Oscillating Search Algorithms for Feature Selection

Somol P., Pudil P.
Dep. of Pattern Recognition, Inst. of Information Theory and Automation
Academy of Sciences of the Czech Republic, 182 08 Prague 8, Czech Republic
{somol,pudil}@utia.cas.cz

Abstract

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.

1. Introduction

In feature selection the search problem of finding a subset X_d of d features from the given set Y of D measurements, $d < D$, so as to maximize an adopted criterion, $J(.)$, has been of interest for a long time. Since the optimal methods (exhaustive search or the Branch-and-Bound method which is additionally restricted to monotonous criteria) are not suitable for high-dimensional problems because of their exponential nature, research has concentrated on suboptimal polynomial search methods.

A number of search methods has been developed, starting with the well-known *sequential forward selection* (SFS) and ending with *floating search methods* [5, 6]. A good overview of subset search methods may be found in [1].

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, Y. One of search directions, *forward* or *backward*, is usually preferred, depending on several factors [4], 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 d.

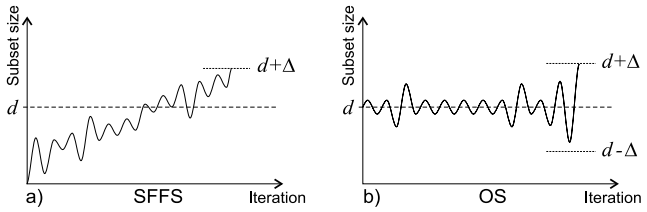


Figure 1. Graphs demonstrate the course of search algorithms: a) sequential floating forward selection [5], b) oscillating search.

Before describing the principle of oscillating search, let us introduce the following notion: the "worst" feature o-tuple in X_d should be ideally such a set $\bar{W} \subset X_d$, that

$$J(X_d \setminus \bar{W}) = \max_{\mathcal{W} \subset X_d, |\mathcal{W}|=o} J(X_d \setminus \mathcal{W}).$$

The "best" feature o-tuple for X_d should be ideally such a set $\bar{B} \subset Y \setminus X_d$, that

$$J(X_d \cup \bar{B}) = \max_{\mathcal{B} \subset Y \setminus X_d, |\mathcal{B}|=o} J(X_d \cup \mathcal{B}).$$

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

2. Oscillating Search

Unlike other methods, the *oscillating search* (OS) is based on repeated modification of the current subset X_d of d features. This is achieved by alternating the *down-* and *up-swings*. The *down-swing* removes o "worst" features from the current set X_d to obtain a new set X_{d-o} at first, then adds o "best" ones to X_{d-o} to obtain a new current set X_d. The *up-swing* adds o "good" features to the current set X_d to obtain a new set X_{d+o} at first, then removes o "bad" ones

from X_{d+o} to obtain a new current set X_d 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 X_d of d features, the algorithm increases the oscillation cycle depth by letting $o = o + 1$. Whenever any swing finds better subset X_d of d features, the depth value o is restored to 1. The algorithm stops, when the value of o exceeds the user-specified *limit* Δ. The course of oscillating search is illustrated on picture 1.

Every oscillation algorithm assumes the existence of some initial set of d 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 (see [1]), 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 (see [5]), 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 (see [1]), 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 X_d of d features. Let $c = 0$. Let $o = 1$.

Step 2: *Down-swing:* By means of the REMOVE(o) step remove the "worst" feature o-tuple from X_d to form a new set X_{d-o}. *(* If the $J(X_{d-o})$ value is not the so-far best one among subsets having cardinality $d - o$, stop the down-swing and go to* **Step 3**.**)* By means of the ADD(o) step add the "best" feature o-tuple from $Y \setminus X_{d-o}$ to X_{d-o} to form a new subset $X_d^{'}$. If the $J(X_d^{'})$ value is the so-far best one among subsets having required cardinality d, let $X_d = X_d^{'}$, $c = 0$ 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" feature o-tuple from $Y \setminus X_d$ to X_d to form a new set X_{d+o}. *(* If the $J(X_{d+o})$ value is not the so-far best one among subsets having cardinality $d + o$, stop the up-swing and go to* **Step 5**.**)* By means of the REMOVE(o) step remove the "worst" feature o-tuple from X_{d+o} to form a new set $X_d^{'}$. If the $J(X_d^{'})$ value is the so-far best one among subsets having required cardinality d, let $X_d = X_d^{'}$, $c = 0$ 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**.

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

The algorithm is described also by a float chart on picture 2. 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.

