A Mixed Learning Strategy for Finding Typical Testors in Large Datasets

Víctor Iván González-Guevara[1], Salvador Godoy-Calderon[1], Eduardo Alba-Cabrera[2](✉), and Julio Ibarra-Fiallo[2]

[1] Instituto Politécnico Nacional, Centro de Investigación en Computación (CIC), Av. Luis Enrique Erro S/N, Unidad Profesional Adolfo López Mateos, Zacatenco, Delegación Gustavo A. Madero, 07738 Ciudad de Mexico, Mexico
{vgonzalez,sgodoyc}@cic.ipn.mx

[2] Colegio de Ciencias e Ingenierías, Departamento de Matemáticas, Universidad San Francisco de Quito (USFQ), Diego de Robles y Vía Interoceanica, Quito, Ecuador
{ealba,jibarra}@usfq.edu.ec
http://www.springer.com/lncs

Abstract. This paper presents a mixed, global and local, learning strategy for finding typical testors in large datasets. The goal of the proposed strategy is to allow any search algorithm to achieve the most significant reduction possible in the search space of a typical testor-finding problem. The strategy is based on a trivial classifier which partitions the search space into four distinct classes and allows the assessment of each feature subset within it. Each class is handled by slightly different learning actions, and induces a different reduction in the search-space of a problem. Any typical testor-finding algorithm, whether deterministic or metaheuristc, can be adapted to incorporate the proposed strategy and can take advantage of the learned information in diverse manners.

Keywords: Feature selection · Testor theory · Algorithms

1 Introduction

Feature Selection is a well known branch of Pattern Recognition responsible for identifying those features, describing objects under study, that provide relevant information for classification purposes. Testor Theory is one of the common tools used for such task. During the last decade several algorithms have been designed for finding the set of all typical testors in a dataset [5,7,9]. Unfortunately, the time complexity of computing all typical testors has an exponential growth with respect to the number of features describing objects. Also, recent research has unveiled different elements that also have effect over the complexity of that problem, such as the number of rows in the initial basic matrix, the density

Mexican authors wish to thank CONACyT and SIP-IPN for their support of this research, particularly through grant SIP-20151393. Also, Ecuatorian authors wish to thank the financial support received from USFQ-Small Grants.

of that matrix, and the number of typical testors within it or the underlying structure of the basic matrix $Agregarcitaaartculo4delrevisorMemo$. All those factors severely complicate finding typical testors in large datasets. Moreover, some of the empirical results found in carefully designed benchmarks like the one on [2], add up to the intuition that no single typical testor-finding algorithm can be found to have the best possible behavior for any given problem. This kind of *No-Free-Lunch* intuition, taken from the field of evolutionary and bio-inspired algorithms, encourages researching techniques that allow increased algorithm performance and solving the problem of finding typical testors in large datasets with the least possible computational cost.

With that goal in mind, this paper proposes a mixed learning strategy designed to allow typical testor-finding algorithms to reduce the search space of any problem. The proposed strategy will be responsible for identifying and storing the pertinent local and global information for the stated purpose, while the underlying typical testor-finding algorithm (a search algorithm) decides when and how it makes use of the learned information.

2 Theoretical Framework

Several research papers have more than exhaustively presented the fundamental definitions of Testor Theory. Here we quickly outline the context for our particular research, and advise the reader who requires a thorough review of those concepts to refer to [4] and [6].

All known typical testor-finding algorithms have a comparison matrix as input. That matrix, called Basic Matrix, contains the summary information about the comparison of all objects belonging to different classes within a given supervision sample. When the original supervision sample is a partition, and all comparisons have been evaluated with a boolean difference function, then the basic matrix (BM) is binary and its rows conform a *Sperner* family. Within such matrix, a *Testor* is defined as a subset τ of columns (or features) such that no zero-row can be found in $BM|_\tau$ (called the τ-restricted matrix). Also, a *Typical Testor* is defined as an irreducible testor (i.e. a testor such that none of its subsets is a testor). As a consequence of its irreducibility, typical testors are identified by the property that each feature in $BM|_\tau$ has at least one *Typical Row*, where the corresponding column contains a 1, and all other columns in that row contain a 0.

In practical terms, typical testors are characterized by being the only feature subsets that fulfill the two following conditions:

1. $BM|_\tau$ has no zero-rows (so τ is a testor)
2. Each column in $BM|_\tau$ has at least one typical row (so τ is typical)

Largely, the complexity of finding typical testors in big matrices lies in the analysis of restricted matrices in search for the fulfillment of both conditions. In order to minimize that effort, and as the basis for our proposed learning strategy we briefly introduce the following concepts:

2.1 Masks and Assessment Indices

The following definitions are introduced in [5]:
Let B be a basic matrix, and let τ be any feature subset in B, then

Definition 1. *The Acceptance Mask of τ ($am(\tau)$) is a binary tuple in which the ith element is 1 if the ith row in $B|_\tau$ has at least a 1 in the columns of features in τ, and 0 otherwise.*

