute the inconsistency rate approximately with a time complexity of $O(P)$ [30]. By definition, consistency measure can work when data has discrete valued features. Any continuous feature should be first discretized using some discretization method available in the literature [25].

In the rest of the paper we use consistency measure and inconsistency rate interchangeably.

3.2. Other evaluation measures

In the following we briefly introduce different evaluation measures found in the literature. 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 [5] grouped different feature selection methods into two broad groups (i.e., filter and wrapper) based on their dependence on an inductive algorithm (classifier) that will finally use the selected subset. By their definition, *filter* methods are independent of an inductive algorithm, whereas *wrapper* methods use an inductive algorithm as the evaluation function. Ben-Bassat [3] grouped the evaluation functions till 1982 into three categories: information or uncertainty measures, distance measures, and dependence measures. He did not consider the classification error rate as an evaluation function. Considering these divisions and latest developments, we divide the evaluation functions into five categories: *distance*, *information (or uncertainty)*, *dependence*, *consistency*, and *classifier error rate*. In the following, we briefly discuss each of them (see [11] for more details).

- (1) *Distance measures*. It is also known as separability, divergence, or discrimination measure. For a two class problem, a feature f_i is preferred to another feature f_j if f_i induces a greater difference between the two-class conditional probabilities than f_j ; if the difference is zero then f_i and f_j are indistinguishable. Distance measure is employed in [20,24,34,39].
- (2) *Information measures*. These measures typically determine the information gain from a feature. The information gain from a feature f_i is defined as the difference between the prior uncertainty and expected posterior uncertainty using f_i . Feature f_i is preferred to feature f_j if the information gain from feature f_i is greater than that from

feature f_j [3]. An example of this type is entropy. Information measure is employed in [2,8,23,40].

- (3) *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 variable. If the correlation of feature f_i with class variable C is higher than the correlation of feature f_j with C , then feature f_i is preferred to f_j . 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 classified as distance and information measures. But, these are still kept as a separate category because, conceptually, they represent a different viewpoint [3]. Dependence measure is employed in [31,33].
- (4) *Consistency measures*. This type of evaluation measures are characteristically different from other measures because of their heavy reliance on the training dataset and use of Min-Features bias in selecting a subset of features [1]. Min-Features bias prefers consistent hypotheses definable over as few features as possible. These measures find out the minimal size subset that satisfies the acceptable inconsistency rate, that is usually set by the user. Consistency measure is employed in [1,30,38].
The above types of evaluation measures are known as “filter” methods because of their independence from any particular classifier that may use the selected features output by the feature selection method.
- (5) *Classifier error rate measures*. In contrast to the above filter methods, classifier error rate measures are called “wrapper methods”, i.e., a classifier is used for evaluating feature subsets [22]. 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 computational cost is rather high compared to other measures. Classifier error rate measure is employed in [14,15,18,29,32,42,44].

3.3. Consistency measure vis-a-vis other measures

Consistency measure has the following differences with other types of methods [13].

- Consistency measure is monotonic while most others are not. Assuming we have subsets $\{S_0, S_1, \dots, S_n\}$ of features, we have a measure U that evaluates each subset S_i . The monotonicity condition requires the following:

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

Theorem 1. *Consistency measure is monotonic.*

Proof. A proof outline is given to show that the 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 . It is known that the discriminating power is inversely 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 redundant, and (iii) features in S_k relevant. 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 $S_i \subset S_j$. It is clear that the above results remain true for cases that S_k contains irrelevant, redundant as well as relevant features. \square

- For the consistency measure, a feature subset can be evaluated in $O(P)$ time. It is usually costlier for other measures. For example, to construct a decision tree in order to have predictive accuracy, it requires at least $O(P \log P)$.
- Consistency measure can help remove both redundant and irrelevant features; other measures may not do so. For example, Relief [20] fails to detect redundant features.
- Consistency measure is capable of handling some noise in the data reflected as a percentage of inconsistencies.
- Unlike the commonly used univariate measures (e.g., distance, information, and dependence measures), this is a multivariate measure which checks a subset of features at a time.

In summary, the consistency measure is *monotonic, fast, multivariate, able to remove redundant and/or irrelevant features, and capable of handling some noise*. As with other filter measures, it's not clear that it also optimizes the accuracy of a classifier trained on the data after feature selection. In the next section we describe different search strategies for consistency measure.

4. Different search strategies

Search techniques are important as exhaustive search of the “optimal” subset is impractical for even moderate N . In this section we will study and compare different search strategies for consistency measure.

Five different algorithms represent standard search strategies: *exhaustive*—Focus [1], *complete*—ABB [28], *heuristic*—SetCover [10], *probabilistic*—LVF [30], and *hybrid* of complete and probabilistic search methods—QBB [12]. In the rest of this section we examine their advantages and disadvantages, and at the end we give some guidelines about which search strategy to be used under different situations.

4.1. Focus: Exhaustive search

Focus [1] 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) where a ‘0’ means absence of the respective feature and ‘1’ means

its presence in the feature subset. It is exhaustive search in nature and originally works on binary, and noise-free data. With some simple modification of Focus, we have FocusM that can work on non-binary data with noise by applying the inconsistency rate (defined earlier) in place of the original consistency measure.

Below we give the FocusM algorithm where $\text{inConCal}()$ function calculates inconsistency rate of a given feature set S for a given dataset D .

