Enabling End-User Datawarehouse Mining
Contract No. IST-1999-11993
Deliverable No. D14.4

Feature Selection Operators based on Information Theoretical Measures

Radim Jiroušek[1], Pavel Pudil[1], Petr Berka[2]

[1] Institute of Information Theory and Automation,
Academy of Sciences of the Czech Republic
CZ-180 02 Prague, Czech Republic
radim@utia.cas.cz, pudil@utia.cas.cz

[2] University of Economics
Laboratory for Intelligent Systems
CZ-13065 Prague, Czech Republic
berka@vse.cz

1. February 2003

1 Introduction

The idea of feature selection is to find attributes most relevant for the subsequent classification task. These attributes can be specified by the user according to his background knowlede and experience or can be selected using some criterion that reflects the relationship between the (input) attributes and the class attribute. For the later type of feature selection, different criteria can be found in the literature (statistical tests, entropy, etc) . A wrapper approach can also be used - in this case different subsets of attributes are evaluated according to their classification accuracy. Another difference between feature selection algortihms can be the search strategy (forward by adding attributes or backward by removing attributes) nad the number of processed attributes (univariate algorithms simply evaluate each candidate attribute independently, multivariate algorithms evaluate a set of attributes a time). In our report we discuss only the information-theoretic measures of dependence used for multivariate feature selection.

This report is not a theoretical discourse on pattern recognition. The report deals exclusively with description of one important step of pattern recognition process, namely with feature selection, which is one of two forms of feature reduction.

The primary purpose of reducing the dimensionality of the original measurements (attributes) stems from engineering considerations. First of all, by minimizing the number of pattern descriptors the cost of hardware implementation of the data acquisition system (sensors, A/D converters, data processing system) can be substantially reduced. Furthermore, the complexity of hardware implementation of a classifier grows rapidly with the number of dimensions of the pattern space. It is therefore important to base decisions only on the most essential, so-called discriminatory information, which is conveyed by features. This, in turn, will effectively minimize the storage capacity, cost and time needed for a pattern recognition system.

Apart from such engineering constraints, dimensionality reduction may prove beneficial from the classification performance point of view. This claim may seem somewhat controversial, since from the standpoint of Bayesian decision rules there are no bad features. One cannot improve the performance of a Bayes classifier by eliminating a feature (this property is called *monotonicity*). However, the use of Bayesian classifiers in practice is limited by the fact that, in general, probability densities are not known and must be estimated from a set of finite (usually small) number of training samples. If the number of training samples is small, problems are commonly manifested due to the so-called *peaking phenomenon* [1, 7, 16]. This phenomenon concerns the dependence of the probability of correct recognition of patterns outside the training set and the number of features used. Initially the performance improves as new features are added, but at some point inclusion of further features may result in an actual degradation in performance. As a consequence, it is possible to improve the accuracy of the classifier's performance by deleting a feature [4, 15].

2 Problem formulation

In this report, data-driven approach to this problem is described. The report can thus serve as a basis for development of feature selection system. This means that all problems connected with implementation of feature selection task are mentioned but not for all of them a solution is described. It is because solution of some problems is necessarily closely connected with the way of implementation (e.g. data structures used), which is not discussed in this report.

Throughout the whole text **data** is assumed to be a set of vectors; coordinates of these vectors correspond to possible attributes of objects, which belong to several predefined classes. Since *supervised* learning is assumed, we suppose that coordinates of the data vectors can be split into two parts: *features*, i.e., attributes which are observable during the pattern recognition process and *descriptors* defining the pattern (class) of the corresponding object. So e.g. in diagnosis-making process features correspond to signs and symptoms whereas descriptors define the diagnosis. Therefore, a usual **data** vectors will consist of a large number of features and one descriptor.

A pattern recognition problem begins with class definitions, which are described by descriptors. More precisely, patterns are assumed to be defined by specific combinations of their values.

Before we proceed any further, it will be very useful to fix a (little bit simplified) notation. Since all combinations of values defining distinguished classes (patterns) must be defined, we can define a single descriptor y with as many values as the number of considered patterns (classes). Therefore we will assume that the considered **data** are given in a form

$$\begin{bmatrix} x_{11}, x_{12}, x_{13}, \ldots, x_{1s}, & y_1 \\ x_{21}, x_{22}, x_{23}, \ldots, x_{2s}, & y_2 \\ x_{31}, x_{32}, x_{33}, \ldots, x_{3s}, & y_3 \\ \vdots & \\ x_{n1}, x_{n2}, x_{n3}, \ldots, x_{ns}, & y_n \end{bmatrix}$$

and each value y_k belongs to one of possible pattern classes.

Notice, that from some or another reason not all s features must always be considered for selection. Without loss of generality we will assume that a set of candidate features is $\xi_j, j = 1, 2, \cdots, d$ for $d < s$. Observed values of the feature ξ_j for kth object is x_{kj} and the object is assigned to the class y_k, which is a value of a descriptor η.

The goal of the feature selection process is to find, in a way optimal, subset $J \subset \{1, \ldots, d\}$ such that values of the descriptor η can be sufficiently well determined from values of features $\{\xi_j\}_{j \in J}$. In ideal case it would mean that there exists a function f:

$$\eta = f((\xi_j)_{j \in J}).$$

In case that **data** do not contain errors, a necessary condition (but far from sufficient) for this function to be "correct" is that for all n objects (i.e. for $k = 1, \ldots, n$)

$$y_k = f((x_{kj})_{j \in J}).$$

However, in practical situation we are usually pretty far from this ideal case. Partly, because such a deterministic function usually does not exist, partly, because we cannot rely upon reliability of data. In correspondence with this, solution of the feature selection problem depends on how we can cope with the following two problems:

- What does it mean that values of the descriptor η can be sufficiently well determined from values of features?
- How to find the best subset of features complying with the previous condition?

3 Information-theoretic measures of dependence

In literaure, several different optimality criteria for feature selection were used. In the current report we will consider only one of them, which, although manifesting both theoretical and practical good characteristics, is often wrongfully omitted. It is the criterion based on a concept of *mutual information.*

Assume for a moment that a probability distribution $P(\xi_1, \ldots, \xi_d, \eta)$ describing dependence of features and descriptor is known. Mutual information describing strength of dependence of a descriptor upon features is defined by the following formula

$$MI((\xi_1, \ldots, \xi_d); \eta) = \sum_{(x_1, \ldots, x_d)} \sum_{y} P(\xi_1 = x_1, \ldots, \xi_d = x_d, \eta = y) \log \frac{P(\xi_1 = x_1, \ldots, \xi_d = x_d, \eta = y)}{P(\xi_1 = x_1, \ldots, \xi_d = x_d) \cdot P(\eta = y)},$$

where the summation is over all combinations of values $(\xi_1, \ldots, \xi_d)$ and all the considered classes (patterns) y (values of η). It is known (see e.g. [6]) that

$$0 \leq MI((\xi_1, \ldots, \xi_d); \eta) \leq \min \Big(H(P(\xi_1, \ldots, \xi_d)); H(P(\eta)) \Big),$$

where $H(P(\cdot))$ denotes the respective Shannon entropy

$$H(P(\xi)) = -\sum_{x} P(x) \log P(x).$$

Moreover, it is known that

$$MI((\xi_1, \ldots, \xi_d); \eta) = 0$$

if and only if pattern η are independent of features $\xi_1, \ldots, \xi_d$, and

$$MI((\xi_1, \ldots, \xi_d); \eta) = \min\Big(H(P(\xi_1, \ldots, \xi_d)); H(P(\eta))\Big)$$

if and only if features and pattern are functionally dependent. Since, as a rule[1] $H(P(\xi_1, \ldots, \xi_d)) \geq H(P(\eta))$, one can normalize the mutual information by entropy of patterns

$$ID((\xi_1, \ldots, \xi_d); \eta) = \frac{MI((\xi_1, \ldots, \xi_d); \eta)}{H(P(\eta))}$$

getting a *information-measure of dependence* introduced by Perez. This measure equals 0 if and only if η is independent of features $\xi_1, \ldots, \xi_d$, and equals 1 if and only if there exists a (deterministic) function

$$\eta = f(\xi_1, \ldots, \xi_d).$$

Therefore, the value of information-measure of dependence can serve as a basis for *stopping rule* for sequential feature selection procedures. Such stopping rules can easily be proposed particularly when a heuristic rule based on a vast number of experiments is taken into consideration: *if $ID((\xi_1, \ldots, \xi_d); \eta)$ is close to 1 then the Bayesian error of pattern recognition process (when η is determined on the basis of values of $\xi_1, \ldots, \xi_d$) is well approximated by $1 - ID((\xi_1, \ldots, \xi_d); \eta)$.* The closer ID to 1 the better this approximation - for values $ID > .95$ the approximations are pretty good.

