CATÓLICA

UNIVERSIDADE CATÓLICA PORTUGUESA | PORTO

Faculdade de Economia e Gestão

DOCUMENTOS DE TRABALHO **WORKING PAPERS**

ECONOMIA **ECONOMICS**

Nº 01/2009

EXACT AND HEURISTIC ALGORITHMS FOR VARIABLE SELECTION: EXTENDED LEAPS AND BOUNDS

A. Pedro Duarte Silva

Universidade Católica Portuguesa (Porto)

Exact and heuristic algorithms for variable selection: Extended Leaps and Bounds

A. Pedro Duarte Silva

Faculdade de Economia e Gestão

Universidade Católica Portuguesa at Porto

E-mail: psilva@porto.ucp.pt

Keywords: Variable Selection Algorithms; All-Subsets; Heuristics

Abstract

An implementation of enhanced versions of the classical Leaps and Bounds algorithm for variable selection is provided. Features of this implementation include: (i) The availability of general routines capable of handling many different statistical methodologies and comparison criteria. (ii) Routines designed for exact and heuristic searches. (iii) The possibility of dealing with problems with more variables than observations.

The implementation is supplied in two different ways: i) as a C++ library with abstract classes that can be specialized to different problems and criteria. ii) as a console application ready to be applied to searches according to some of the most important comparison criteria proposed to date.

The code of the C++ library and console application described here, can be freely obtained by sending an email to the author.

1. Introduction

A recurrent theme in applied statistics and data analysis is the selection, or comparison, of the subsets extracted from an initial pool of p candidate variables. Traditional approaches to this problem use various comparison criteria to assess the relative merits of different subsets. However, as the number of alternative sets ($= 2^p-1$) increases exponentially with the number of original variables, an exhaustive evaluation of all-possible subsets quickly becomes infeasible and greedy strategies, such as stepwise methods, are often used in practice. More sophisticated approaches based on general heuristics or meta-heuristics have also been proposed (e.g. Winkler and Gilli 2004) and are occasionally employed.

In linear regression problems, exact algorithms capable of handling data sets with a moderate number of original variables are known since the nineteen seventies. One of the most successful of these algorithms is Furnival and Wilson's Leaps and Bounds (1974) which, more than 30 years after its development, remains a global standard against with other proposals need to be compared. Furthermore, the Leaps algorithm has two important, but less known, features that have not been yet exploited to its full potential: (i) Its basic search strategies can be adapted to a wide range o different statistical problems. (ii) For problems with a large number of variables, variants that relax strict optimality requirements lead to heuristics specialized to the variable selection problem.

The purpose of this article is to present a flexible extended version of the Leaps algorithm that may be used in statistical problems where comparisons of variable subsets

are warranted. This extended algorithm is provided, in the C++ code enclosed, as a library of search routines and general classes from which specializations to different problems can be derived. Furthermore, a console application that uses the library to perform subset searches according to different criteria for regression models, linear models with multivariate responses, discriminant analysis and exploratory data analysis problems, is also enclosed. Several other extensions and variants of the Leaps algorithm have been proposed before (e.g. McCabe 1975, Lawless and Singhal 1978, 1987a, 1987b, Hosmer et al. 1989, Duarte Silva 1998, 2001, 2002) however, none of them is as general, flexible and easily extendable as the one described here.

The original Leaps algorithm assumes the absence of perfect colinearities in its input data and can lead to numerical stability problems if the data is almost collinear. For that reason it has been subject to some criticism. For instance, Miller (2002) remarks that, by relying on matrices derived from sums of squares, this algorithm tends to magnify any round-off cancellation errors that may occur. In contrast, methods that use triangular factorizations of the original data do not suffer from the same problem and are numerically more stable. Miller developed an implementation of his own, available as an *R* package (*R* Development Core Team 2008) named *leaps*, that uses the QR decomposition of the original data, has better numerical properties than the original algorithm, and can handle the not so uncommon situation when there are more potential variables than observations. However, this implementation sacrifices some of the speed of the original algorithm, notably by increasing its worst case time complexity from $O(2^p)$ to $O(p^3 2^p)$. Other research has lead to exhaustive (Smith and Bremner, 1989) and branch and bound (Gatu and Kontoghiorghes, 2006) algorithms based on QR decompositions, with the same worst case time complexity as Furnival and Wilson's original proposal. Algorithms based on different

factorisations were also proposed (Kim 2000), but their numerical and computational properties are not well known.