Algorithm. FocusM

Input: Data D , feature set S

Output: Consistent Feature Subset

1. $\delta = \text{inConCal}(S, D)$
2. For $size = 0$ to $|S|$
3. for all subsets S' with $|S'| = size$
4. if $\text{inConCal}(S', D) \leq \delta$
5. return S'

As FocusM is exhaustive search it guarantees an optimal solution. However, a brief analysis can tell that FocusM's time performance can deteriorate fast with increasing M . This issue is directly related to the size of the search space. The search space of FocusM is closely related to the number of relevant features. For instance, taking a simple example of four features, if one out of four features is relevant, the search space is $\binom{4}{1} = 4$; and if all 4 features together satisfy consistency, the search space is $\sum_{i=1}^4 \binom{4}{i} = 41$. In general, the smaller the number of relevant features M , the smaller the search space of FocusM and higher its efficiency. But for data with small $N - M$ FocusM is inefficient, and one requires more efficient techniques. The following is such a technique.

4.2. ABB: Complete search

Branch & Bound for feature selection was first proposed in [34]. In contrast to Focus, it starts with a full set of features, and removes one feature at a time. If the full set contains three features, the root is (1 1 1) where '1' means presence of the corresponding feature and '0' its absence. Its child nodes are (1 1 0), (1 0 1), and (0 1 1), etc. 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 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, savings in search time do not sacrifice the optimality of the selected subset, and hence it is a non-exhaustive yet complete search. As was pointed out in [41], the measures used in [34] such as accuracy of classification have disadvantages (e.g., non-monotonicity). As a remedy, the authors proposed the concept of approximate monotonicity which means that branch and bound can continue even after encountering a node that does not satisfy the bound. But the node that finally is accepted must satisfy the set bound. In ABB (Automatic Branch and Bound) [28], we proposed an automated Branch & Bound algorithm having its bound set to the inconsistency rate of the original feature set. The algorithm is given below.

Algorithm. ABB**Input:** Data D , feature set S **Output:** Consistent Feature Subset

1. $\delta = \text{inConCal}(S, D)$
2. $T = S$
/* subset generation */
3. For all feature f in S
4. $S_j = S - f$ /* remove one feature at a time */
5. $j++$
6. For all S_j
7. if (S_j is legitimate \wedge $\text{inConCal}(S_j, D) \leq \delta$)
8. if $\text{inConCal}(S_j, D) < \text{inConCal}(T, D)$
9. $T = S_j$
/* recursion */
10. $\text{ABB}(S_j, D)$
11. return T

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 l th 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.

The legitimacy test is based on whether a node (subset) is a child node of a pruned node. A node is illegitimate if it is a child node of a pruned one (which is already found to be illegitimate). Each node is represented by a binary vector where 1's stand for presence of a particular feature in that subset and 0's for its absence. The test is done by checking the Hamming distance between the child node under consideration and pruned nodes. If the Hamming distance with any pruned node is 1 (i.e., the difference of the two representative binary vectors is 1), the child node is the child of the pruned node. Notice that by this way at every level we are able to determine all the illegitimate nodes.

Example. Refer to Fig. 2 there are four features of which only the first two (underlined) are relevant. The root $S_0 = (1 \ 1 \ 1 \ 1)$ of the search tree is a binary array with four '1's. Following ABB, we expand the root to four child nodes by turning one of the four '1's into '0' (L2). All four are legitimate: $S_1 = (1 \ 1 \ 1 \ 0)$, $S_2 = (1 \ 1 \ 0 \ 1)$, $S_3 = (1 \ 0 \ 1 \ 1)$, and $S_4 = (0 \ 1 \ 1 \ 1)$. Since one of the relevant features is missing, $U(S_3)$ and $U(S_4)$ will be greater than $U(S_0)$ where U is the inconsistency rate over the given data. Hence, the branches rooted by S_3 and S_4 are pruned and will not grow further. Only when a new node passes the legitimacy test will its inconsistency rate be calculated. Doing so improves the efficiency of ABB because P (number of patterns) is normally much larger than N (number of features). The rest of the nodes are generated and tested in the same way.

Since inconsistency is a monotonic measure, ABB guarantees an optimal solution. However, a brief analysis suggests that time performance of ABB 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.

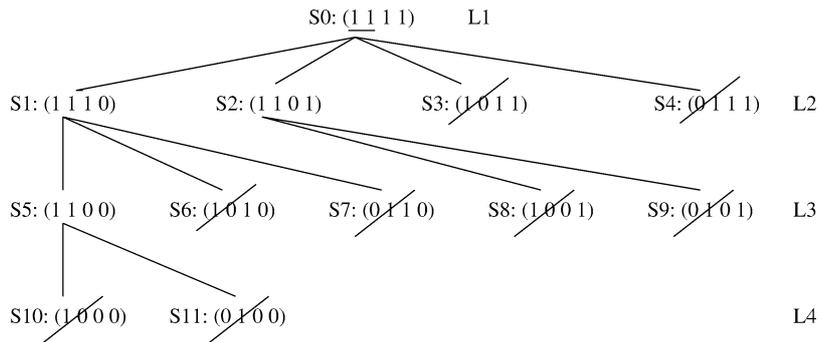


Fig. 2.

For instance, taking a simple example of four features, if one out of four features is relevant, the search space is 15 nodes which is almost exhaustive (the total is $\sum_{i=0}^4 \binom{4}{i} = 16$ nodes). And when all four features are relevant, it is 5 nodes, i.e., the root plus 4 child nodes. In general, the more the number of relevant features, the smaller the search space due to early pruning of the illegitimate nodes. See [28] for detailed results (the number of subsets evaluated, the number of subsets pruned, etc.) of ABB over a number of benchmark datasets.