3.1 Comments on computation of information-measure of dependence

Each feature selection algorithm has to compute the criterion function – in our case the information-measure of dependence $ID((\xi_j)_{j\in J}; \eta)$ – for a great number of subsets $J \subseteq \{1, \ldots, d\}$. The faster the value is computed the greater part of the subset space is searched and thus, as a rule, better results are achieved. Therefore, speeding up computations of the criterion function is a crucial condition for a success. Moreover, if there is a possibility to compute several these values (for different subsets $J_1, \ldots J_\ell$) simultaneously, it must be exploited whenever it is possible.

Basicly, there are two ways to compute information-measure of dependence

$$\begin{aligned} ID((\xi_j)_{j\in J}; \eta) &= \frac{MI((\xi_j)_{j\in J}; \eta)}{H(P(\eta))} \\ &= \frac{1}{H(P(\eta))} \sum_{((x_j)_{j\in J})} \sum_{y} P((\xi_j = x_j)_{j\in J}, \eta = y) \log \frac{P((\xi_j = x_j)_{j\in J}, \eta = y)}{P((\xi_j = x_j)_{j\in J}) \cdot P(\eta = y)}. \end{aligned}$$

One can proceed directly according to this formula only in situations when a small number $|J|$ of features is considered. If this is the case, one can reserve a register for each combination of values $((x_j)_{j\in J}, y)$ and going through **data** once, calculate how many times each combination of values occures. Then these frequencies serve for computation of relative frequencies (or some other reasonable statistics) that are taken for estimates of the respective probabilities.

However, if J is too large to perform this type of computation (what is large depend on how many different values each feature has, and what is the size of **data** n), one has to take advantage of the fact that the size of **data** is limited. In this case one should realize that it is not necessary to consider all the combinations of values $((x_j)_{j\in J}, y)$ (which would be, for example, for $|J| = 100$ impossible) but it is enough to consider only the combinations appearing in **data**. In this way one, in fact, realizes the formula

$$ID((\xi_j)_{j\in J}; \eta) = \frac{1}{H(P(\eta))} \sum_{((x_j)_{j\in J}, y): P((x_j)_{j\in J}, y) > 0} P((\xi_j = x_j)_{j\in J}, \eta = y) \log \frac{P((\xi_j = x_j)_{j\in J}, \eta = y)}{P((\xi_j = x_j)_{j\in J}) \cdot P(\eta = y)}.$$

This type of computation can easily be achieved by sorting the vectors of **data** according to the values of selected features, which makes it possible to find all the combinations and their frequencies when going through **data** again only once.

It is perhaps not necessary to remark that both the formulae are equivalent since $0 \log 0 = 0$ and that computation of the normalizing constant $\frac{1}{H(P(\eta))}$ can be performed only once.

[1] One can construct an artificial counterexample, but in practical situations it cannot happen that $H(P(\xi_1, \ldots, \xi_d)) < H(P(\eta))$.

4 Basic principles of subset space search methods

Now we start discussing the most important issue of the whole feature selection process. Cover [2] has shown that in order to guarantee finding of an optimal subset $(\xi_j)_{j\in J}$ of features, exhaustive search is a necessary procedure. However, in almost all practical cases the value of d result in the number of possible subsets that are too large, i.e. the search is too long, the required memory is too large, or both. For example, in a typical texture classification problem, we might be trying to select 10 features out of say 60 available attributes which would require evaluation of more than 7.54×10^{10} feature sets. Obviously, exhaustive search seems to be prohibitive in such an application. Therefore, some computationally feasible procedures to avoid the exhaustive search are essential even though the feature set obtained may be suboptimal. For the above reason, the question of the trade-off between the optimality and efficiency of algorithms for feature selection is recognized, and the mainstream of research on feature selection has thus been directed toward suboptimal search methods. Cover [2] has also pointed out the potential of any non-exhaustive procedure to select the worst possible set of features. This fact prevents any claim of optimality for any non-exhaustive method. All that can be claimed is that certain methods are better than others. It should be noted that the search strategy for feature selection is independent of the criterion function used.

4.1 Classification of search strategies

The search strategies can be basically classified in two directions:

1. according to the search result achieved:
   - *optimal search procedures* - they are able to find an optimal solution (however, they are either computationally not feasible or applicable under certain restrictive conditions only).
   - *suboptimal search procedures* (the optimality of the result cannot be guaranteed but usually they yield a solution either optimal or very close to the optimal)
2. according to the way by which possible candidate feature sets are searched and the final subset is constructed - either by repeated adding (or by removing) of individual features (or small groups of features) in the respective steps until the feature subset having the size m with required properties is achieved.
   - *"forward" strategies* (also called "bottom up"). A usual procedure (in case of suboptimal methods) is to construct the subset of m features from an empty initial set by sequential adding of features.
   - *"backward" strategies* (also called "top down"). A usual procedure (in case of suboptimal methods) is to construct the subset of m features from the initial set of all the attributes by sequential removing of features.

   Remark. There are strategies where both the "adding" and "removing" steps are combined, however, one of the directions is always prevailing.

4.2 Comparison of forward and backward methods from theoretical and practical viewpoints

When deciding which of the two possible directions of search to use, one must be aware of their differences both from the theoretical and practical point of view. The differences are particularly significant in the case of simple methods (SFS, SBS) (see the next section). At the initial stages of algorithms, the decision about including a particular feature at the first few steps of the sequential forward algorithms is made on the basis of only low order statistical dependencies of original pattern components (only subsets of low cardinality are considered).

This drawback does not concern the backward sequential selection methods. The decision on excluding a particular feature is from the beginning made by taking into consideration the statistical dependencies of original pattern components in full dimensionality (subsets of high cardinality are considered),

The just stated facts seem to prefer uniquely the backward methods. However, the problem is somewhat more complicated. In practice we are restricted to training sets of limited size, which is often not adequate to the problem dimensionality. In this case obviously the training set is not sufficiently representative with respect to the basic data set. From it a hidden drawback of the backward methods follows, namely that though in comparison with the forward methods they take better into consideration complex mutual relations, these relations are not always determined quite reliably. On the other hand, though the forward methods utilize at the beginning only less complex mutual relations, they are, however, determined more reliably even for training sets of a moderate size.

From the computational point of view there are some basic differences, too, since the numerical properties of the forward and backward methods are different. Since the backward methods start at full dimension d, it may happen that due to dependencies of data components, the procedure to calculate the evaluation criterion numerically fails. Thus from purely numerical reasons, to begin with the forward methods is preferable in those cases when deterministic (or very strong stochastic) dependences between the data components can be expected.

The differences between more sophisticated forward and backward methods (particularly SFFS and SBFS) methods are basically the same as those between the straight methods. However, the arguments in favour or against are not so strong in this case, since due to the "floating" principle both the methods have the chance to correct decisions made wrongly in previous steps. In practice both the methods give quite often, though not always, the identical results. Nevertheless, the numerical considerations discussed in the previous paragraph hold even in this case.