The implementation described in this paper is based on the original proposal of Furnival and Wilson, but is capable of handling problems with more variables than observations, can be used as an exact search as well as an heuristic, and monitors the relative rounding error of the comparison criteria restricting the search to subsets where this error remains within controlled limits. This approach only ignores subsets with strong multicolinearity, which often should be avoided anyway because of their poor statistical properties. Furthermore, whenever the algorithms do no perform a full analysis of all possible subsets, an warning message is issued.

The remainder of the paper is structured as follows. The next section describes the Extended Leaps algorithmitic strategies. Section 3 presents the C++ library that implements these strategies and briefly describes a console application and an *R* package that use this library. Section 4 concludes the paper.

The code of the C++ library and console application described here can be freely obtained by sending an email to psilva@porto.ucp.pt.

2. Extended Leaps and Bounds Algorithms

Assume that there are p candidate variables to enter a given statistical model or data analysis methodology. Denote the full set comprising all p candidates by $S_1 = \{X_1, X_2, \ldots, X_p\}$ and the proper subsets of S_1 by $S_a \subset S_1$ ($a = 2, 3, \ldots, 2^p - 1$). Further assume that a relevant criterion for subset comparisons, $C(.)$, is known. Based on $C(.)$, the analyst wants to identify a pool of "interesting" sets for further analysis. Suppose that given known

criterion values for the sets S_a and S_c together with intermediate results stored in properly reserved memory locations, $M(.)$, it is possible for the sets of the form $S_b = S_a \cup \{ X_u \}$ and $S_d = S_c \setminus \{ X_u \}$ to compute the criterion values $C(S_b)$ and $C(S_d)$ with considerable less computational effort than it would be required if these values had to be evaluated from scratch. If intermediate results related to S_b or S_d are to be used in the evaluation of $C(.)$ for other subsets, then they also need to be updated and stored. The effort required for these updates is typically related to number of variables that are to be included or excluded from S_b or S_c in order to create sets derived from them. This effort will also be assumed to be smaller than the effort required to evaluate $C(.)$ without prior information.

Under these assumptions the computational effort to evaluate all $C(.)$ can be minimized by repeatedly updating the criterion through a sequence of subset evaluations such that the average number of variables that in latter steps will be added or removed to each subset is kept as low as possible. One simple strategy to achieve this goal, originally proposed by Furnival (1971), is to sequence the evaluations by increasing order of the binary representation index: $I(S_a) = \sum_{i=1}^{p} \delta(X_i)^{i-1}$ where $\delta(X_i) = 1$ if $X_i \in S_a$ and $\delta(X_i) = 0$ otherwise. If p different memory locations $M(1)$, $M(2)$, … , $M(p)$ are reserved in such way that $M(p)$ stores the original data and $M(u\text{-}1)$ stores intermediate results related to a subset that includes X_u but excludes X_1 through X_{u-1}, then it can be shown that, when updating $C(.)$ by adding a X_u to a previous set: (i) all necessary information can be retrieved from one of the available memory locations; (ii) only future updates based on adding variables X_1 through X_{u-1} will need the information stored in $M(u-1)$; (iii) $M(u\text{-}1)$ will need to be replaced only 2^{p-u-1} different times. Table1 exemplifies the sequence of criterion updates

and memory management for the case where *p*=3. In this table, *t* represents the number of variables for which intermediate results need to be updated at each step.

Table 1 about here

The original algorithm addressed the linear regression model $y = X\beta + \varepsilon$ using as comparison criterion the coefficient of determination $C(.) = R^2 = r_{yX}\ R_{XX}^{-1}\ r_{Xy}$ where R_{XX} stands for the matrix of correlations amongst the regressors and r_{yX} (r_{Xy}) for the row (column) vector of correlations between the regressors and the response[1]. When the variable X_u is added to the subset S_a leading to the subset $S_b = S_a \cup \{X_u\}$, C(.) can be updated by the formula:

$$C(S_b) = C(S_a) + \frac{(r_{y X_u | S_a})^2}{R_{\bar{S}_a \bar{S}_a | S_a}(u,u)}$$

which requires only two multiplication-type and one addition-type floating point operations (from now on referred to as flops), provided that the partial correlations

$$r_{yX_u|S_a} = r_{yX_u} - r_{yS_a}\ R_{S_aS_a}^{-1}\ R_{S_aX_u} \quad ; \quad R_{\bar{S}_a\bar{S}_a|S_a} = R_{\bar{S}_a\bar{S}_a} - R_{\bar{S}_aS_a}\ R_{S_aS_a}^{-1}\ R_{S_a\bar{S}_a}$$

are known.