The analysis of Focus and ABB shows the following:

- Focus is efficient when M is small, and
- ABB is efficient when $N - M$ is small.

In other cases, the two algorithms can be inefficient. An immediate thought is whether we can use the inconsistency measure in heuristic search and how it fares.

4.3. SetCover: Heuristic search

SetCover [10] exploits the observation that the problem of finding a smallest set of consistent features is equivalent to ‘covering’ each pair of examples that have different class labels. Two instances with different class labels are said to be ‘covered’ when there exists at least one feature which takes different values for the two instances [35]. This enables us to apply Johnson’s algorithm [19] for *set-cover* to this problem. Johnson’s algorithm shows that the size of the resulting consistent feature subset is $O(M \log P)$. However, experimental results show that these estimations are in fact much larger than the obtained results, i.e., the size of the consistent subsets in the experiments is much less than $O(M \log P)$.

The algorithm works as follows: The consistency criterion can be restated by saying that 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 ‘covers’ all those example pairs with different class labels on which f takes different values. Once all pairs are ‘covered’ is the resulting set S consistent.

Algorithm. SetCover**Input:** Data D , feature set S **Output:** Consistent Feature Subset

1. $\delta = \text{inConCal}(S, D)$
2. $S' = \phi$
3. repeat
4. $\text{lowestInCon} = C$ C is a large constant
5. for all features $f \in S$
6. $S'' = \text{append}(S', f)$
7. $\text{tempInCon} = \text{inConCal}(S'', D)$
8. if $\text{tempInCon} < \delta$
9. return S''
10. else
11. if $\text{tempInCon} < \text{lowestInCon}$
12. $\text{lowestInCon} = \text{tempInCon}$
13. $\text{lowestFeature} = f$
14. $S' = \text{append}(S', f)$
15. $S = S - f$
16. until $\text{lowestInCon} \leq \delta$

In [10] we report extensive experimental results which show that SetCover is fast, close to optimal, and deterministic.¹ It works well for datasets where features are rather independent of each other. It may, however, have problems where features are correlated. This is because it selects the best feature in each iteration based on the number of instance-pairs covered. So, any feature that is most correlated to the class label is selected first. An example is the CorrAL dataset [21] which has 64 instances with six binary features and two classes. Feature f_5 is irrelevant to the target concept which is $(f_1 \wedge f_2) \vee (f_3 \wedge f_4)$. Feature f_6 is *correlated* to the target concept 75% of the time. SetCover first selects the feature f_6 due to the fact that among all six features f_6 covers the maximum number of instances (75%). Then it selects features f_1, f_2, f_3 , and f_4 ; so, it selects the wrong subset $(f_6, f_1, f_2, f_3, f_4)$ overall.

So, we found that exhaustive methods have inherent drawback because they require large computational time. Heuristic methods such as SetCover, although very fast and accurate, can encounter problems if the data has highly correlated features. Hence, a new solution is needed that avoids the problems of exhaustive and heuristic search. Probabilistic search is a natural choice.

¹ In the literature for search techniques, there are many methods such as genetic algorithms, simulated annealing, tabu search and estimation of distribution algorithm. Our choice of SetCover is mainly guided by the requirement that a heuristic algorithm must be fast to complete and should return good results. So, although the above mentioned search techniques are known to return good subsets, but they are not chosen due to their known higher computational time requirement.

4.4. LVF: Probabilistic search

Las Vegas algorithms [7] for feature subset selection can make probabilistic choices of subsets in search of an optimal set. Another similar type of algorithm is the Monte Carlo algorithm in which it is often possible to reduce the error probability arbitrarily at the cost of a little increase in computing time [7]. In [29] we proposed a probabilistic algorithm called **Las Vegas Filter (LVF)** where probabilities of generating any subset are equal. LVF adopts the inconsistency rate as the evaluation measure. It generates feature subsets randomly with equal probability, and once a consistent feature subset is obtained that satisfies the threshold inconsistency rate (δ which by default is set to the inconsistency rate of the original feature set), the size of generated subsets is pegged to the size of that subset, i.e., subsets of higher size are not evaluated anymore. This is based on the fact that inconsistency rate is monotonic, i.e., a superset of a consistent feature set is also consistent. This guarantees a continuously diminutive consistent feature subsets as output of LVF. LVF is fast in reducing the number of features in the early stages and can produce optimal solutions if computing resources permit. In the following the LVF algorithm is given.

Algorithm. LVF

Input: Data D , feature set S , $MaxTries$

Output: Consistent Feature Subset

1. $\delta = \text{inConCal}(S, D)$
2. $T = S$
3. for $j = 1$ to $MaxTries$
4. randomly choose a subset of features, S_j
5. if $|S_j| \leq |T|$
6. if $\text{inConCal}(S_j, D) \leq \delta$
7. if $|S_j| < |T|$
8. $T = S_j$
9. output S_j
10. else
11. append S_j to T
12. output S_j as ‘yet another solution’
13. return T

In the algorithm ‘yet another solution’ means a solution of the same size as that of the most recently found solution. By appending solutions of equal size the algorithm produces a list of equal-sized feature subsets at the end. A more sophisticated version of LVF is sampling without replacement. The number of features in each sampling is determined by the size of the current smallest subset. This could speed up subset generation.