5 Suboptimal search procedures

If omitting the exhaustive search, which is in practical situations intractable, the only optimal procedure used for feature selection is the Branch & Bound algorithm proposed first for this purpose in 1977 by Narendra and Fukunaga [10]. Their version is a “top-down” algorithm with backtracking. It is based on the assumption that the adopted criterion function fulfills a special monotonicity condition, which is, however, not met by the considered information-measure of dependence. Therefore, we will not discuss optimal procedures in this report. Before starting to discuss suboptimal procedures, let us introduce a simplified notation of a criterion function, which will be used in the rest of the report:

$$ID((\xi_j)_{j\in J};\eta) = C(\{\xi_j\}_{j\in J}).$$

In all the feature set search algorithms to be discussed in this and next sections, the best feature set is constructed by consecutive adding to and/or removing from a working group of feature set, a small number of attributes at a time until the required feature set, $\{\xi_j\}_{j\in J}$, of cardinality m is obtained. More specifically, to form the best m-feature set the starting point of the search can be either an empty set, which is then successively built up or, alternatively, the starting point can be the complete set of attributes, $\{\xi_j\}_{j=1,\dots,d}$, from which superfluous attributes are successively eliminated.

In order to describe the various algorithms, let χ_k be a set containing k elements, from the complete set of available attributes, $\{\xi_j\}_{j=1,\dots,d}$, i.e.

$$\chi_k \subseteq \{\xi_j\}_{j=1,2,\dots,d}, \quad |\chi_k| = k.$$

Further denote by $\bar{\chi}_k$ the set of d-k features obtained by removing k attriubtes, from the complete set of attributes, i.e.

$$\bar{\chi}_k \subseteq \{\xi_j\}_{j=1,\dots,d} \quad |\chi_k| = d-k.$$

Obviously, different sets χ_k and $\bar{\chi}_k$ are obtained by choosing different attributes ξ_i. In addition, the following symbols will be used:

$$\chi_0 \equiv \emptyset$$
$$\bar{\chi}_0 \equiv \{\xi_j\}_{j=1,\dots,d}$$

All bottom up and top down search algorithms are based on the monotonicity property of the used feature selection criterion function. It says that for nested feature sets $\chi_1, \chi_2, \cdots, \chi_\ell$, i.e.

$$\chi_1 \subset \chi_2 \subset \cdots \subset \chi_\ell$$

the criterion function C satisfies

$$C(\chi_1) \leq C(\chi_2) \leq \cdots \leq C(\chi_\ell).$$

5.1 Fixed search strategies

It is well known that the set of d individually best features is *not* necessarily the best feature set of size d even for the case of statistical independent features [3]. Better results can be achieved by the following two simple algorithms based on "greedy" startegy.

5.1.1 Sequential forward selection

The sequential forward selection (SFS) algorithm is a simple bottom up search procedure where one feature at a time is added to the current feature set. The algorithm starts from an empty set, and the individually best attribute is selected as the first feature. At each subsequent stage the candidate feature to be included in the set is selected from among the remaining available attributes, so that in combination with the features already selected it yields the best value of the criterion function.

The Sequential Forward Selection Algorithm

Suppose k features have already been selected from the complete set of attributes $\{\xi_j\}_{j=1,\ldots,d}$ to form feature set χ_k. The $(k+1)^{st}$ feature is then chosen from the set of available attributes, $\{\xi_j\}_{j=1,\ldots,d} \setminus \chi_k$, so that

$$C(X_{k+1}) = \max_{\forall\xi}\{C(\chi_k \cup \{\xi\}) : \ \xi \in \{\xi_j\}_{j=1,\ldots,d} \setminus \chi_k\}$$

$$\text{Initialization :} \quad \chi_0 \equiv \emptyset$$

End of Algorithm

It is known that the best pair of features does not necessarily contain the individual best feature selected in the first step of the algorithm [2]. As a result, the main source of suboptimality of the SFS method is that once a feature is included in the selected feature set, there is no mechanism for removing it from the feature set even if at a later stage, when more features have been added, this feature becomes superfluous.

As pointed out by Kittler [8], another drawback of the SFS method is that although it takes into account the statistical dependence between candidate features and those already selected, the number of candidate features to be added at each step of the algorithm is restricted to one. Hence, due to this restriction it is impossible to take into consideration statistical dependence between elements of the set of available attributes.

5.1.2 Sequential backward selection

The SFS algorithm has a counterpart called the sequential backward selection (SBS) algorithm which is a top-down process. Starting from the complete set of available attributes $\{\xi_j\}_{j=1,\ldots,d}$, one attribute at a time is discarded until *d-m* attributes have been deleted. At each stage the attribute to be discarded from the current feature set is selected so that the newly reduced set of features gives the maximum possible value of the criterion function.

The Sequential Backward Selection Algorithm

Suppose k features have already been removed from the complete set of attributes, $\{\xi_j\}_{j=1,\ldots,d}$, to form feature set $\bar{\chi}_k$. The $(k+1)^{st}$ feature to be eliminated is then chosen from the set $\bar{\chi}_k$ so that

$$C(\bar{\chi}_{k+1}) = \max_{\forall\xi}\{C(\bar{\chi}_k \setminus \{\xi\}) : \ \xi \in \bar{\chi}_k\}$$

$$\text{Initialization :} \quad \bar{\chi}_0 \equiv \{\mathrm{\xi_j}\}_{\mathrm{j=1,\ldots,d}}$$

End of Algorithm

The drawbacks of the SBS method are analogous to those of the SFS method. However, the main difference between these two methods is that the SBS procedure provides as a by-product a measure of maximum achievable class separability with the given set of features which can be used to assess the amount of information loss in the feature selection process. As far as the computational complexity is concerned, the SFS method is simpler than the SBS method since it requires that the criterion function be evaluated at most in m-dimensional spaces. In contrast, in the SBS method the criterion function must be computed in spaces of dimensionality ranging from d down to m.

5.1.3 Generalized sequential forward and generalized sequential backward selection

The statistical dependence of features to be added to or discarded from the current feature set, depending on the type of search procedure employed, can be taken into consideration by adding to or subtracting from the current feature set more than one attribute at a time. The following methods have been designed by Kittler [8] to accommodate this approach.

The generalized sequential forward selection GSFS(r) algorithm is similar to the basic SFS algorithm but with r attributes being considered at any one time instead of just one attribute. Starting from an empty set, we perform exhaustive search to find the best r attributes from the complete set of available attributes $\{\xi_j\}_{j=1,\dots,d}$. At each subsequent stage the next best r features are picked from the remaining available attributes so that in combination with the features already selected they yield the best value of the criterion function.

The Generalized Sequential Forward Selection Algorithm

Suppose k features have already been selected to form feature set χ_k. Generate all the possible sets of size r, χ_r, from the set of available attributes $\{\xi_j\}_{j=1,\dots,d} \setminus \chi_k$. Then select the next r features so that in combination with χ_k, the overall criterion function is maximum, i.e.

$$C(\chi_{k+r}) = \max_{\forall \chi_r}\{C(\chi_k \cup \chi_r) : \ \chi_r \subseteq \{\xi_j\}_{j=1,\dots,d} \setminus \chi_k$$

$$\text{Initialization :} \quad \chi_0 \equiv \emptyset$$

End of Algorithm

The generalized sequential backward selection GSBS(r) algorithm is, by analogy, essentially the same as the SBS algorithm with the exception that more than one features are discarded at a time. Starting from the complete set of available attributes $\{\xi_j\}_{j=1,\dots,d}$, the worst r attributes are discarded. At each subsequent stage the worst r attributes from the remaining available set of attributes are deleted until m features are left in the remaining set.