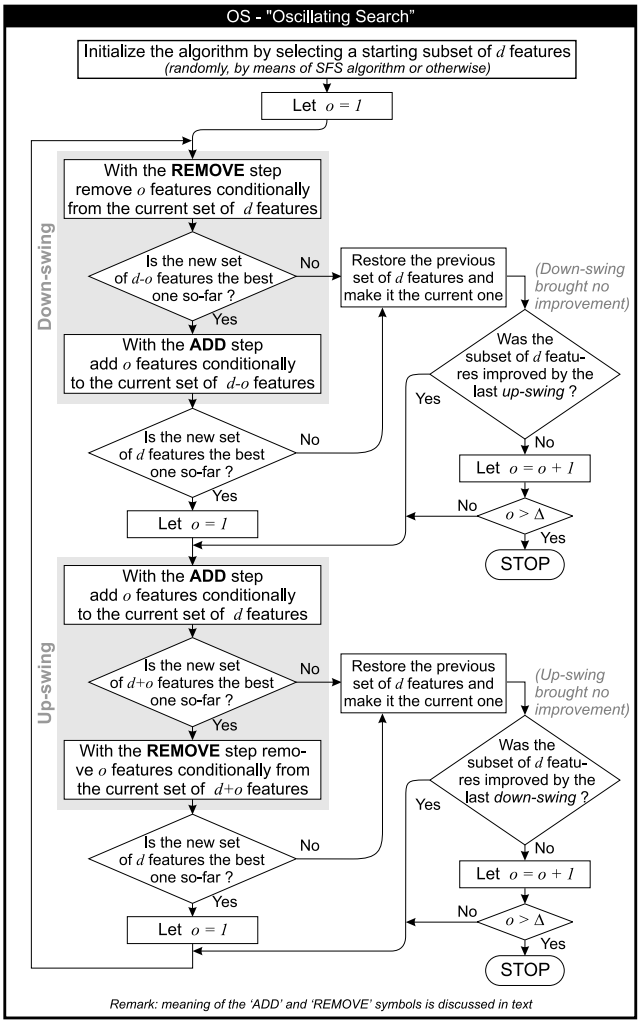


Figure 2. Simplified OS algorithm flow-chart.

3. Oscillating Search Properties

All oscillating algorithms may be characterized as algorithms independent on the direction of search (neither bottom-up nor top-down). Every algorithm cycle attempts to improve the current subset of cardinality d. Therefore OS algorithms do not loose time by evaluating subsets of distant cardinalities. All OS versions overcome effectively the "nesting problem" which SFS and SBS suffer from.