LVF performance. We conducted experiments to observe how the consistent feature subset size (M') drops as the number of randomly generated feature sets increases. A total of 10 benchmark datasets are taken from the UC Irvine machine learning repository [4].

Table 1
LVF performance: The size of consistent feature subsets decreases sharply after several initial runs

Dataset	P	N	M'	M	# Subsets evaluated	# Max subsets
LED-24	1200	24	12	5	230	2^{24}
Lung	32	56	19	4	155	2^{56}
Lymphography	148	18	8	6	215	2^{18}
Mushroom	7125	22	8	4	22	2^{22}
Par3+3	64	12	5	3	25	2^{12}
Promoters	106	57	15	4	187	2^{57}
Soybean	47	35	12	2	42	2^{35}
Splice	3190	60	19	9	284	2^{60}
Vote	435	16	13	8	215	2^{16}
Zoo	74	16	9	5	25	2^{16}

Table 1 describes the datasets where P is the number of instances, N is the original number of features, M is the size of the smallest consistent subset, and M' is the intermediate size of some consistent subsets output by LVF. The Par3+3 dataset is a modified version of the original Parity3 dataset. The target concept is the parity of the first three bits. It contains twelve features among which the first three are relevant, the next three are irrelevant, and the other six are redundant (duplicate of the first six). Table 1 also shows the possible maximum number of subsets (i.e., 2^N) and the results (M') after several runs of LVF (each run stands for an evaluation of a feature subset).

Analysis. The trend found in all the experiments is that M' drops sharply in the beginning from N in a small number of runs (as shown in Table 1). Afterwards, it takes quite a long time to further decrease M' . Two typical graphs for Mushroom and Promoters datasets are shown in Fig. 3. Some analysis can confirm this finding. At the beginning, each feature subset has a probability of $1/2^N$ to be generated, and 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 Fig. 3. The finding is that LVF reduces the number of features quickly during the initial stages; after that LVF still searches in the same way (i.e., blindly) while the computing resource is spent on generating many subsets that are obviously not good. This analysis paves the way for the next algorithm which is a hybrid of LVF and ABB.

4.5. QBB: Hybrid search

QBB is a hybrid of LVF and ABB. Analyses showed that ABB's performance is good when $N - M$ is small whereas LVF is good in reducing the size of consistent feature subset very quickly in the early stages. QBB (**Q**uick **B**ranch and **B**ound) is a two phase algorithm that runs LVF in the first phase and ABB in the second phase. In the following QBB algorithm is given.