The Generalized Sequential Backward Selection Algorithm

Suppose k features have been discarded from the set of available attributes $\{\xi_j\}_{j=1,\dots,d}$ to form feature set $\bar{\chi}_k$. Now form all the possible sets $\bar{\chi}_{k+r}$ by removing various combinations of r attributes from $\bar{\chi}_k$. Then select as $\bar{\chi}_{k+r}$ the candidate feature set that maximizes the criterion function, i.e.

$$C(\bar{\chi}_{k+r}) = \max_{\forall \chi_r}\{C(\bar{\chi}_k \setminus \chi_r) : \ \chi_r \subseteq \bar{\chi}_k\}$$

$$\text{Initialization :} \quad \bar{\chi}_0 \equiv \{\xi_\mathrm{j}\}_{\mathrm{j=1,\dots,d}}$$

End of Algorithm

A few comments are in order at this point. Although the GSFS(r) algorithm is more reliable than the basic SFS method, it is also more costly in computational terms; for at each stage $\binom{d-k}{r}$ feature sets, $\chi_k \cup \chi_r$, have to be inspected to find the feature set χ_{k+r}. Similarly, the GSBS(r) algorithm requires a substantial increase in computation since at each stage $\binom{d-k}{r}$ sets must be evaluated in comparison with the *d-k* candidate sets in the case of the SBS algorithm. However, the GSBS(r) algorithm takes into consideration not only statistical dependence among features in the current feature set, $\bar{\chi}_k$, but also the relationship between the discarded attributes. Note that both the GSFS(r) and GSBS(r) algorithms still suffer from the nesting of successive feature sets.

5.1.4 Plus l-minus r Algorithm

The nesting of feature sets, which may rapidly influence in suboptimality of both the SFS and SBS algorithms, can be *partially* overcome by alternating the process of augmentation and depletion of the feature set. This process can be viewed as the search method using the dynamic principle of optimization. After adding l attributes to the current feature set, r features are removed. Thus the net change in the size of feature set is equivalent to l-r, hence the name “plus l-minus r (l, r)" selection algorithm. This process is continued until the feature set reaches the required size. There are many different ways to achieve the net change in size of the current feature set by l-r features, however the easiest is to apply the basic SFS and SBS algorithms alternately.

The Plus l-Minus r Algorithm

Let χ_k be the current feature set and if $l > r$ then

- **Step 1** : Apply SFS l times to generate feature set χ_{k+l}
- **Step 2** : Apply SBS r times to obtain feature set χ_{k+l-r}
- **Step 3** : Stop if $k + l - r = m$ else return to Step 1

The procedure for $l < r$ is the same as above with Step 1 and Step 2 interchanged. Note that the (l, r) algorithm with $r = 0$ is the SFS algorithm and with $l = 0$ it is the SBS algorithm.

End of Algorithm

Although the nesting in the (l, r) algorithm is avoided, it should be noted that eliminating nesting does not eliminate the Cover paradox [2]. Also, the (l, r) algorithm still suffers from other drawbacks of the SFS and SBS algorithms namely that groups of features are added and removed from the current feature set irrespective of their mutual relationship, i.e. only one candidate feature is being considered at a time. Furthermore, it is in principle impossible to order features according to their significance since the features in subsets of cardinalities differing by one, e.g. χ_k and χ_{k+1} or χ_k and χ_{k-1}, may differ by more than one features. Finally, another drawback of the (l, r) algorithm is that no theoretical indication for what values of l and r will yield the best result, consequently multiple runs are usually needed.

5.1.5 Generalized “plus l-minus r" algorithm

Another approach to achieve the net change in size of the current feature set set by l-r features is to employ the GSFS(l) and GSBS(r) algorithms in alternation. After adding the best l features with respect to the already selected features, the worst r features are removed from the newly enlarged feature set. This process is continued until the required size of features is reached.

The Generalized “Plus l-Minus r" Algorithm

Suppose k features have been selected to form set χ_k and if $l > r$ then

- **Step 1** : Enlarge χ_k by applying GSFS(l) to generate feature set χ_{k+l}
- **Step 2** : Reduce χ_{k+l} by applying GSBS(r) to obtain feature set χ_{k+l-r}
- **Step 3** : Stop if $k + l - r = d$ else return to Step 1

Initialization of the generalized (l,r) algorithm is identical to that of the (l,r) algorithm discussed earlier. Note that the procedure for $l < r$ is the same as above with Step 1 and Step 2 interchanged.

End of Algorithm

This greater sophistication of the feature set selection procedure is achieved once again at the expense of extra computations which is, in turn, preventing l and r from being too large. However, it is possible to curb computational complexity of the algorithm if the GSFS(l) and GSBS(r) algorithms are split into a number of substeps (see [8]).

5.2 Floating search strategies

So far, the simple way to avoid nesting of feature sets is to employ either the (l,r) or generalized (l,r) algorithm which involves successive augmentation and depletion process. Consequently, the resulting dimensionality in respective stages of both algorithms is fixed depending on the prespecified values of l and r. Unfortunately, there is no theoretical way of predicting the values of l and r to achieve the best feature set. Alternatively, instead of fixing these values, there is a couple of more sophisticated procedures of which these values are flexibly changing so as to approximate the optimal solution as much as possible. Although both of these methods switch between including and excluding features, they are recognized as two different algorithms according to the dominant direction of the search. The search in the forward direction is known as the *sequential forward floating selection* (SFFS), while in the opposite direction is known as the *sequential backward floating selection* (SBFS) [12]. They are known as floating methods because the resulting dimensionality in respective stages of the algorithm is not changing monotonously but is actually "floating" up and down.

Before describing the corresponding algorithms formally, the following notions have to be introduced:
We shall say that the feature ξ from the set χ is
(a) the *most significant* (best) feature *in the set* χ if $C(\chi \setminus \{\xi\}) = \min_{\zeta\in\chi} C(\chi \setminus \{\zeta\})$,
(b) the *least significant* (worst) feature *in the set* χ if $C(\chi \setminus \{\xi\}) = \max_{\zeta\in\chi} C(\chi \setminus \{\zeta\})$.
We shall say that the feature ξ from the set $\{\xi_j\}_{j=1,\dots,d} \setminus \chi$ is
(c) the *most significant* (best) feature *with respect to the set* χ if

$$C(\chi \cup \{\xi\}) = \max_{\zeta\in\{\xi_j\}_{j=1,\dots,d}\setminus\chi} C(\chi \cup \{\zeta\}),$$

(d) the *least significant* (worst) feature *with respect to the set* χ if

$$C(\chi \cup \{\xi\}) = \min_{\zeta\in\{\xi_j\}_{j=1,\dots,d}\setminus\chi} C(\chi \cup \{\zeta\}).$$

Having defined the notions of the least and most significant features in the set and with respect to the set, we can describe the SFFS and SBFS algorithms.

5.2.1 Sequential forward floating search

The SFFS is basically a bottom up search procedure which includes new features by means of applying the basic SFS procedure to the current feature set, followed by a series of successive conditional exclusion of the worst feature in the newly updated set if a further improvement can be made to the previous sets.

The Sequential Forward Floating Selection Algorithm

Suppose k features have already been selected from the complete set of attributes $\{\xi_j\}_{j=1,\dots,d}$ to form set χ_k with the corresponding criterion function $C(\chi_k)$. In addition, the values of $C(\chi_i)$ for all preceding subsets of size $i = 1, 2, \cdots, k-1$, are known and stored.

- **Step 1** :(Inclusion). Find the most significant feature ξ with respect to the set χ_k and create a feature set $\chi_{k+1} = \chi_k \cup \{\xi\}$.
- **Step 2** :(Conditional Exclusion). Find the least significant feature ζ in the set χ_{k+1}. If ξ is the least significant feature in the set χ_{k+1}, i.e. $\xi = \zeta$ then set $k = k{+}1$ and return to Step 1, but if some other $\xi' \in \chi_k$ is the least significant feature in the set χ_{k+1}, then exclude ξ' from χ_{k+1} to form a new feature set χ'_k, i.e.

$$\chi'_k = \chi_{k+1} \setminus \{\xi'\}.$$

  Note that now $C(\chi'_k) > C(\chi_k)$. If $k = 2$, set $\chi_k = \chi'_k$ and return to Step 1 else go to Step 3.
- **Step 3** :(Continuation of conditional exclusion). Find the least significant feature ξ'' in the set χ'_k. If $C(\chi'_k \setminus \{\xi''\}) > C(\chi_{k-1})$ then exclude ξ'' from χ'_k to form a newly reduced set χ'_{k-1}, i.e.

$$\chi'_{k-1} = \chi'_k \setminus \{\xi''\}.$$

  Set $k = k$-1. Now if $k = 2$ then set $\chi_k = \chi'_k$ and return to Step 1 else repeat Step 3.

