A Branch and Bound Algorithm for Feature Subset Selection

PATRENAHALLI M. NARENDRA AND KEINOSUKE FUJIKAWA

Abstract—A feature subset selection algorithm based on branch and bound technique is developed to select the best subset of m features from an n feature set. Existing procedures for feature subset selection, such as sequential selection and dynamic programming, do not guarantee optimality of the selected feature subset. Exhaustive search, on the other hand, is generally computationally unfeasible. The proposed algorithm is very efficient and it selects the best subset without exhaustive search. Computational aspects of the algorithm are discussed. Results of several experiments demonstrate the very substantial computational savings realized. For example, the best 11 feature set from a 24 feature set was selected with the computational effort of evaluating only 6600 subsets. Exhaustive search would require the evaluation of 2,704,116 subsets.

Index Terms—Branch and bound, combinatorial optimization, feature selection, recursive computation.

I. INTRODUCTION

The problem of feature subset selection is to select a subset of (m) features from a larger set of (n) features in accordance to optimize the value of a criterion over all subsets of size m. There are \( \binom{n}{m} \) = \( \frac{n!}{m!(n-m)!} \) such subsets. Exhaustive enumeration of all the subsets is computationally prohibitive, as the number of subsets to be considered grows very rapidly with the number of features; for example, \( \binom{10}{5} = 252 \) while \( \binom{100}{50} = 2.048 \times 10^{41} \), which is 7,041,161. Heuristic techniques [1] and dynamic programming solutions [2] are more efficient, but they avoid exhaustive enumeration, but they offer no guarantee that the selected subset yields the best value of the criterion among all subsets of size m.

Here, we present a branch and bound formulation of the feature subset selection problem. The algorithm is very efficient because it avoids exhaustive enumeration by rejecting superfluous subsets without direct evaluation and guarantees that the selected subset yields the globally best value of any criterion that satisfies monotonicity. Branch and bound methods are powerful combinatorial-optimization tools and similar formulations have been applied to other problems in pattern recognition, such as clustering [3] and nearest neighbor computation [4]. Here an efficient subset enumeration scheme is developed to realize maximum advantage of the branch and bound principle. Several equations which facilitate rapid computation are derived for the case of quadratic criteria such as the discriminant function, divergence and Bhattacharyya distance for the normal case, etc. A suboptimal variant of the globally optimal branch and bound algorithm has also been presented. Results of comparison experiments demonstrating the efficiency of the algorithms are also included.

Manuscript received August 11, 1978; revised June 18, 1979. This work was supported in part by NSF Grant CII-7723727.

II. THE BRANCH AND BOUND ALGORITHM

Let the number of features in the original set be n. We have to select a subset of m features so that the value of a criterion is optimized over all subsets of size m. Let \( \mathcal{Z}_i \) be the \( [m-1] \) features to be discarded to obtain an m feature subset. Each variable \( Z_i \) can take on values in \( \{1, 2, ..., n\} \). But the order of the \( Z_i \) is immaterial, hence, \( \mathcal{Z}_i \) will contain only unique values of \( Z_i \) such that

\[
2 < Z_1 < \ldots < Z_m \tag{1}
\]

A more general enumeration of the subsets will be given later. The feature subset selection criterion is \( J(Z_1, \ldots, Z_m) \), a function of the m features obtained by discarding \( Z_{m+1}, \ldots, Z_n \) from the m feature set. The feature subset selection problem is to find the optimum subset \( \mathcal{Z}_m \) such that

\[
J(Z_1, \ldots, Z_m) = \max J(Z_1, \ldots, Z_m) \quad Z_1, \ldots, Z_m \in \mathcal{Z}_m \tag{2}
\]

Fig. 1 is a decision tree enumerating all the possible subsets satisfying relation (1) for \( m = 6 \) and \( n = 20 \). Each node is uniquely identified by the discarded feature, for example (14) for node A.

Let us assume that the criterion J satisfies monotonicity, which is defined by

\[
J(Z_1) \leq J(Z_1, Z_2) \leq \ldots \leq J(Z_1, \ldots, Z_m) \quad (3)
\]

The monotonicity is not particularly restrictive; it merely means that a subset of features should not be better than any larger set that contains the subset. Indeed, a large variety of feature selection criteria do satisfy monotonicity, for example, Bhattacharyya distance and divergence are monotone.