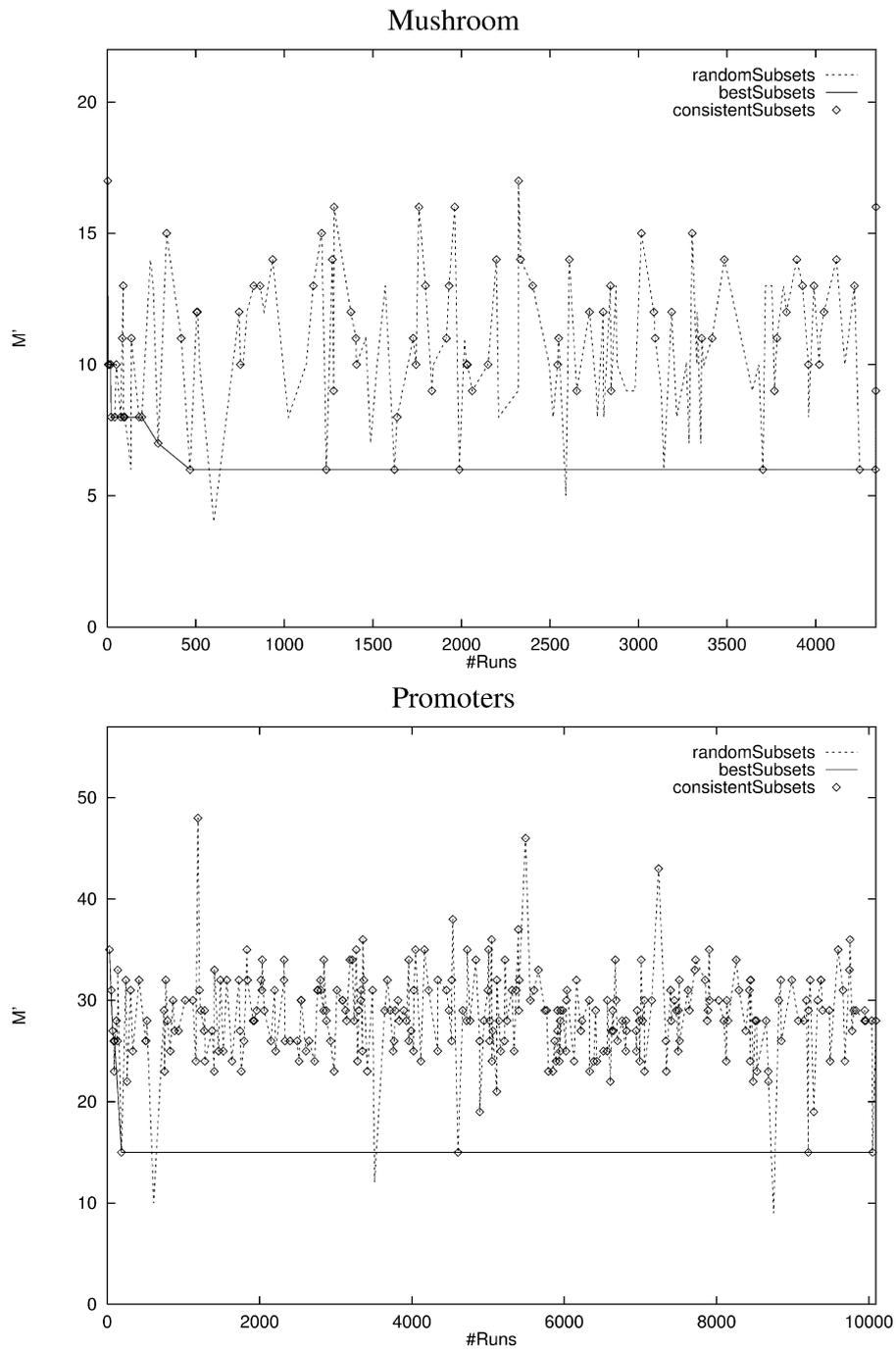


Fig. 3. LVF performance over Mushroom and Promoters datasets: The size of the consistent feature subsets decreases sharply but flattens afterwards.

Algorithm. QBB**Input:** Data D , feature set S , $MaxTries$ **Output:** Consistent Feature Subset

1. $\delta = \text{inConCal}(S, D)$
2. $T = S$
3. $S' = \text{LVF}(S, MaxTries, D)$
/* All consistent subsets which are output by LVF are in S' */
4. $MinSize = |S|$
5. for each subset S_j in S'
6. $T' = \text{ABB}(S_j, D)$
7. if $MinSize > |T'|$
8. $MinSize = |T'|$
9. $T = S_j$
10. return T

The rationale behind running LVF first is that a small number of runs of LVF from the beginning output consistent subsets of small size. Similarly, running ABB on these smaller subsets makes it very focused towards finding the smallest consistent subsets.

A performance issue. A key issue about QBB performance is: what is the crossing point at which ABB takes over from LVF. We carried out experiments over the earlier reported datasets by assigning a fixed total number of runs (a run means evaluation of a feature subset) and by varying the cross-over point. It showed that dividing the total runs equally between LVF and ABB is a robust solution and is more likely to yield the best results. Details of the experiments are reported in [12]. A simple analysis shows that if the crossing point is quite early, then consistent subsets output by LVF will be large in size for ABB to perform well under time constraint, but if the crossing point is close to the total number of runs, then LVF return very small size subsets which ABB may not be able to reduce further.

4.6. Guidelines

Above we described five different search strategies for consistency measure and discussed their pros and cons. In this section we provide guidelines for a user to select the best algorithm under particular circumstances.

Theoretical analysis and experimental results 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 (i.e., $N - M$ is small), ABB should be chosen; but ABB will take a long time for even moderately large $N - M$. Consider the dataset Parity3+3 which has three relevant features out of twelve (see Table 1). In such cases FocusM is preferred over ABB as M is very small. Similarly, consider the vote dataset (see Table 1) that has $N = 16$ and $M = 8$. ABB due to its early pruning of inconsistent subsets, evaluates only 301 subsets to output a subset of minimal size whereas FocusM has to evaluate 39,967 subsets to output a subset of the minimal size. But for datasets with large numbers of features (large N)

and moderate to large number of relevant features (moderate to large M), FocusM and ABB should not be expected to terminate in realistic time. An example is the Letter dataset (taken from UCI ML repository [4]) that has 20,000 instances ($N = 16$ and $M = 11$). Both FocusM and ABB take very long time to find minimal size feature subsets. Hence, in such cases one should resort to heuristic or probabilistic search for faster results. Although these algorithms may not guarantee minimal size subsets but they will be efficient in generating consistent subsets of size close to minimal in much less time. SetCover is heuristic, fast, and deterministic but it may face problems for data having highly correlated features. An example is Corral dataset that has a feature correlated to the class variable 75% of the time (see Section 4.3 for discussion). Because of the correlated feature it outputs a wrong subset. LVF is probabilistic, not prone to the problem faced by SetCover, but slow to converge in later stages. As we have shown in Table 1 and Fig. 3, it reduces the feature subset size quickly in the beginning but then it slows down afterwards. An example is Promoters dataset that has $N = 57$ and $M = 4$. LVF requires to evaluate only 187 subsets to output a subset of size 15, but for next 10's of thousands of subsets its output subset size does not further decrease. QBB, by design, captures the best of LVF and ABB. As verified by experiments, it is reasonably fast (slower than SetCover), robust, and can handle features with high correlation.

In summary,

- (1) If time is not an issue, then FocusM and ABB are preferable because they ensure smallest consistent subset. In addition, if the user has knowledge of M then if M is small FocusM is preferable otherwise ABB is chosen. In the absence of any a priori knowledge about M both can be run simultaneously until any one terminates.
- (2) In the usual case of limited computing time a user is best off choosing from LVF, SetCover and QBB. LVF is suitable for quickly producing small (yet not small enough) consistent subsets. SetCover is suitable if it is known that features are not correlated. Otherwise, QBB is a robust choice.

5. Further experiments

We carry out further experiments to examine (1) whether features selected using consistency measure can achieve the objective of dimensionality reduction without sacrificing predictive accuracy and (2) how the different algorithms fare in terms of time and size of consistent subsets. The experimental procedure is to run ABB to get the minimal size, compare the performance (average time and size of selected subsets) of different algorithms; and compare the accuracy of two different classifiers (C4.5 [36] and Backpropagation neural network [46]) before and after feature selection by QBB (which is the most robust among all methods). Ten benchmark datasets, as reported earlier in Table 1, are taken for the experiments.