Initially, the procedure starts from an empty set, $\chi_0 \equiv \emptyset$, and the first two features are selected by the SFS method.

End of Algorithm

5.2.2 Sequential backward floating search

The SBFS is a top down search procedure which excludes features by means of applying the basic SBS procedure to the current feature set and followed by a series of successive conditional inclusions of the most significant feature from the available features if an improvement can be made to the previous sets.

The Sequential Backward Floating Selection Algorithm

Suppose k features have already been removed from the complete set of attributes $\bar{\chi}_0 = \{\xi_j\}_{j=1,\ldots,d}$ to form feature set $\bar{\chi}_k$ with the corresponding criterion function $C(\bar{\chi}_k)$. Furthermore, the values of criterion function for all supersets $\bar{\chi}_i$, $i = 1, 2, \cdots, k-1$, are known and stored.

- **Step 1** :(Exclusion). Use the basic SBS method to remove feature ξ from the current set $\bar{\chi}_k$ to form a reduced feature set $\bar{\chi}_{k+1}$, i.e. the least significant feature ξ is deleted from the set $\bar{\chi}_k$.
- **Step 2** :(Conditional inclusion). Find among the excluded features the most significant feature ζ with respect to the set $\bar{\chi}_{k+1}$, i.e.

$$C(\bar{\chi}_{k+1} \cup \{\zeta\}) = \max_{\forall \zeta'} C(\bar{\chi}_{k+1} \cup \{\zeta'\}), \quad \zeta' \in \{\xi_j\}_{j=1,\ldots,d} \setminus \bar{\chi}_{k+1}.$$

  If ξ is the most significant feature with respect to $\bar{\chi}_{k+1}$, i.e $\xi = \zeta$ then set $k = k+1$ and return to Step 1. If $\xi \neq \zeta$, i.e.

$$C(\bar{\chi}_{k+1} \cup \{\zeta\}) > C(\bar{\chi}_k)$$

  then include ζ to the set $\bar{\chi}_{k+1}$ to form a new feature set $\bar{\chi}'_k$, i.e.

$$\bar{\chi}'_k = \bar{\chi}_{k+1} \cup \{\zeta\}.$$

  If $k = 2$, set $\bar{\chi}_k = \bar{\chi}'_k$ and return to Step 1 else go to Step 3.
- **Step 3** :(Continuation of conditional inclusion). Find among the excluded features the most significant feature ζ'' with respect to the set $\bar{\chi}'_k$. If $C(\bar{\chi}'_k \cup \{\zeta''\}) > C(\bar{\chi}_{k-1})$ then include ζ'' to the set $\bar{\chi}'_k$ to form the new enlarged set $\bar{\chi}'_{k-1}$, i.e.

$$\bar{\chi}'_{k-1} = \bar{\chi}'_k \cup \{\zeta''\}.$$

  Set $k = k-1$. Now if $k = 2$ then set $\bar{\chi}_k = \bar{\chi}'_k$ and return to Step 1 else repeat Step 3.

The procedure begins by deleting the first two features using the basic SBS method.

End of Algorithm

Unlike the (l, r) and generalized (l, r) algorithms in which factors such as the net change in the size of the current feature set, and especially the amount of computational time, are governed by the values of l and r, the SFFS and SBFS methods are not restricted to these factors. This is because both methods are freely allowed to correct wrong decisions made in the previous steps so as to approximate the optimal solution as much as possible. As a result, there is no net change in the size of the current feature set in both methods.

5.3 Adaptive floating search

Though the floating search algorithms have been found very successful and evaluated by independent comparative studies as the currently most efficient algorithms, even their performance can still be somewhat improved, though at the expense of the increased computational time. In this section the generalization of floating search algorithms, called "adaptive floating search" is presented.

In the previous section we have introduced notions of the best (most significant) and the worst (least significant) feature in the set, and with respect to the set of features χ. Before discussing the adaptive floating search algorithms formally, the following generalization of these notions has to be introduced.

Denote by τ_o an o-tuple belonging to the set $\Theta(o, \chi_k)$ of all $\binom{k}{o}$ possible o-tuples from χ_k:

$$\Theta(o, \chi_k) = \{\tau \subset \chi_k : |\tau| = o\}.$$

We shall say that the feature o-tuple τ_o from the set χ_k is
(a) the *most significant* (*best*) feature o-tuple *in the set* χ_k if

$$C(\chi_k \backslash \tau_o) = \min_{\tau \in \Theta(o, \chi_k)} C(\chi_k \backslash \tau),$$

(b) the *least significant* (*worst*) feature o-tuple *in the set* χ_k if

$$C(\chi_k \backslash \tau_o) = \max_{\tau \in \Theta(o, \chi_k)} C(\chi_k \backslash \tau).$$

We shall say that the feature o-tuple υ_o from the set $\{\xi_j\}_{j=1,\ldots,d} \backslash \chi_k$ is
(c) the *most significant* (*best*) feature o-tuple *with respect to the set* χ_k if

$$C(\chi_k \cup \upsilon_o) = \max_{\upsilon \in \Theta(o, \{\xi_j\}_{j=1,\ldots,d} \backslash \chi_k)} C(\chi_k \cup \upsilon),$$

(d) the *least significant* (*worst*) feature o-tuple *with respect to the set* χ_k if

$$C(\chi_k \cup \upsilon_o) = \min_{\upsilon \in \Theta(o, \{\xi_j\}_{j=1,\ldots,d} \backslash \chi_k)} C(\chi_k \cup \upsilon).$$

Remark. For $o = 1$ all the terms relating to the feature o-tuple significance coincide with the terms relating to the *individual significance* of a feature.

5.3.1 Adaptive floating search (AFS) properties

For the sake of simplicity, let us denote original floating search methods (SFFS and SBFS) together as "classical floating search" methods and denote them by CFS. CFS methods use only single feature adding or removing, respectively, in the course of the algorithm.

Our new search strategy aims to utilize the best of both generalized strategies and classical floating strategies. The course of search is similar to that of CFS, but the individual search steps are generalized. However, the new algorithm is by no means just a generalized version of CFS.

The basic generalization of CFS would be that replacing the simple SFS or SBS steps inside the CFS procedure by their generalized versions GSFS(o) or GSBS(o), respectively. Unfortunately for a potential user, it is generally not known which value of o to choose to get the best results. Moreover, if the user chooses the value of o too high for his particular problem, this leads to useless increase of computing time.

As opposed to the above mentioned generalized methods, the AFS method does not need the user-specified level of generalization. This level (value of o) is determined dynamically in the course of search according to the current situation so as to achieve better results. In this sense, AFS brings about a similar improvement in comparison with the generalized search strategies as the CFS search brought about in comparison with simple "non-generalized" strategies (CFS introduced dynamical "floating" of adding/removing steps).

Because of the time-demanding character of generalized steps (especially when used in high-dimensional problems) we introduced a user defined parametric limit r_{max}, restricting the maximum generalization level which the method can use. The current generalization level, which changes in the course of search, is denoted by o. The AFS is called "adaptive" because of its ability to adjust the limit under which the actual generalization level can be automatically set. Simply said, the nearer the current subset size k is to the final one m, the higher is the generalization limit. This characteristic aims to save computing time by limiting the generalization levels while the current subset is still far from the desired one. Therefore, we introduce the variable r representing the actual generalization limit for a given dimension.

To summarize, r_{max} is a user-specified absolute generalization limit, r is the actual generalization limit determined adaptively by the algorithm for the current subset ($r \leq r_{max}$ always holds), o is the current generalization level depending on the current situation ($o \leq r$ always holds).