Let \( l \) be a lower bound on the optimum (maximum) value of the criterion \( J(Z_1, \ldots, Z_m) \) defined as

\[
V \leq J(Z_1, \ldots, Z_m) \quad (4)
\]

This means that whenever the criterion evaluated for an m is less than the bound B, all nodes that are descendants of that node also have criterion values less than B, and therefore cannot be the optimum solution. This forms the basis for the branch and bound algorithm. The branch and bound algorithm successively generates portions of the solution tree and combines the criteria. Whenever a suboptimal partial sequence of m is found to satisfy (4), the subtree under the node is implicitly rejected, and enumeration begins on partial sequences which have not yet been explored.

The Enumeration Scheme

We note that with reference to Fig. 1, the nodes at a given level do not all have the same number of terminal nodes. Node (12) has three successors, while node (14) has only one. As a result, if the subsequences set (4) is satisfied for node (12), i.e., if \( J(12) \leq B \), its sequences are rejected as being suboptimal, while for node (14), only the single sequence

1 If \( Z_i \) can take the best value of \( J(Z_1, \ldots, Z_m) \) found so far in the search, for example.
The Branch and Bound Algorithm

The following notation will be used in the algorithm.

L(B): An ordered list of the features enumerated at level i.

P(i): The pointer to the element in L(i) being currently considered. For example, if the current element in L(i) is the 4th, then P(i) = 4.

SUCCESSOR(i): The number of successors that the 4th element in L(i) can have.

AVAL: A list of available feature values that (L(i)) can assume.

Step 1: (Initialization): Set B = ∅, AVAL = {1, 2, ..., m} + 1. Let L(0) = ∅, SUCCESSOR(1) = 1, P(1) = 1.

Step 2: [Definition of L(0)] Set NODE = P(0). (−1). Compute J(L(0)), ..., Z(L(0))-1) for all k in AVAL. Rank the features in AVAL in the increasing order of J(L(0), ..., Z(L(0))-1) and store the smallest y of these in L(0) being the feature in AVAL yielding the smallest J(L(0)). Set P(1) = SUCCESSOR(1), NODE. Set SUCCESSOR(j) = p-j+1, for j = 2, ..., p. Remove L(0) from AVAL.

Step 2: [Reject new node]: If L(0) is empty, go to Step 4. Otherwise, set Z = A where A is the last element in L(0), set P(1) = 1, where Z is the current number of elements in L(0). Delete A from L(0).

Step 3: [Check bounds]: If L(0), ..., Z < B, return Z to AVAL and go to Step 4. If level = m, go to Step 5. Otherwise, set i = 1 and go to Step 2.

Step 4: [Backtrack]: Set i = i + 1. If i < n, terminate the algorithm. Otherwise, return Z to AVAL and go to Step 2.

Step 5: [Final level, update bounds]: Set Z = L(0), ..., Z and save (Z, L(0), ..., Z) as Z, L(0), ..., Z. Return Z to AVAL. Go to Step 4.

The diagram in Fig. 2 illustrates the algorithm. The functioning of the algorithm is as follows: starting from the root of the tree, the successors of the current node enumerated in the ordered list L(i)(i). The successor, for which the partial criterion J(Z, i) > Z, is maximum (the rightmost successor), is picked up as the new current node and the algorithm moves on to the next higher level. The tree (L(i)) at each level keeps track of the nodes that have been explored. The successor variable determines the number of successors a node will have in the next level. AVAL keeps track of the feature values that can be enumerated at any level. Whenever the partial criterion is found to be less than the bound, the algorithm backtracks to the previous level and selects another unexplored node for expansion. Whenever the algorithm reaches the last level, the lower bound B is updated to the current value of J(Z, i), Z, and the current sequence (Z, L(0), ..., Z) is saved as (Z, L(0), ..., Z). When all the nodes in L(i) for given i are exhausted, the algorithm backtracks to the previous level. When the algorithm backtracks to level 0, it terminates.

Suboptimal Solutions

We have presented an efficient branch and bound formulation which guarantees global optimality of the subset under the assumption of convexity. If we allow global optimality to be compromised, it is possible to improve the efficiency even further. We present a scheme that employs look-ahead in the basic branch and bound algorithm; so that nonmonotonic solutions were detected higher up in the tree, eliminating more suboptimal solutions.