5.1. Performance comparison

Fig. 4 shows the average time and average number of selected features over 10 datasets of different algorithms. First, ABB is run over the 10 datasets to find the M (minimum

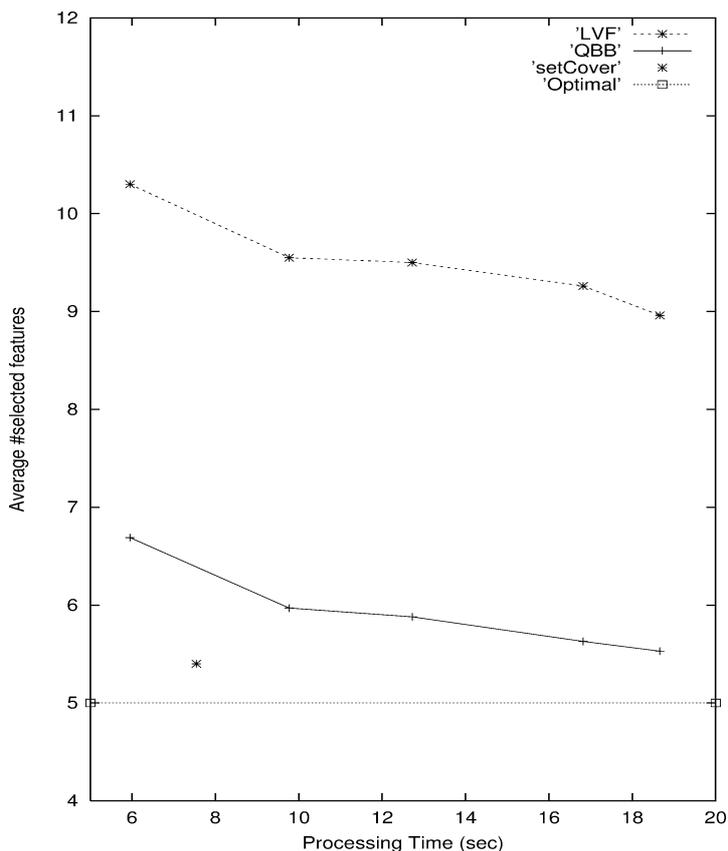


Fig. 4. Comparison of different algorithms: Results of FocusM and ABB are out of bounds due to their very large processing time.

number of relevant features) values for each dataset. Their average is 5 which is denoted as M_{avg} . This value is used as a reference line (in the bottom of the graph) in Fig. 4. Out of the 5 competing algorithms, FocusM, ABB and SetCover are deterministic, whereas LVF and QBB are non-deterministic due to their probabilistic nature. As per the findings of [12], 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 for each dataset. 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 datasets having large differences between N and M values such as Lung Cancer, Promoters, Soybean, and Splice, ABB takes very long time (a number of hours) to terminate. For datasets having large N values and not very small M values such as Splice dataset ($N = 60$, $M = 9$) FocusM takes many hours to terminate. The comparison in Fig. 4 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 some features are highly correlated.

5.2. Classification error

Predictive accuracy of a classifier is often used as a validation criterion. Classification error holds a relationship with predictive accuracy as the sum of predictive accuracy and error rate is 1. Among the different search strategies discussed in the paper, we choose QBB due to its robustness in the following experiments. We choose C4.5 (a widely used decision tree induction algorithm) [36] and Backpropagation neural networks [46] as two classifiers for validation.

For C4.5, we use the default settings: apply it to datasets *before* and *after* feature selection, and obtain the results of 10-fold cross-validation in Table 2. In the table we report average tree size and average error rate for each dataset after pruning. As QBB is non-deterministic, QBB is run 10 times for each dataset, i.e., 10 feature subsets are output by QBB for each dataset. The average for “before feature selection” is over the 10-fold cross-validation of C4.5, and the average for “after feature selection” is over the chosen 10 subsets of selected features and for each feature subset the 10-fold cross validation is performed, i.e., average is taken the total 10 10-fold cross validation for C4.5. We estimate the P-value using two-tailed t-test for two-sample unequal variances ($\alpha = 0.05$). For the t-test we test the null hypothesis that the means of the results “before feature selection” and “after feature selection” are equal. The experiment shows the improvement or no significant decrease in performance for most datasets (8 out of 10) in C4.5’s accuracy after feature selection. For Led24 dataset the P-values show “divide by 0” because the tree size is 19.0 for all folds both before and after feature selection, and similarly the error rate is 0.0 for all folds. So we put “1” in the place for P-values.