Remark. for $r_{max} = 1$ the AFS is identical to classical floating search.

Adaptive determination of r is done as follows: at the beginning of every forward or backward algorithm phase, respectively:

1. if $|k - m| < b$, let $r = r_{max}$
2. else if $|k - m| < b + r_{max}$, let $r = r_{max} + b - |k - m|$
3. else let $r = 1$

Here b denotes the neighbourhood of the final dimension, where the highest generalization levels are allowed. Basically it is possible to set $b = 0$.

Thus, in the generalized course of the AFS algorithm $o = 1$ is used in usual algorithm stages (e.g., in the beginning). Only special algorithm stages (when conditional forward, resp. backward steps brought no improvement) allow increasing of o and, therefore, a more detailed search.

By setting r_{max} or b to higher values, the user has a possibility to let the algorithm perform more thorough search with better chances to find the optimal solution, of course at the expense of longer computation time. The setting of these two parameters is not so critical as setting the generalization level in classical GSFS(o), resp GSBS(o) and Plus-l-Minus-r. Increasing r_{max} or b does not lead to a different search, but to a more detailed search.

Remark. Every AFS algorithm run includes steps indentical with CFS ones. Also for this reason we expect the AFS algorithm to find equal or better solutions than CFS. The computer time needed for AFS is expected to be substantially longer (due to the generalization) than for CFS. However, if we constructed the generalized floating search in the simple way, it would consume incomparably more time.

CFS were found to occasionally prefer worse working subsets in the course of search. To describe that case, let us remind the principle of CFS (specifically SFFS) first:

1. Add the *most significant feature* to the current subset of size k. Let $k = k + 1$.
2. Conditionally remove the *least significant feature* from the current subset.
3. If the current subset is the so-far best subset of size $k - 1$, let $k = k - 1$ and go to step 2. Else return the conditionally removed feature and go to step 1.

Note that backward steps are conditional. Only backward steps bringing improvement are allowed. On the other hand, forward steps cannot be conditional. If they were conditional, the algorithm could theoretically fall into an infinite cycle (repeated by conditional adding and removing a feature). Because of their unconditionality, the forward steps can lead to finding worse subset, than was the so-far best one of a given dimension. A less prospective "search branch" is thus uselessly followed.

Removal of this problem we have accepted is simple. If the forward step found a subset worse than the best one known so-far, the current one is forgotten and the so-far best one becomes the current one. Note that this "violent" swapping of current subset cannot lead to infinite cycling as finding of a worse subset by the forward step must have been preceded by finding a better subset in some lower dimension.

Now, having defined the notion and discussed the included principles we can describe the ASFFS and ASBFS algorithms.

5.3.2 ASFFS algorithm

The ASFFS (Adaptive Sequential Forward Floating Search) is basically a "bottom up" procedure. The algorithm is initialized by setting $k = 0$ and $\chi_0 = \emptyset$. In order to keep the algorithm description traceable, we did not include all the steps needed to ensure its proper functioning, especially when the current dimension gets near to 0 or d. Such steps serve to avoid the algorithm running outside the meaningful dimension boundaries.

Suppose the so-far best values of criterion function $C(\chi_i)$ are stored as C_i^{max} for all $i = 1, 2, \cdots, d$. The corresponding so-far best feature subsets χ_i are also stored. Initial values of C_i^{max} for all $i = 1, 2, \cdots, d$ should be set to lowest possible value. Furthermore, suppose k is the size of the current subset.

ASFFS Algorithm

A. Forward Phase.
Each phase begins with adaptive setting of r: If $|k-m| < b$, let $r = r_{max}$. Else if $|k-m| < r_{max}+b$, let $r = r_{max} + b - |k-m|$. Else let $r = 1$.

```
Step 1: Let $o = 1$.
Step 2: (Conditional inclusion)
        Using the basic GSFS($o$) method, select the most significant o-tuple $v_o$ from the set of available
        attributes $\{\xi_j\}_{j=1,\ldots,d} \backslash \chi_k$ with respect to the set $\chi_k$, then add it to $\chi_k$ to form feature set $\chi_{k+o}$.
Step 3: If $C(\chi_{k+o}) > C^{max}_{k+o}$, let $C^{max}_{k+o} = C(\chi_{k+o})$, let $k = k + o$ and go to Step 6. (The so-far best subset
        of size $k + o$ was found).
Step 4: (Conditional increase of generalization step)
        If $o < r$, let $o = o + 1$ and go to Step 2. (The conditionally included features are removed).
Step 5: (None of the subsets tested in the forward phase were better than the so-far best ones.)
        Forget the current subset $\chi_k$. Let $k = k + 1$. Now consider the so-far best subset of size $k$ to be
        the current subset $\chi_k$.
Step 6: (Testing the terminating condition) If $k \geq d + \Delta$, stop the algorithm.
```

B. Backward Phase.
Each phase begins with adaptive setting of r: If $|k-m| < b$, let $r = r_{max}$. Else if $|k-m| < r_{max}+b$, let $r = r_{max} + b - |k-m|$. Else let $r = 1$.

```
Step 7: Let $o = 1$.
Step 8: (Conditional exclusion)
        Using the basic GSBS($o$) method, select the least significant o-tuple $\tau_o$ in the set $\chi_k$, then remove
        it from $\chi_k$ to form feature set $\chi_{k-o}$.
Step 9: If $C(\chi_{k-o}) > C^{max}_{k-o}$, let $C^{max}_{k-o} = C(\chi_{k-o})$, let $k = k-o$ and go back to the beginning of Backward
        Phase. (The so-far best subset of size $k - o$ was found).
Step 10: (Conditional increase of generalization step)
         If $o < r$, let $o = o + 1$ and go to Step 8. (The conditionally excluded features are returned).
Step 11: (None of the subsets tested in the backward phase were better than the so-far best ones.) Go to
         Forward phase.
```

End of Algorithm

5.3.3 ASBFS algorithm

The algorithm is initialized in the same way as ASFFS, except $k = D$ and $\chi_D = \{\xi_j\}_{j=1,\ldots,d}$. The ASBFS (Adaptive Sequential Backward Floating Search) is the "top down" counterpart to the ASFFS procedure. Since it is analogous to the forward one, it is unnecessary to describe it here.

5.4 Oscillating search

In this section a new sub-optimal subset search method for feature selection is introduced. As opposed to other till now known subset selection methods the oscillating search is not dependent on pre-specified direction of search (forward or backward). The generality of oscillating search concept allowed us to define several different algorithms suitable for different purposes. We can specify the need to obtain good results in very short time, or let the algorithm search more thoroughly to obtain near-optimum results. In many cases the oscillating search over-performed all the other tested methods. The oscillating search may be restricted by a preset time-limit, what makes it usable in real-time systems.

Note, that most of known suboptimal search strategies are based on step-wise adding of features to initially empty feature set, or step-wise removing features from the initial set of all features, $\{\xi_j\}_{j=1,\ldots,d}$. One of search directions, *forward* or *backward*, is usually preferred, depending on several factors [13], the expected difference between the original and the final required cardinality being the most important one. Regardless of the direction, it is apparent, that all these algorithms spend a lot of time testing feature subsets having cardinalities far distant from the required cardinality m.

Before describing the principle of oscillating search, recall the notions introduced at the beginning of Section 5.3: the "worst" feature o-tuple in χ_m is the set $\tau_w \in \Theta(o, \chi_m)$, that

$$C(\chi_m \setminus \tau_w) = \max_{\tau \in \Theta(o,\chi_m)} C(\chi_m \setminus \tau).$$

The "best" feature o-tuple with respect to χ_m is the set $\tau_b \in \Theta(o, \{\xi_j\}_{j=1,\dots,d} \setminus \chi_m)$, that

$$C(\chi_m \cup \tau_b) = \max_{\tau \in \Theta(o,\{\xi_j\}_{j=1,\dots,d} \setminus \chi_m)} C(\chi_m \cup \tau).$$

In practice we allow also suboptimal finding of the "worst" and "best" o-tuples to save the computational time (see later).