We define upper bounds b of the criterion for each stage as follows:

b is the best (largest) value of J(Z, i) < Z found so far in the search. It is updated every time a new (Z, L(0), ..., Z) is computed to be greater than the current value of b. In Step 3 of the algorithm, we replace the test J(Z, i) < Z by J(Z, i) < b, where b is an integer constant representing the look-ahead factor. It is understood that b < b < b is an integer constant representing the look-ahead factor. It is understood that b < b < b is an integer constant representing the look-ahead factor.

With this change, instead of comparing the partial criterion J(Z, i) with the bound b, we compare with J(Z, i) - b < b, where b is an integer constant representing the look-ahead factor. For i = m - 1 minus b, 2 b, more branches are eliminated in Step 3 of the algorithm and only the more promising nodes are examined at higher levels of the tree. So,
for 1 < i ≤ 1 the algorithm enumerates fewer nodes and is therefore more efficient, although now there is no guarantee that the subset obtained is globally optimal. Incidentally, for i = 0 the algorithm approximates the familiar backward selection scheme. Therefore, we can expect that for 0 < i ≤ 1 the algorithm would fall between the backward selection scheme and the globally optimal branch and bound algorithm, both in efficiency and optimality of the subset obtained. Indeed, this is borne out by the experimental results reported in Section IV.

III. RECURSIVELY COMPUTABLE CRITERIA

We noted in the previous section that the algorithms are implemented with the criterion evaluated for the partial sequences \(X_1, \ldots, X_i\). The nature of the enumeration schemes requires that the value of the criterion be computed successively as features are deleted from the full set. For the class of quadratic criteria, we derive recursive equations to evaluate the criterion as a new feature is deleted from the present partial set. We will first consider the following quadratic form

\[ J_4 = X_i S_4 X_i^T \tag{6} \]

where \(X_i\) is a \(k\)-vector and \(S_4\) a \(k\times k\) positive definite matrix when \(k\) features are present.

The quadratic form is the basis of various quadratic criteria such as the discriminant function, the Fisher criterion, and Mahalanobis distance. Bhattacharyya's distance and the divergence, for the normal case, are terms of the same form as (6).

A. Recursive Computation of \(S_4^{-1}\)

The inversion of \(S_4\) is the major computational effort in evaluating (6) as features are successively deleted from the full set of features. When the ith feature is deleted, it is necessary to compute the inverse of \(S_4\) with the ith row and column deleted. Without loss of generality, let the feature being deleted correspond to the ith row and column of \(S_4\).

\[ S_4 = \begin{bmatrix} S_{i+1} & Y' \\ Y' & \tau \end{bmatrix} \begin{bmatrix} A^{-1} & 1 \\ 1 & 1 \end{bmatrix} \tag{7} \]

A fundamental identity in matrix algebra [7] gives \(S_i^{-1}\) in terms of \(S_i\), as

\[ S_i^{-1} = \frac{1}{d} \begin{bmatrix} S_{i+1}^{-1} + \frac{1}{d} S_{i+1}Y' \tau S_{i+1}^{-1} & \frac{1}{d} S_{i+1}Y' \\ \frac{1}{d} S_{i+1}Y' & \frac{1}{d} \end{bmatrix} \tag{8} \]

where

\[ d = \tau - Y' \tau S_{i+1}^{-1} Y \tag{9} \]

If we write

\[ S_i' = \begin{bmatrix} A & C \\ C^T & d \end{bmatrix} \begin{bmatrix} A' & 1 \\ 1 & 1 \end{bmatrix} \tag{10} \]

then, by (8) and (10) it can be verified that

\[ S_i^{-1} = A' - \frac{1}{d} C C^T \tag{11} \]

Hence, \(S_{i+1}\) can be computed from \(S_i\) with little computational effort. With reference to the algorithm, the inverse matrices \(S_i^{-1}\) are stored for each level. The inverse at any level is computed from that of the previous level using the recursive equation (11). Whenever the algorithm backtrack proceeds down another branch, the inverse for the new \(S_i\) can be recomputed from the inverse at the level at which the branching occurred. For example, suppose as in Fig. 2, after level 2 is explored, the algorithm backtracks to the node 12,3. The value of \(S_2^{-1}\) for the node 12,3 can be computed from the current value of \(S_2^{-1}\) at level 1. The \(S_i'\) for level two is updated to be this value as feature 5 is now chosen to be \(S_5\).
B. Recursive Computation of $X_1S_1X_2$

It is also possible to recursively compute the function $X_1S_1X_2$ by $k = 1$ features given to value with $k$ features and the $X_2$ features from the previous level. This is useful in a sense of the algorithm where it is necessary to compute the criterion after feature selection is done to a stage from a partial list. This avoids computing $S_1^2$ during the stage. Once a node is selected however (step 1) the $S_1^2$ for the new level has to be updated using (11).