For the experiments with Backpropagation, each dataset is divided into a training set (two-third of the original size) and the rest one-third for testing. Running Backpropagation is very time consuming and involves the setting of some parameters, such as the network

Table 2
C4.5 decision tree results of QBB

Datasets	Tree size			Error rate		
	Before	After	P-value	Before	After	P-value
LED-24	19.0	19.0	1.0	0.0	0.0	1.0
Lung	17.8	12.9	0.013	66.7	54.5	0.253
Lymphography	26.9	19.5	0.001	20.8	19.7	0.757
Mushroom	30.8	40.5	0.0	0.0	0.0	0.057
Par3+3	15.6	15.0	0.638	31.4	0.0	0.0
Promoters	22.6	13.8	0.0	16.9	35.9	0.0
Soybean	7.1	8.0	0.0	4.0	0.2	0.57
Splice	207.7	535.0	0.0	5.6	31.2	0.0
Vote	16.0	16.9	0.48	4.4	4.4	1.0
Zoo	18.0	18.4	0.777	10.6	7.6	0.539

Table 3
Backpropagation neural network results of QBB

Datasets	Cycles	Hidden units	Error rate	
			Before	After
LED-24	1000	12	0.06	0.0
Lung	1000	28	75.0	75.0
Lymphography	7000	9	25.0	25.0
Mushroom	5000	11	0.0	0.0
Par3+3	1000	6	22.2	0.0
Promoters	2000	29	46.8	25.0
Soybean	1000	18	10.0	0.0
Splice	6000	30	25.64	42.33
Vote	4000	8	6.7	4.0
Zoo	4000	8	10.3	3.4

structure (number of layers and number of hidden units), learning rate, momentum, number of CYCLES (epochs). 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 dataset. 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 datasets. The experiment is carried out in two steps: (1) a trial run to find a proper number of CYCLES for each dataset which is determined by a sustained trend where error does not decrease; and (2) two runs on datasets 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 3 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.

6. Conclusion and future directions

In this paper, we carry out a study of consistency measure with different search strategies. The study of the consistency measure with other measures shows that it is monotonic, fast, multivariate, capable of handling some noise and can be used to remove redundant and/or irrelevant features. As with other filter measures, it is not clear that it also optimizes the accuracy of a classifier used after feature selection. We investigate different search strategies for consistency measure, which are: exhaustive, complete, heuristic, probabilistic, and hybrid. Their advantages and shortcomings are discussed and an experimental comparison is done to determine their relative performance. Some guidelines are given to choose the most suitable one in a given situation.

The fact that the consistency measure does not incorporate any search bias with regards to a particular classifier enables it to be used with a variety of different learning algorithms. As shown in the second set of experiments with 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 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 apparent was the case for the Splice dataset where both C4.5 and Backpropagation's performance deteriorate after feature selection.

This work can be extended in various directions. We plan to explore a line of research that focuses on comparison of different heuristic methods over consistency measure. We discussed in Section 4 that there are other heuristic search techniques such as genetic algorithms, simulated annealing, tabu search, and estimation of distribution algorithms. SetCover was the choice of heuristic search based on the known fact that these above algorithms are costly in computational time. However, the actual comparison results among these techniques can as well be interesting from research point of view.

Acknowledgements

We would like to thank Amit Mandvikar for help in performing experiments. Our understanding of the subject also enhanced by numerous discussions with Prof. Hiroshi Motoda.

References

- [1] H. Almuallim, T.G. Dietterich, Learning boolean concepts in the presence of many irrelevant features, *Artificial Intelligence* 69 (1–2) (1994) 279–305.
- [2] D.A. Bell, H. Wang, A formalism for relevance and its application in feature subset selection, *Machine Learning* 41 (2000) 175–195.
- [3] M. Ben-Bassat, Pattern recognition and reduction of dimensionality, in: P.R. Krishnaiah, L.N. Kanal (Eds.), *Handbook of Statistics*, North-Holland, Amsterdam, 1982, pp. 773–791.
- [4] C.L. Blake, C.J. Merz, UCI repository of machine learning databases, 1998, <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
- [5] A.L. Blum, P. Langley, Selection of relevant features and examples in machine learning, *Artificial Intelligence* 97 (1997) 245–271.
- [6] A. Blumer, A. Ehrenfeucht, D. Haussler, M.K. Warmuth, Occam's razor, in: J.W. Shavlik, T.G. Dietterich (Eds.), *Readings in Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1990, pp. 201–204.
- [7] G. Brassard, P. Bratley, *Fundamentals of Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1996.
- [8] C. Cardie, Using decision trees to improve case-based learning, in: *Proceedings of Tenth International Conference on Machine Learning*, Amherst, MA, 1993, pp. 25–32.
- [9] S. Das, Filters, wrappers and a boosting-based hybrid for feature selection, in: *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, Williamstown, MA, 2001, pp. 74–81.
- [10] M. Dash, Feature selection via set cover, in: *Proceedings of IEEE Knowledge and Data Engineering Exchange Workshop*, Newport, CA, IEEE Computer Society, 1997, pp. 165–171.
- [11] M. Dash, H. Liu, Feature selection methods for classification, *Intelligent Data Analysis: An Internat. J.* 1 (3) (1997).
- [12] M. Dash, H. Liu, Hybrid search of feature subsets, in: *Proceedings of Pacific Rim International Conference on Artificial Intelligence (PRICAI-98)*, Singapore, 1998, pp. 238–249.
- [13] M. Dash, H. Liu, H. Motoda, Consistency based feature selection, in: *Proceedings of Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Kyoto, Japan, 2000, pp. 98–109.
- [14] P.A. Devijver, J. Kittler, *Pattern Recognition: A Statistical Approach*, Prentice Hall, Englewood Cliffs, NJ, 1982.