In this case, the information required for evaluating *C(.)* in subsets derived from S_b consists on the $(u^2+u-2)/2$ different elements of the matrix $R_{\bar{S}_b\bar{S}_b|S_b}$ and vector $r_{yS_{\bar{b}}|S_b}$ associated with the variables X_1 through X_{u-1}. These elements can be computed in

[1] It is well known that the maximization of R^2 is equivalent to the classical criterion of minimizing the sum of squared residuals. Sometimes the algorithm is presented in terms of the later criterion.

($u^2+3u-4)/2$ multiplication and ($u^2+u-4)/2$ addition flops. It follows that the total number of multiplication flops of the exhaustive search equals $\sum_{u=1}^{p} 2^{p-u} \frac{u^2+3u}{2} = 6\,2^p - \frac{1}{2}p^2 - \frac{7}{2}p - 6$ and, as proven by Furnival and Wilson (1974), it is not possible to compute the coefficients of determination (or the residual sums of squares) for all S_a with less multiplication flops. Therefore, this procedure has a time complexity of $O(2^p)$, a result that can be generalized for any problem where an update of $C(.)$ and an memory $M(.)$ associated with $t = u-1$ different variables requires an effort of the order $O(t^2)$. In effect, it can be easily proved by induction that

$\sum_{t=0}^{p-1} 2^{p-t-1}(at^2+bt+c) = (3a+b+c)(2^p) - ap^2 - (2a+b)p - (3a+b+c)$ which implies that for a problem where an update requires at^2+bt+c flops, as p grows the average number of flops per subset approaches the constant $3a+b+c$ from below.

For criteria that never improve with the removal of variables, further computational savings can be achieved by discarding at once all the proper subsets of the sets with poor criterion values. Trying to maximize all those savings, Furnival and Wilson devised a search strategy aimed at achieving simultaneously the following objectives:

(i) Find the best subsets of each dimension in the earlier stages of the algorithm and use these sets to create bounds on the allowable values for $C(.)$. (ii) Using the bounds found in (i) discard, without further computations, as many subsets as possible. (iii) Sequence the algorithm in such a way that an update involving t different variables never needs to be performed more than 2^{p-t-1} times.

For this purpose the algorithm uses a double search tree. The first branch grows from the empty set adding single variables to previous nodes. The second branch grows the full

set removing the same variables from previous sets. After each removal the resulting criterion value is compared against the current bounds in order to test if the new sub-tree can be immediately dropped out. To make this strategy effective the effect of removing the most promising variables from the largest subsets should be tested as soon as sharp bounds are available. With that goal in view, a depth first search of the forward branch and a one-level breadth first then depth first of the backward branch are performed simultaneously with the variables sorted according to their expected impact on *C(.)*. A simple but effective way of measuring this impact is to consider the changes in *C(.)* when each variable is in turn removed form the full S_1 set[2]. Figure 1 shows a routine, in pseudo-code, that implements the full strategy when called recursively with initial arguments given by the memory with the initial data in the forward search as F_{m0} the memory with the initial data in the backward search as B_{m0}, the variable with least expected impact on *C(.)* as X_f. and the remaining variables sorted from X_2.to X_l.in decreasing order their perceived importance. Table 2 illustrates the sequence of subset evaluations for a problem with $p = 4$ variables.

Figure 1 about here

Table 2 about here

In order to understand basic properties of the algorithm and the issues to be considered in its adaptation to different problems some comments are in order.

[2] Funival and Wilson recommended that the ordering of $X_1,\ldots, X_p$ should be updated periodically in function of the subsets being considered at each step. However, in my experience, for exact algorithms this potentially expensive procedure does not seem to lead to any noticeable improvement.