Let the criterion with a feature being deleted by $j_1 = X_1S_1X_2$. Then let us assume that the $k$ feature is being deleted at before $X_1S_1X_2 = j_1$. This is the criterion with $k - 1$ features where

$$X_1S_1X_2 = j_1$$

and $j_1$ is defined as in (11).

As a consequence of (11), it is easy to verify that

$$X_1S_1X_2 = X_1S_1X_2$$

$$X_1S_1X_2 = X_1S_1X_2 - X_1S_1X_2 - X_1S_1X_2 + X_1S_1X_2$$

Note that (11) is a row of $S_1$ corresponding to the features being deleted. Notice (12) $X_1$ is being reordered of that tree with $X_2$. Thus, the $j_1$ for $j_1 = 1, 2, \ldots, m$ the algorithm may be directly evaluated from $j_1, j_2, \ldots, j_m$ by (13) without actually having to compute $S_1^2$ for all the variables $j$. Incidentally, (13) also furnishes proof that $j$ is in decreasing.

v. COMPUTER EXPERIMENTS

The algorithms were tested on multiparameter data obtained from air pollution regression studies performed on the laboratory for applications for reticulo-inelastic (LaS) at Purdue University.

The data were considered by 42 sample volumes each, from two classes normalized by kriging and 20%. There were 12 data channels corresponding to 12 bands of the spectrum in which the system was preprogrammed. Each channel $j$ of, course, a feature and the problem was to select a subset of the features which was true, according to a given criterion. The criterion chosen was the criterion function defined as follows:

$$J_0 = (M_0 - M_0^T X_0^T X_0^{-2} X_0 M_0 - M_0)$$

where $M_0$ are the sample means for the two classes and $X_0$ are the within class scatter matrices.

First, the algorithm was applied to choose the best four out of the 12 channels. There are $12! = 479$ four-channel subsets. The total number of partial solutions of all sites examined (values modeled was 36. Since the criterion it is to be evaluated once for each node, basically, this would make a computational saving of the order of left exhaustive search. However, we note that the recursive equations were used for evaluating the bounds. Hence, we estimate a further saving of the order of four because four feature subsets would be considered as exhaustive search without recursive computation.

The experiment was repeated with data from additional classes such as 12, 24, 36, 48, etc., taken in the same way. The average number of subsets examined was 560, with a minimum of 50 and a maximum of 54.

To evaluate the performance of the algorithm for large problems, an additional set of 12 features was generated by taking the square of the first 12 features. The remaining set of the same size was included for the 24 features set. In other words, the algorithm was to select the best 24 features for the 24 feature set. As can be expected, the resulting selection was generally, there may be several solutions that yield very close values of the criterion.

The optimal algorithm was applied to the 24 feature problem to choose the best 12 features. There were $12!(12^2)$ = 36, 155, 900 solutions. The algorithm required field 20 and such 20 of the PDP-11 central processing unit (CUP) used an average. This demonstrates a very considerable saving over the exhaustive search. Table 2 gives the number of nodes required and the number of nodes, for which the inequality (11) was satisfied at each level. Some of the nodes are not included in the table. The algorithm, however, included in Table 2, are considered the domain of the algorithm. The additional complexity of the feature ordering appears justified at the right of the efficiency also, this median the algorithm independent of the initial ordering of the features.