5.4.1 Oscillating search algorithms

The *oscillating search* (OS) is based on repeated modification of the current subset χ_m of m features. This is achieved by alternating the *down-* and *up-swings*. The *down-swing* removes o "worst in the set" features from the current set χ_m to obtain a new set χ_{m-o} at first, then adds o "best with respect to the set" ones to χ_{m-o} to obtain a new current set χ_m. The *up-swing* adds o "best with respect to the set" features to the current set χ_m to obtain a new set χ_{m+o} at first, then removes o "worst in the set" ones from χ_{m+o} to obtain a new current set χ_m again. Let us denote two successive opposite swings as an *oscillation cycle*. Using this notion, the oscillating search consists of repeating oscillation cycles. The value of o will be denoted *oscillation cycle depth* and should be set to 1 initially. If the last oscillation cycle did not find better subset χ_m of m features, the algorithm increases the oscillation cycle depth by letting $o = o + 1$. Whenever any swing finds better subset χ_m of m features, the depth value o is restored to 1. The algorithm stops, when the value of o exceeds the user-specified *limit* Δ.

Every oscillation algorithm assumes the existence of some initial set of m features. Obtaining such an initial set will be denoted as an *initialization*. Oscillating algorithms may be initialized in different ways; the simplest ways are random selection or the SFS procedure. From this point of view the oscillating search may serve as a mechanism for tuning solutions obtained in another way.

Whenever a feature o-tuple is to be added (or removed) from the current subset in the till now known methods, one of two approaches is usually utilized: the *generalized* adding (or removing) finds the optimum o-tuple by means of exhaustive search (e.g. in GSFS, GSBS) or the *successive adding (or removing) single features o-times* (e.g. in basic Plus-l-Minus-r), which may fail to find the optimum o-tuple, but is significantly faster.

In fact, we may consider finding feature o-tuples to be equivalent to the feature selection problem, though at a "second level". Therefore, we may use any search strategy for finding feature o-tuples. Because of proved effectiveness of *floating search* strategies we adopted the floating search principle as the third approach to adding (or removing) feature o-tuples in oscillating search. Note that in such a way one basic idea has resulted in defining a couple of new algorithms, as shown in the sequel.

For the sake of simplicity, let us denote the adding of feature o-tuple by ADD(o) and the removing of feature o-tuple by REMOVE(o). Finally, we introduce three versions of oscillating algorithm:

1. *Sequential oscillating search*: ADD(o) represents a sequention of o successive SFS steps, REMOVE(o) represents a sequention of o successive SBS steps.

2. *Floating oscillating search*: ADD(o) represents adding of o features by means of the SFFS procedure, REMOVE(o) represents removing of o features by means of the SFBS procedure.

3. *Generalized oscillating search*: ADD(o) represents adding of o features by means of the GSFS(o) procedure, REMOVE(o) represents removing of o features by means of the GSBS(o) procedure.

Oscillating Search Algorithm

Remark. c serves as a swing counter.

Step 1: *Initialization:* by means of the SFS procedure (or otherwise) find the initial set χ_m of m features. Let $c = 0$. Let $o = 1$.

Step 2: *Down-swing:* By means of the REMOVE(o) step remove the "worst in the set" feature o-tuple from χ_m to form a new set χ_{m-o}. *(** If the $C(\chi_{m-o})$ value is not the so-far best one among subsets having cardinality $m - o$, stop the down-swing and go to **Step 3**. **)*
By means of the ADD(o) step add the "best with respect to the set" feature o-tuple from $\{\xi_j\}_{j=1,\ldots,d} \setminus \chi_{m-o}$ to χ_{m-o} to form a new subset χ'_m. If the $C(\chi'_m)$ value is the so-far best one among subsets having required cardinality m, let $\chi_m = \chi'_m$, $c = 0$ and $o = 1$ and go to **Step 4**.

Step 3: *Last swing did not find better solution:*
Let $c = c + 1$. If $c = 2$, then none of previous two swings has found better solution; extend the search by letting $o = o + 1$. If $o > \Delta$, stop the algorithm, otherwise let $c = 0$.

Step 4: *Up-swing:* By means of the ADD(o) step add the "best with respect to the set" feature o-tuple from $\{\xi_j\}_{j=1,\ldots,d} \setminus \chi_m$ to χ_m to form a new set χ_{m+o}. *(** If the $C(\chi_{m+o})$ value is not the so-far best one among subsets having cardinality $m + o$, stop the up-swing and go to **Step 5**. **)*
By means of the REMOVE(o) step remove the "worst in the set" feature o-tuple from χ_{m+o} to form a new set χ'_m. If the $C(\chi'_m)$ value is the so-far best one among subsets having required cardinality m, let $\chi_m = \chi'_m$, $c = 0$ and $o = 1$ and go to **Step 2**.

Step 5: *Last swing did not find better solution:*
Let $c = c + 1$. If $c = 2$, then none of previous two swings has found better solution; extend the search by letting $o = o + 1$. If $o > \Delta$, stop the algorithm, otherwise let $c = 0$ and go to **Step 2**.

End of Algorithm

Remark. Parts of code enclosed in *(** and **)* brackets may be omitted to obtain a bit slower, more thorough procedure.

The three introduced algorithm versions differ in their effectiveness and time complexity. The *generalized oscillating search* gives the best results, but its use is restricted due to the time complexity of generalized steps (especially for higher Δ). The *floating oscillating search* is suitable for finding solutions as close to optimum as possible in a reasonable time even in high-dimensional problems. The *sequential oscillating search* is the fastest but possibly the least effective algorithm version with respect to approaching the optimal solution.

6 Feature selection operators specifications

The feature selection operators implemented according to the algorithms described in Section 5 are all machine learning ones. We selected only following algortihms to be impemented: sequential forward selection (section 5.1.1), sequential backward selection (section 5.1.2), sequential forward floating search (section 5.2.1), and sequential backward floating search (section 5.2.2). All these algorithms can be summarised as shown in Fig. 1.

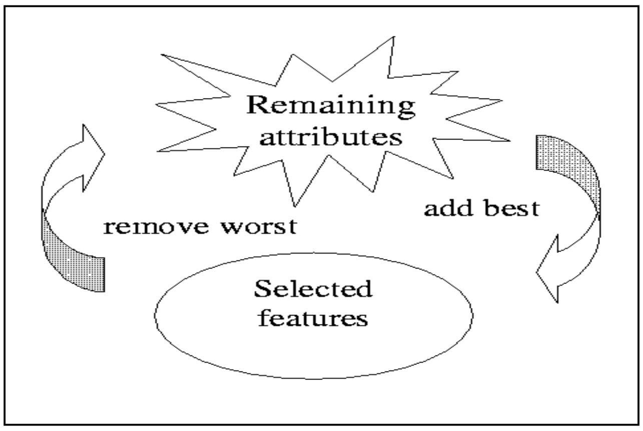

Fig. 1 General scheme of feature selection operators

The forward selection operators try to find (and add) the best attribute from the set of remaining attributes, thus performing the cycle

```
    forall att in Remaining_attributes evaluate
        {att} ∪ Selected_features
```

The backward selection operators try to find (and remove) the worst feature from the set of already selected features, thus performing the cycle

```
    forall feat in Selected_features evaluate
        Selected_features \ {feat}
```

For fixed search (sequential forward selection and sequential backward selection), only one search strategy is used. For floating search, after adding (resp. removing) an attribute, an attempt is done to find a worst (resp. best) attribute that can be removed (resp. added) to improve the actually created subset of attributes.

All these four operators use the required number of attributes as the stopping criterion for the search.

To complete the list of operators, we include also a manual one, where the user has full control over the selection of attributes.

All operators described bellow share a common part that creates a specification of selected features (SQL statement) for TheOutputConcept according to AttList.