- [15] J. Doak, An evaluation of feature selection methods and their application to computer security, Technical Report, University of California, Department of Computer Science, Davis, CA, 1992.
- [16] M.A. Hall, Correlation-based feature selection for discrete and numeric class machine learning, in: *Proceedings of Seventeenth International Conference on Machine Learning (ICML)*, Stanford, CA, Morgan Kaufmann, San Mateo, CA, 2000, pp. 359–366.
- [17] I. Inza, P. Larranaga, R. Etxeberria, B. Sierra, Feature subset selection by Bayesian network-based optimization, *Artificial Intelligence* 123 (2000) 157–184.
- [18] G.H. John, R. Kohavi, K. Pfleger, Irrelevant features and the subset selection problem, in: *Proceedings of the Eleventh International Conference on Machine Learning*, New Brunswick, NJ, 1994, pp. 121–129.
- [19] D.S. Johnson, Approximation algorithms for combinatorial problems, *J. Comput. System Sci.* 9 (1974) 256–278.
- [20] K. Kira, L.A. Rendell, The feature selection problem: Traditional methods and a new algorithm, in: *Proceedings of AAAI-92*, San Jose, CA, 1992, pp. 129–134.
- [21] R. Kohavi, Wrappers for performance enhancement and oblivious decision graphs, PhD Thesis, Department of Computer Science, Stanford University, CA, 1995.
- [22] R. Kohavi, G.H. John, Wrappers for feature subset selection, *Artificial Intelligence* 97 (1–2) (1997) 273–324.
- [23] D. Koller, M. Sahami, Toward optimal feature selection, in: *Proceedings of International Conference on Machine Learning*, Bari, Italy, 1996, pp. 284–292.
- [24] I. Kononenko, Estimating attributes: Analysis and extension of RELIEF, in: *Proceedings of European Conference on Machine Learning*, Catania, Italy, 1994, pp. 171–182.
- [25] H. Liu, F. Hussain, C.L. Tan, M. Dash, Discretization: An enabling technique, *J. Data Mining Knowledge Discovery* 6 (4) (2002) 393–423.
- [26] H. Liu, H. Motoda (Eds.), *Feature Extraction, Construction and Selection: A Data Mining Perspective*, Kluwer Academic, Boston, MA, 1998.
- [27] H. Liu, H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*, Kluwer Academic, Dordrecht, 1998.
- [28] H. Liu, H. Motoda, M. Dash, A monotonic measure for optimal feature selection, in: *Proceedings of European Conference on Machine Learning*, Chemnitz, Germany, 1998, pp. 101–106.
- [29] H. Liu, 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, pp. 419–424.
- [30] H. Liu, R. Setiono, A probabilistic approach to feature selection—A filter solution, in: *Proceedings of International Conference on Machine Learning*, Bari, Italy, 1996, pp. 319–327.
- [31] M. Modrzejewski, Feature selection using rough sets theory, in: P.B. Brazdil (Ed.), *Proceedings of the European Conference on Machine Learning*, Vienna, Austria, 1993, pp. 213–226.
- [32] A.W. Moore, M.S. Lee, Efficient algorithms for minimizing cross validation error, in: *Proceedings of Eleventh International Conference on Machine Learning*, New Brunswick, NJ, Morgan Kaufmann, San Mateo, CA, 1994, pp. 190–198.
- [33] A.N. Mucciardi, E.E. Gose, A comparison of seven techniques for choosing subsets of pattern recognition, *IEEE Trans. Comput.* C-20 (1971) 1023–1031.
- [34] P.M. Narendra, K. Fukunaga, A branch and bound algorithm for feature selection, *IEEE Trans. Comput.* C-26 (9) (1977) 917–922.
- [35] A.L. Oliveira, A.S. Vincentelli, Constructive induction using a non-greedy strategy for feature selection, in: *Proceedings of Ninth International Conference on Machine Learning*, Aberdeen, Scotland, Morgan Kaufmann, San Mateo, CA, 1992, pp. 355–360.
- [36] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.
- [37] T.W. Rauber, Inductive pattern classification methods—Features—Sensors, PhD Thesis, Department of Electrical Engineering, Universidade Nova de Lisboa, 1994.
- [38] 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*, Amherst, MA, 1993, pp. 284–290.
- [39] J. Segen, Feature selection and constructive inference, in: *Proceedings of Seventh International Conference on Pattern Recognition*, Montreal, QB, 1984, pp. 1344–1346.

- [40] J. Sheinvald, B. Dom, W. Niblack, A modelling approach to feature selection, in: Proceedings of Tenth International Conference on Pattern Recognition, Atlantic City, NJ, Vol. 1, 1990, pp. 535–539.
- [41] W. Siedlecki, J. Sklansky, On automatic feature selection, *Internat. J. Pattern Recognition Artificial Intelligence* 2 (1988) 197–220.
- [42] D.B. Skalak, Prototype and feature selection by sampling and random mutation hill-climbing algorithms, in: Proceedings of Eleventh International Conference on Machine Learning, New Brunswick, NJ, Morgan Kaufmann, San Mateo, CA, 1994, pp. 293–301.
- [43] P. Soucy, G.W. Mineau, A simple feature selection method for text classification, in: Proceedings of IJCAI-01, Seattle, WA, 2001, pp. 897–903.
- [44] H. Vafaie, I.F. Imam, Feature selection methods: Genetic algorithms vs. greedy-like search, in: Proceedings of International Conference on Fuzzy and Intelligent Control Systems, 1994.
- [45] S. Watanabe, *Pattern Recognition: Human and Mechanical*, Wiley Interscience, New York, 1985.
- [46] A. Zell, et al., *Stuttgart Neural Network Simulator (SNNS)*, User Manual, Version 4.1, Technical Report, 1995.