Firstly, nothing in Figure 1 contains anything specifically related to linear regression and the whole procedure can be implemented in generic form, given some function(s) capable of updating the criteria and other information associated with each variable subset. That is the approach followed by the Extended Leaps library, described in the following section, which relays on the mechanism of C++ abstract classes to separate the basic algorithm from the details concerning particular statistical problems. Different types of analysis and criteria can be easily included by deriving concrete classes that implement the specifics of each criteria update.

Secondly, the success of the Leaps algorithm depends critically on its two most distinctive features, one being the branch and bound strategy of pruning large parcels of the search tree after evaluating a few subsets, and the other the efficient sequencing scheme that allows each update to work with a small amount of data. To take full advantage of the algorithm both features should be present, but if for some problem or criteria one of them fails the algorithm can still take advantage of the other. For instances, the branch and bound strategy only requires a common monotonic property that can be applied regardless of the way the updating is performed. Furthermore, the sequencing of the algorithm is designed to combine the computational savings achieved by keeping t as low as possible, with an attempt to maximize the impact of pruning, by trying to find the best subsets of the largest dimensionalities as early as possible. This strategy works the best if, as originally intended, the algorithm proceeds until all subsets are explicitly or implicitly considered. However, since the subsets of lower dimensionalities are typically fully explored only in the latter stages of the search, when leaps is used as an heuristic being stopped after a reasonable amount of time, the original sequencing has an undesirable tendency of giving poor results for the smaller subsets. A more effective strategy in this case is to reverse the order of the

variables and the strategies used for the forward and backward branches of the search, using a breath first then depth approach in the forward branch and a depth first approach in the backward branch. A routine that implements this strategy is presented and illustrated in Figure 2 and Table 3.

Figure 2 about here

Table 3 about here

Thirdly, even for non-monotone criteria the computational savings obtained by the sequencing scheme can still be substantial. Furthermore if the branch and bound plan is dropped, a full two-way search does not have to be applied since the much simpler pure forward sequence illustrated in Table 1 will have similar performance. For some problems even if the comparison is monotone it may be preferable to give up branch and bound in order to guarantee numerical stability. That will be the case of a linear regression model with strong multicolinearitty in the full data set but not in important subsets that can be identified by a forward search. It should be noticed that even the case of perfect multicolinearity in the full set, for example if there are more original variables than observations, can be handled without difficulties by this approach. However, when the algorithm is used as an heuristic, Furnival's scheme may fail to identify reasonable subsets in the lower dimensionalities. Correcting this problem with pure forward search that achieve Furnival's minimal computational effort does not seem as straightforward as when a double search is employed. Specific strategies to address this problem are currently under investigation.

3. Extended Leaps Implementations

The Extended Leaps library is composed of a collection of C++ classes and functions that implement the basic leaps and bounds algorithms without making specific assumptions about the criteria used in the subset comparisons. At the heart of the library there are three functions: *Forward_Search()*, *Leaps_Search()* and *Rev_Leaps_Search()* that implement the search routines described in the previous section. These functions take as an argument an abstract class, named *subsetdata*, whose concrete specializations should specify the details concerning criteria updates for different problems and comparison criteria. In particular, specializations of the abstract member function of *subsetdata* named *updateC()* should implement criteria updates when single variables are added to or removed from a given subset. Likewise, specializations of *updateM()* should specify how to update the information required to evaluate further subsets. Additional abstract classes provide the basic structure for storing the data that may be required by *updateC()* and *updateM()*.

A boolean argument (passed by reference) of *updateC()* and *updateM()*, named *reliable*, is provided in order to report if an estimate of the relative rounding errors remain below user-specified limits. To facilitate such a reliability check, the library includes a class named *errmonitreal* that keeps track of a first order estimate of an upper bound on the relative error of each computed real number according to:

$$fl(\hat{x} \pm \hat{y}) \approx [x(1+\delta_X) \pm y(1+\delta_Y)](1+\eta) \approx (x \pm y)\left(1+\frac{|x|\delta_X + |y|\delta_Y}{\hat{x} \pm \hat{y}} + \eta\right)$$

$$fl(\hat{x} * \hat{y}) \approx x(1+\delta_X) * y(1+\delta_Y)(1+\eta) \approx (x * y)(1+\delta_X+\delta_Y+\eta)$$

$$fl(\hat{x} / \hat{y}) \approx [x(1+\delta_X) / y(1+\delta_Y)](1+\eta) \approx (x / y)(1+\delta_X+\delta_Y+\eta)$$

Here, η represents the machine unit roundoff and δ_X , δ_Y the relative rounding errors of the computed real numbers $\hat{x}$ and $\hat{y}$.