Select

```
Create View T as Select AttList from T0
```

6.1 SimpleForwardFeatureSelectionGivenNoOfAttributes

This operator adds (from TheAttributes) one feature a time starting from empty set until the required number of features NoOfAttributes is reached. The attributes are selected with respect to TheClassAttribute, the group optimises the information dependence criterion. Use this operator if only a small number of original attributes is to be selected.

- **Operator**
    - *name* SimpleForwardFeatureSelectionGivenNoOfAttributes
    - *loopable* – no
    - *multistepable* – no
    - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheAttributes	1	NULL	IN	BA
TheClassAttribute	1	1	IN	BA
NoOfAttributes	1	1	IN	V
TheOutputConcept	1	1	OUT	CON

- **Constraints**
    - TheClassAttribute is CATEGORIAL
    - TheClassAttribute is in TheInputConcept
    - TheAttributes is CATEGORIAL
    - TheAttributes is in TheInputConcept
    - NoOfAttributes is NUMERIC
    - NoOfAttributes > 0

- **Conditions**
    - TheAttributes NOT_NULL
    - TheClassAttribute NOT_NULL

- **Assertions**
    - TheOutputConcept PROJ of TheInputConcept

- **OPChecks**
    - NoOfAttributes < no.of.attributes.in.TheAttributes

6.2 SimpleBackwardFeatureSelectionGivenNoOfAttributes

This operator removes (from TheAttributes) one feature a time starting from all attributes until the required number of features NoOfAttributes is reached. The attributes are selected with respect to TheClassAttribute, the group optimises the information dependence criterion. Use this operator if a large number of original attributes is to be selected.

- **Operator**
    - *name* SimpleBackwardFeatureSelectionGivenNoOfAttributes
    - *loopable* – no
    - *multistepable* – no
    - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheAttributes	1	NULL	IN	BA
TheClassAttribute	1	1	IN	BA
NoOfAttributes	1	1	IN	V
TheOutputConcept	1	1	OUT	CON

- **Constraints**
    - TheClassAttribute is CATEGORIAL
    - TheClassAttribute is in TheInputConcept
    - TheAttributes is CATEGORIAL
    - TheAttributes is in TheInputConcept
    - NoOfAttributes is NUMERIC
    - NoOfAttributes > 0

- **Conditions**
    - TheAttributes NOT_NULL
    - TheClassAttribute NOT_NULL

- **Assertions**
    - TheOutputConcept PROJ of TheInputConcept

- **OPChecks**
    - NoOfAttributes < no.of.attributes.in.TheAttributes

6.3 FloatingForwardFeatureSelectionGivenNoOfAttributes

This operator adds (from TheAttributes) one feature a time starting from empty set until the required number of features NoOfAttributes is reached. The attributes are selected with respect to TheClassAttribute, the group optimises the information dependence criterion. Unlike the simple operator, after adding a feature a check is performed if another feature should be removed. Use this operator if only a small number of original attributes is to be selected.

- **Operator**
    - *name* FloatingForwardFeatureSelectionGivenNoOfAttributes
    - *loopable* – no
    - *multistepable* – no
    - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheAttributes	1	NULL	IN	BA
TheClassAttribute	1	1	IN	BA
NoOfAttributes	1	1	IN	V
TheOutputConcept	1	1	OUT	CON

- **Constraints**
    - TheClassAttribute is CATEGORIAL
    - TheClassAttribute is in TheInputConcept
    - TheAttributes is CATEGORIAL
    - TheAttributes is in TheInputConcept
    - NoOfAttributes is NUMERIC
    - NoOfAttributes > 0

- **Conditions**
    - TheAttributes NOT_NULL
    - TheClassAttribute NOT_NULL

- **Assertions**
    - TheOutputConcept PROJ of TheInputConcept

- **OPChecks**
    - NoOfAttributes < no.of.attributes.in.TheAttributes

6.4 FloatingBackwardFeatureSelectionGivenNoOfAttributes

This operator removes (from TheAttributes) one feature a time starting from all attributes until the required number of features NoOfAttributes is reached. The attributes are selected with respect to TheClassAttribute, the group optimises the information dependence criterion. Unlike the simple operator, after removing a feature a check is performed if another feature should be added. Use this operator if a large number of original attributes is to be selected.

- **Operator**
  - *name* FloatingBackwardFeatureSelectionGivenNoOfAttributes
  - *loopable* – no
  - *multistepable* – no
  - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheAttributes	1	NULL	IN	BA
TheClassAttribute	1	1	IN	BA
NoOfAttributes	1	1	IN	V
TheOutputConcept	1	1	OUT	CON

- **Constraints**
  - TheClassAttribute is CATEGORIAL
  - TheClassAttribute is in TheInputConcept
  - TheAttributes is CATEGORIAL
  - TheAttributes is in TheInputConcept
  - NoOfAttributes is NUMERIC
  - NoOfAttributes > 0

- **Conditions**
  - TheAttributes NOT_NULL
  - TheClassAttribute NOT_NULL

- **Assertions**
  - TheOutputConcept PROJ of TheInputConcept

- **OPChecks**
  - NoOfAttributes < no.of.attributes.in.TheAttributes

6.5 UserDefinedFeatureSelection

This operator selects the attributes TheSelectedAttributes given by the user.

- **Operator**
  - *name* UserDefinedFeatureSelection
  - *loopable* – no
  - *multistepable* – no
  - *manual* – yes

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheSelectedAttributes	1	NULL	IN	BA
TheOutputConcept	1	1	OUT	CON

- **Constraints**
  - ??TheOutputConcept is in TheInputConcept
  - TheSelectedAttributes is in TheInputConcept

- **Conditions**
- **Assertions**
    - TheOutputConcept PROJ of TheInputConcept

References:

[1] J. M. Van Campenhout. On the peaking of the Hughes mean recognition accuarcy: The resolution of an apparent paradox. *IEEE Transactions on System, Man and Cybernetics*, SMC-8:390-395, May 1978.

[2] T. M. Cover and J. M. Van Campenhout. On the possible orderings in the measurement selection problem. *IEEE Transactions on System, Man and Cybernetics*, SMC-7:657-661, September 1977.

[3] T. M. Cover. The best two independent measurements are not the two best. *IEEE Transactions on System, Man and Cybernetics*, SMC-4:116-117, January 1974.

[4] R. P. W. Duin and B. J. Krose. On the possibility of avoiding peaking. In: Proc 5th international conference on pattern recognition, 1375-1378, Miami beach, Florida, USA, December 1980.

[5] P. A. Devijver and J. Kittler. Pattern Recognition: A Statistical Approach. Prentice-Hall, 1982.

[6] R. G. Gallager. Information Theory and Reliable Communication, John Wiley, 1968.

[7] G. F. Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, IT-14(1):55-63, January 1968.

[8] J. Kittler. Feature set search algorithms. In: Pattern Recognition and Signal Processing, (C. H. Chen, Ed.), 41-60, The Netherlands: Sijthoff and Noordhoff, 1978.

[9] S. Kullback. Information Theory and Statistics. Dover Publications, Inc., New York, 1968.

[10] P.M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, C-26:917-922, September 1977.

[11] P. Pudil. Methodology and Advances in Feature Selection for Statistical Pattern Recognition, DrSc Thesis, UTIA AV CR, Praha, 2001.

[12] P. Pudil, S. Blaha and J. Novovicova, PREDITAS - software package for solving pattern recognition and diagonostic problems. In: Proc of BPRA 4th Internat Conf. on Pattern Recognition, Cambridge (J. Kittler, ed.) Spring-Verlag Berlin-Heidelberg-New York, 1988, 146-152.

[13] P. Pudil, J. Novovicova and S. Blaha. Statistical Approach to Pattern Recognition: Theory and Practical Solution by Means of PREDITAS System, *Kybernetika*, 27(1991), pp.1-78, Supplement.

[14] P. Pudil, J. Novovicova, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15:1119-1125, November 1994.

[15] S. J. Raudys and A. K. Jain. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(3):252--264, March 1991.

[16] G. V. Trunk. A problem of dimensionality: A simple example. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(3):306-307, July 1979.