Suboptimal Solutions

The backstaged scheme of Section II was incorporated into the basic algorithm. Experiments were repeated with the 24 channel and 12 channel synthetic data. Table 1(c) and (d) gives the corresponding result with all different values of the backstaged factor for $F = 10$. The algorithm performs better with selection with the correct suboptimal scheme, and no backstaging is included in the global optimal scheme (the basic branch and bound algorithm). With the 12-channel example, we see that for $F = 1$, the optimum subset is obtained. But the number of nodes required for $F = 10$ is not substantially smaller than for the optimal case (10). The latter example in Table 1(c) is more indicative of the effectiveness of the scheme. The optimum subset is obtained with $F = 1$, and the number of nodes required is only 1008.
<table>
<thead>
<tr>
<th>LEVEL</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of nodes enumerated</td>
<td>12</td>
<td>91</td>
<td>223</td>
<td>821</td>
<td>1091</td>
<td>1874</td>
<td>2034</td>
<td>1742</td>
<td>1242</td>
<td>910</td>
<td>347</td>
<td>189</td>
</tr>
<tr>
<td>No. of nodes rejected</td>
<td>0</td>
<td>13</td>
<td>56</td>
<td>127</td>
<td>199</td>
<td>452</td>
<td>756</td>
<td>950</td>
<td>466</td>
<td>623</td>
<td>180</td>
<td>148</td>
</tr>
</tbody>
</table>

**TABLE II**

Results of Suboptimal Selection Scheme for the 12-Channel Example

<table>
<thead>
<tr>
<th>LEVEL OF OPTIMIZATION</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>(OPTIMAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes explored</td>
<td>8</td>
<td>22</td>
<td>36</td>
<td>36</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>CPU Time (Pop-10) in Seconds</td>
<td>0.48</td>
<td>0.98</td>
<td>0.98</td>
<td>7.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Criterion Value with Subset Selected</td>
<td>7.88</td>
<td>7.98</td>
<td>7.98</td>
<td>7.98</td>
<td>7.98</td>
<td>7.98</td>
</tr>
<tr>
<td>Feature Subset Selected</td>
<td>(4 6 10)</td>
<td>(4 6 10 11)</td>
<td>(4 6 10 11)</td>
<td>(4 6 10 11)</td>
<td>(4 6 10 11)</td>
<td>(4 6 10 11)</td>
</tr>
</tbody>
</table>

**TABLE III**

Results of the Suboptimal Selection Scheme for the 24-Channel Example

<table>
<thead>
<tr>
<th>LEVEL OF OPTIMIZATION</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>(OPTIMAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of nodes explored</td>
<td>10</td>
<td>102</td>
<td>389</td>
<td>1008</td>
<td>1790</td>
<td>2378</td>
<td>1877</td>
<td>5297</td>
<td>5549</td>
<td>5846</td>
<td>5846</td>
<td>5846</td>
<td>5846</td>
</tr>
<tr>
<td>CPU Time (Pop-10) Seconds</td>
<td>2.23</td>
<td>4.27</td>
<td>11.52</td>
<td>45.32</td>
<td>79.71</td>
<td>141.22</td>
<td>223.35</td>
<td>264.6</td>
<td>268.27</td>
<td>268.49</td>
<td>268.5</td>
<td>268.5</td>
<td>268.5</td>
</tr>
<tr>
<td>Feature Subset Selected</td>
<td>3 3</td>
<td>3 1</td>
<td>2 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>
VI. SUMMARY

A branch and bound algorithm to select the globally best feature subset, based on an efficient subset enumeration scheme, was presented. A suboptimal variant of the basic branch and bound algorithm was suggested. For a general class of feature selection criteria, recursive equations were developed to facilitate the implementation of the algorithm. Results of computer experiments were presented to substantiate the claim that the algorithms are very efficient.

REFERENCES


Correction to "An Application of Relaxation Labeling to Line and Curve Enhancement"

STEVEN W. ZUCKER, ROBERT A. HUMMEL, AND AZRIEL ROSEN Feld

In the above paper, Figs. 4-7, and 10-12 were inadvertently misrepresented. They are correctly displayed here.

Manuscript received June 8, 1977.

S. W. Zucker was with the Computer Science Center, University of Maryland, College Park, MD 20742. He is now with the Department of Electrical Engineering, McGill University, Montreal, PQ, Canada.

R. A. Hummel and A. Rosenfeld are with the Computer Science Center, University of Maryland, College Park, MD 20742.


Fig. 4. (a) Computer-generated line in noise. (b) Local probability assignments for LIA obtained from maximum line detector response. (c)-(g) Iterations 1-6. (h) and (i) relaxation process applied in (h).