In order to use the library in the evaluation of variable subsets according to a new criterion, the programmer should first identify the data requirements for each update and specialize all the abstract classes accordingly. Then, she/he should provide a mechanism for reading the data and placing it in the appropriate data structures. Finally, the programmer should specify how to choose a particular search function and call the chosen function with the new specialization of *subsetdata* as an argument. For monotone criteria the search function will typically be *Leaps_Search()* or *Rev_Leaps_Search()*, with the former recommended when proven optimality is to be expected within a reasonable time an the later when the algorithm is to be used as an heuristic. However, when the comparison criterion can not be reliably computed in the full data set, or it is not monotone, the *Forward_Search()* routine should be employed.

Two important specializations of *subsetdata* are provided within the library. The first one *singleqfdata* evaluates criteria of the form $C_1(S_a) = v_{S_a}{}^T M_{S_a}^{-1} v_{S_a}$ where v_{S_a} and M_{S_a} are a vector and a symmetric matrix that depend only on the variables included in S_a. Particular cases of the $C_1(S_a)$ criterion include the coefficient of determination in linear regression, the sample Mahalanobis distance between group centroids in Discriminant Analysis problems (see Duarte Silva 1988, 2001) and the value of Wald statistic when testing the relevance of excluded variables in generalized linear models (see Lawless and Singhal 1978, 1987a). The second *subsetdata* specialization, named *wilksdata*, evaluates criteria of the form $C_2(S_a) = |E_{S_a}| / |T_{S_a}|$ where $|E_{S_a}|$ and $|T_{S_a}|$ are the determinants of two symmetric matrices that depend only on S_a. A criterion of this form is the value of

Wilks statistic for testing linear hypothesis in multivariate Gaussian linear models, a criterion suggested by McCabe (1975) in the context of Discriminant Analysis. Other specializations of *subsetdata* are provided outside the library with the console application that will be briefly described in following paragraphs.

In the source code that accompanies this paper it is included the implementation of a console application, named *ELeapsCons*, that uses the *Extended Leaps* library to search for variable subsets in the context of Linear Regression, Multivariate Linear Models (including models for Canonical Correlation, Linear Discriminant Analysis and the analysis of MANOVA/MANCOVA effects) and exploratory Multivariate Analysis. It is expected that future versions of *ELeapsCons* will also be able to address subset comparisons in Generalized Linear Models. Being a console application, *ELeapsCons* handles its input and output through text files. Command line arguments are used to specify several options such as the number of different subsets kept per dimensionality, the comparison criterion chosen, the minimal and maximal dimensionality searched, the time allowed for the search, etc. All these options come with default values needing to be invoked only when the user wants to override them.

The original data processing of *ELeapsCons* resorts to several matrix operations and decompositions (spectral, singular value, ….) that should be provided by an external matrix system. In particular, the source code of *ELeapsCons* assumes that all the functions classes and operations defined in the file *MatrixOp.h* are available. The sintax of *MatrixOp.h* is based on an open-source matrix system, *NewMat*, developed by Robert Davies (1994) and publicly available at http://www.robertnz.net. The simplest way to access the functionality specified in *MatrixOp.h* is to precede the compilation of *ELeapsCons* by the installation of the *NewMat* system. Alternatively what can use its favourite Matrix system, such has

LINPACK, EISPACK or LAPACK, as long as an interface that translates the functions defined in such system into the syntax specified in *MatrixOp.h* is previously developed. The Extended Leaps library itself does not relay on any specialized matrix operations and can be compiled on its own.