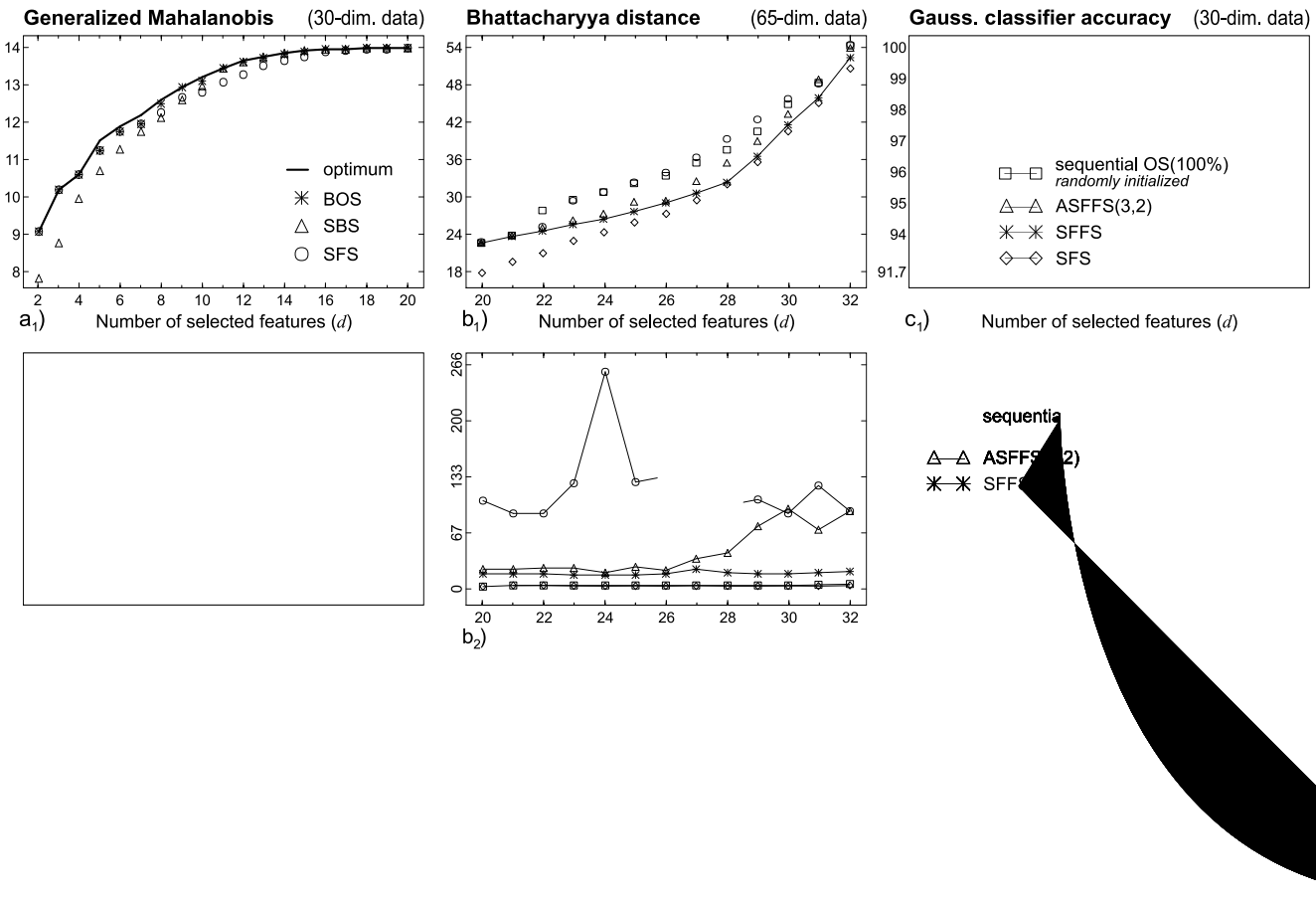
The fastest improvement of the current subset may be expected in initial phases of the algorithm, because of the low initial cycle depth. Later, when the current feature subset evolves closer to optimum, low-depth cycles fail to improve the subset and therefore the algorithm increases the depth (o) value. Though this will increase the chance to get closer to optimum, the trade-off between finding a better solution and the computational time clearly exists. It has been found out that it is often possible to find out very good solution after a short time corresponding to the first few low-depth cycles. This algorithm behavior is advantageous, because it is possible to obtain usually very good solutions after very short time. The oscillating search principle proves to be useful for different purposes:

- It may be looked upon as a universal tuning mechanism, being able to improve solutions obtained in any other way. In this sense the oscillating search is usable in addition to any subset search method.
- Let us denote the sequential OS with $\Delta = 1$, initialized by SFS, the *basic oscillating search* - BOS. This simple algorithm is suitable for problems being currently solved by SFS or SBS methods. Both the time and implementation complexity of the BOS is comparable to the complexity of these simple methods (see Experiments). However, the BOS algorithm is much more effective.
- The floating OS finds usually one of the best achievable suboptimal solutions. When used for solving high-dimensional problems, the floating OS finds usually better solutions, than both classical and adaptive floating strategies [5, 6].
- The randomly initialized sequential OS is a very fast heuristic, that often finds results not achievable in other ways. However, because of its randomized nature, we may not expect it to find good results each time.
- Repeating of oscillation cycles allows to stop the algorithm after specified amount of time and still to obtain a usable solution. Because of this property the OS is suitable also for use in real-time systems.

Sub-optimal search algorithms are often criticized because of their dependence on user parameters. The OS procedure assumes user-setting of the *oscillation cycle depth* Δ. Setting higher Δ value should result in more thorough search. Note, that we may consider the maximum meaningful Δ value, which is $max\{d, D-d\}$. Based on the knowledge of this maximum, we may allow the user to specify Δ value relatively, i.e. independently of current problem properties (d and D values). Maximum performance would be therefore achieved by utilizing OS(100%), although lower values (20 – 30%) are more suitable for most tasks, especially when computational time is to be restricted.

4. Experiments

The oscillating search methods were tested on a large number of different problems. We demonstrate their performance on 2-class, 30-dimensional mammogram data from Wisconsin Diagnostic Breast Center (obtained from ftp.ics.uci.edu) and 2-class, 65-dimensional mammogram data from PRIM Lab., University of California, Irvine.


Generalized Mahalanobis (30-dim. data)
optimum
BOS
SBS
SFS
a1) Number of selected features (d)
Bhattacharyya distance (65-dim. data)
b1) Number of selected features (d)
Gauss. classifier accuracy (30-dim. data)
sequential OS(100%)
randomly initialized
ASFFS(3,2)
SFFS
SFS
c1) Number of selected features (d)
b2)