A previous version of the Extended Leaps library is the basis of the exact search routines of an *subselect R* package, named *subselect*, which addresses the same problems and uses the same criteria as *ELeapsCons*. However, the exact searches of *subselect* employ only on the original Leaps and Bound ordering scheme and thus are recommended only for problems with well conditioned data where proven optimality can be achieved within a reasonable amount of time. Larger problems are addressed in *subselect* by heuristic routines based on simulated annealing, genetic and restricted search algorithms that were developed and implemented by Cadima, Cerdeira and Minhoto (2004). Possible problems with perfect or almost perfect collinear data are always checked before hand in *subselect*. Furthermore, this package provides some pre-processing facilities to eliminate redundant variables until multicolinearity is no longer a relevant problem.

4. Conclusions and Directions for Further Research

Choosing one or several subsets from a large pool of candidate variables is one of oldest, most recurrent but still not yet fully resolved problems in many statistical and data analysis methodologies. The classical approach to this problem is to resort to some variant of, one-variable at the time, stepwise algorithms. Such algorithms although being subject to many well known shortcomings (see Miller 2002 and Huberty and Olejnik 2006 for discussions) remain popular amongst practitioners.

In linear regression, algorithms capable of comparing all possible variable subsets for problems with a moderate number of variables are known since de ninety seventies. One of the best known of these algorithms is the classical Leaps and Bounds of Furnival and Wilson. Leaps and Bound is best know for variable selection in the context of Linear Regression. However, its basic ideas can be easily adapted to a wide range of statistical problems and data analysis methodologies. Furthermore, although Leaps and Bounds was originally devised as an exact algorithm for problems with a moderate number of variables, it can be adapted in larger problems to work as an heuristic specifically designed for the problem of variable selection. Finally, problems with strong or perfect multicolinearity in the full data set can still be analysed by Leaps and Bounds adaptations that sequence the search in order avoids numerical problems. In particular, Leaps and Bounds routines are not restricted to problems with more observations than variables.

This paper and its companion software make some important steps in the directions described above. By clearly separating all the specific algorithmic details from the particularities of each criterion and data analysis methodology, adaptations to different types of analysis are enhanced. The availability of different variants and sequencing schemes that respect the most distinctive Leaps and Bounds features but are designed for conditions where the original algorithm was unreliable or less effective, increases the range of problems that can be tackled by this approach.

Much remains to be done. Firstly, the Reversed Leaps routine is just a first step in the development of search schemes that have good performance in problems where an exact search is not viable. It is particularly important to search for schemes that have a good performance when the algorithm is used as an heuristic. One possibility is to use mixed strategies that combine pure forward passes with double searches in specific parcels of the

search tree. Furthermore, while the use of frequent variable sorting, as initially suggested by Furnival and Wilson, does not seem to lead to any noticeable improvements in exact searches it may improve the results of heuristic versions of the algorithm. The performance of these strategies needs to be compared against that of the Reversed Leaps routine and other heuristics.

Secondly, the problems for which Leaps and Bounds adaptations are currently available reflect to some extent the particular interests of the researchers that have been most active in this area. However, there are many other statistical problems that require variable selection and subset comparisons methodologies that may benefit form similar adaptations. Specializations of the classes and data structures presented in this paper will allow such adaptations to benefit from the current and future algorithmic developments of the ELeaps library.

Finally, this paper has focused on the algorithmic optimization of comparison criteria and has ignored the related issues of bias selection and post-selection inference. It is known that data-based model selection invalidates some of the assumptions in which classical inference methodologies rely on and that traditional goodness of fit statistics may become too optimistic. Methods to assess and correct to effects of selection bias should be employed whenever a data driven variable selection methodology is applied. Resampling or cross-validation strategies that repeated the search for adequate subsets for each newly generated data (see, e.g., Le Roux, Stell and Louw 1997), are a relatively simple way to deal with the most serious consequences of this problem .

References

Cadima, J. Cerdeira, J.O. and Minhoto, M. 2004. Computational aspects of algorithms for variable selection in the context of principal components. *Computational Statistics and Data Analysis*, **47**, 225-236.

Davies, R.B. 1994. Writing a matrix package in C++. In OON-SKI'94: The Second Annual Object-Oriented Numerics Conference, 207-213. Rogue Wave Software, Corvallis.

Duarte Silva, A.P., 1998. A leaps and bounds algorithm for variable selection in two-group discriminant analysis *in* "Advances in Data Science and Classification" (A. Rizzi, M. Vichi, and H. Bock, Ed.) IFCS Springer, 227-232.

Duarte Silva, A.P. 2002. Discarding variables in a principal component analysis: algorithms for all-subsets comparisons. *Computational Statistics*, **17**, 251-271.

Duarte Silva, A.P. 2001. Efficient Variable Screening for Multivariate Analysis. *Journal of Multivariate Analysis*, **76** (1), 35-62

Furnival, G.M. 1971. All possible regressions with less computation. *Technometrics*. **13**, 403-408.

Furnival, G.M. and Wilson, R.W. 1974. Regressions by leaps and bounds. *Technometrics*. **16**, 499-511.

Gatu, C. and Kontoghiorghes, C. 2006. Branch-and-Bound Algorithms for Computing the Best-Subset Regression Models. *Journal of Computational and Graphical Statistics*. **15** (1), 139-156.

Hosmer, D.W. Jovanovic, B. and Lemeshow, S. 1989. Best subsets logistic regression. *Biometrics*. **45**, 1265-21270.

Huberty, C.J. and Olejnik, S. 2006. *Applied MANOVA and Discriminant Analysis. 2nd Edition. John Wiley*. Hoboken, NJ.

Kim, S-S. 2000. All possible subset regressions using the triangular decomposition. *Journal of Statistical Computation and Simulation* **65**, 81-94.

Lawless, J.F. and Singhal, K. 1978. Efficient screening of nonnormal regression models. *Biometrics* **34**, 318-327.

Lawless, J.F. and Singhal, K. 1987a. ISMOD: An all-subsets regression program for generalized models. I. Statistical and computational background. *Computer Methods and Programs in Biomedicine* **24**, 117-124.

Lawless, J.F. and Singhal, K. 1987b. ISMOD: An all-subsets regression program for generalized models. II. Program guide and examples. *Computer Methods and Programs in Biomedicine* **24**, 125-134.

Le Roux, N.J. Stell, S.J. and Low, N. 1997. Variable selection and error rate estimation in discriminant analysis. *Journal of Statistical Computing and Simulation. **59**, 195-219.*

Miller, A.J. 2002. "Subset Selection in Regression", 2nd Edition. Chapman and Hall.

McCabe, G.P. 1975. Computations for Variable Selection in Discriminant Analysis. *Technometrics*. **17**, 103-109.

R Development Core Team. 2008. R: A language and environment for statistical computing. R Fondation for statistical computing. Vienna, Austria. ISBN 3-900051-070, URL http://www.R-project.com

Smith, D.M. and Bremner, J.M. 1989. All possible subset regressions using the QR decomposition. *Computational Statistics and Data Analysis* **7**, 217-235.

Winker, P., Gilli, M. 2004. Application of optimization heuristics to estimation and modelling problems, *Computational Statistics and Data Analysis*. **47**, 211-223.

Table 1

Furnival sequence for an exhaustive forward search ($p = 3$)

Pivot	Original Subset (S_a)	New Subset (S_b)	Pivot Variable (X_u)		Intermediate Results	
				t	Retrieve from Memory	Save in Memory
1	ϕ	$\{X_1\}$	X_1	0	M(3)	None
2	ϕ	$\{X_2\}$	X_2	1	M (3)	M (1)
3	$\{X_2\}$	$\{X_1, X_2\}$	X_1	0	M (1)	None
4	ϕ	$\{X_3\}$	X_3	2	M (3)	M (2)
5	$\{X_3\}$	$\{X_1, X_3\}$	X_1	0	M (2)	None
6	$\{X_3\}$	$\{X_2, X_3\}$	X_2	1	M (2)	M (1)
7	$\{X_2, X_3\}$	$\{X_1, X_2, X_3\}$	X_1	0	M (1)	None

Figure 1 Leaps and Bounds – Original Routine

```
Leaps_Search(Initial Memories F_m0 , B_m0, First Variable X_f , Last Variable X_l)
{
  Let F_m = F_m0
  For (i = f to l – 1)  {
      Get S_F(i-f) and S_B(0) intermediate results from memories F_m and B_m0
      Let u = l – i + 1
      Pivot S_F(i-f) and S_B(0) onto S_F(i-f+1) and S_B(i-f+1) by sweeping on variable X_u
      Update Current_Bounds
      If  (i < l - 1)  {
          Save S_F(i-f+1) and S_B(i-f+1) intermediate results in memories F_u–2  and B_u– 2
          Let  Prev_FMem(u-2) = F_m
          Let F_m = F_u – 2
       }
    }
    If (l –f  > 1) For (i = 1 to l – f - 1) {
         Get S_B(l-f-i) intermediate results from memory B_i
         If (S_B(l-f-i) is better than Current_Bounds) Leaps_Search(Prev_FMem(i) ,B_i, X_f, X_i+1)
   }
}
```

Table 2

Sequence for an original leaps search (p = 4)

Pivots	X_u	t	Forward Search		Backward Search	
			S_a	S_b	S_c	S_d
1-2	X_4	2	ϕ	$\{X_4\}$	$\{X_1,X_2,X_3,X_4\}$	$\{X_1,X_2,X_3\}$
3-4	X_3	1	$\{X_4\}$	$\{X_3, X_4\}$	$\{X_1,X_2,X_3,X_4\}$	$\{X_1,X_2,X_4\}$
5-6	X_2	0	$\{X_3, X_4\}$	$\{X_2, X_3, X_4\}$	$\{X_1,X_2,X_3,X_4\}$	$\{X_1,X_3,X_4\}$
7-8	X_2	0	$\{X_4\}$	$\{X_2, X_4\}$	$\{X_1,X_2,X_4\}$	$\{X_1,X_4\}$
9-10	X_3	1	ϕ	$\{X_3\}$	$\{X_1, X_2, X_3\}$	$\{X_1, X_2\}$
11-12	X_2	0	$\{X_3\}$	$\{X_2, X_3\}$	$\{X_1, X_2, X_3\}$	$\{X_1, X_3\}$
13-14	X_2	0	ϕ	$\{X_2\}$	$\{X_1, X_2\}$	$\{X_1\}$

Figure 2 Reversed Leaps and Bounds Routine

```
Rev_Leaps_Search(Initial Memories F_m0 , B_m0, First Variable X_f , Last Variable X_l)
{
  Let B_m = B_m0
  For (i = f to l – 1)  {
      Get S_F(0) and S_B(i-f) intermediate results from memories F_m0 and B_m
      Pivot S_F(0) and S_B(i-f) onto S_F(i-f+1) and S_B(i-f+1) by sweeping on variable X_i
      Update Current_Bounds
      If  (i < l - 1)  {
          Save S_F(i-f+1) and S_B(i-f+1) intermediate results in memories F_l-i-1 and B_l-i-1
          Let  Prev_BMem(l – i – 1) = B_m
          Let B_m = B_l-i-1
       }
    }
    If (l –f  > 1) For (i = 1 to l – f - 1) {
         Get S_B(l-f-i-1) intermediate results from memory Prev_BMem(i)
         If (S_B(l-f-i-1) is better than Current_Bounds)
                Rev_Leaps_Search(F_i, Prev_BMem(i) , X_l-i , X_1 )
   }
}
```

Table 3

Sequence for a reverse leaps search (p = 4)

Pivots	X_u	t	**Forward Search**		**Backward Search**	
			S_a	**S_b**	**S_c**	**S_d**
1-2	X_1	2	ϕ	$\{X_1\}$	$\{X_1,X_2,X_3,X_4\}$	$\{X_2,X_3,X_4\}$
3-4	X_2	1	ϕ	$\{X_2\}$	$\{ X_2,X_3,X_4\}$	$\{ X_3,X_4\}$
5-6	X_3	0	ϕ	$\{X_3\}$	$\{ X_3,X_4\}$	$\{ X_4\}$
7-8	X_3	0	$\{X_2\}$	$\{X_2, X_3\}$	$\{X_2,X_3,X_4\}$	$\{X_2,X_4\}$
9-10	X_2	1	$\{X_1\}$	$\{X_1, X_2\}$	$\{X_1,X_2,X_3,X_4\}$	$\{X_1,X_3,X_4\}$
11-12	X_3	0	$\{X_1\}$	$\{X_1, X_3\}$	$\{X_1, X_3, X_4\}$	$\{X_1, X_4\}$
13-14	X_3	0	$\{X_1, X_2\}$	$\{X_1, X_2, X_3\}$	$\{X_1,X_2,X_3,X_4\}$	$\{X_1,X_2,X_4\}$