Definition 2. *The Compatibility Mask of τ ($cm(\tau)$) is a binary tuple in which the ith element is 1 if the ith row in $B|_\tau$ has only a 1 in the column of a feature in τ and 0 otherwise.*

To the previous definitions, we add the following,

Definition 3. *The Typicity Mask of τ ($tm(\tau)$) is an integer tuple in which the ith element is the number of typical rows that feature x_i has in $B|_\tau$.*

The defined masks allow the characterization of all feature subsets, either as a Testor or as a Typical. For that task we define the following indexes:

Definition 4. *The Testor Error index of subset τ in B ($e_T(\tau)$), is the number of zero rows in $B|_\tau$ (i.e. the number of zero entries in $am(\tau)$).*

Definition 5. *The Typical Error index of subset τ in B ($e_{Ty}(\tau)$), is the number of features in τ that do not have at least one typical row in $B|_\tau$ (i.e. the number of zero entries in $tm(\tau)$).*

For algorithmic purposes, both error indexes are interpreted as the number of changes a particular feature subset has to undergo in order to become a testor or a typical testor.

2.2 The Classifier

By using the error indexes defined in previous subsection we define a trivial classifier that effectively partitions the search space (i.e. the power set of all feature subsets) in four classes: *Testors*, *Typicals*, *TypicalTestors*, and *Incompatibles* (See Fig.1).

The first three classes have already been presented: *Testors*, *Typicals* and *Typical Testors* are characterized by the conditions discussed in section 2. *Incompatibles*, on the other hand, are those feature subsets whose restricted matrix contains one or more zero-rows, and where not all features have a typical row.

Often, the reason why a particular feature fails to have a typical row, is because another feature, in the same subset, damages its potential typical rows by having a 1 in the same row. We call that condition an *incompatibility* between those two features, and have identified it as one of the most important phenomena to be learned. When, during the search for typical testors, an incompatibility is found within a feature subset, the search algorithm can safely ignore the analysis of any other feature subset containing the identified pair of incompatible features.

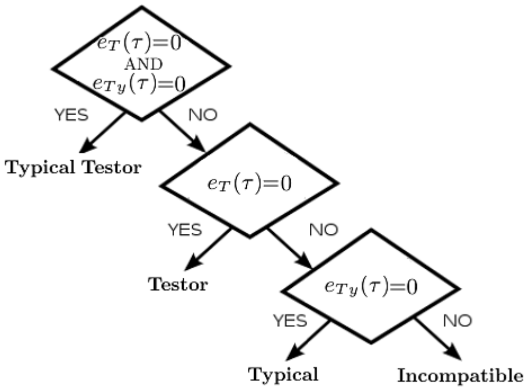

Fig. 1. The feature subset classifier

3 Learning Strategy

By *learning* we mean the process by which any piece of information (knowledge), relevant for the solution of a problem, is identified and stored to be used later. So, as stated before, the goal of the proposed strategy is to provide some additional knowledge to enable any typical testor-finding algorithm for a more efficient search process.

All typical testor-finding algorithms proceed iteratively by analyzing one or more feature subsets from the search space on each iteration, and then deciding which other elements in the search space are to be, or not to be analyzed next. This decision is made by a set of rules that follow a pre-defined order in which the search space is to be traversed (even a random order). Deterministic algorithms such as those in [5,7,9] generally analyze one subset at a time, while metaheuristic algorithms like those in [1,3,8] tend to analyze more than one subset on each iteration.

When a typical testor-finiding algorithm is adapted to use this strategy, the learning module learns all that can be learned from each feature subset the search algorithm analyzes. The resulting knowledge can then be used as a form of *taboo list* that allows the algorithm to skip the analysis of some subsets with the guarantee that no typical testor is going to be missed by the overall process. The specific way in which the proposed learning strategy can be adapted to any algorithm is briefly described in the next subsection.

3.1 Adapting the Strategy

Regardless of the specific order that a typical testor-finding algorithm follows, there are only two specific modifications that it needs to undergo in order to adapt the proposed learning strategy:

1. Allow the learning module to analyze each feature subset the algorithm selects,
2. Avoid the analysis of any feature subset whose structure has already been learned,

The analysis of each feature subset starts by classifying it as *Testor*, *Typical*, *TypicalTestor* or *Incompatible*. The resulting label triggers different learning actions. If the analyzed subset turns out to be *TypicalTestor* the whole subset is learned globally to ensure that the algorithm never tests any of its subsets or supersets. Similarly, if the subset is *Incompatible* (i.e. it has zero-rows and includes feature incompatibilities) all pairs of incompatible features are globally learned so that the search algorithm never tests a subset that includes any of those incompatibilities. In both cases the learned information is considered global in the sense that it remains constant during the whole run of the search algorithm. Locally learned information, on the other hand, only stores information that serve as a reference point for deciding which subsets not to analyze. This kind of knowledge is updated as the algorithm runs. Such is the case for subsets labeled as *Testor* or as *Typical* which have one important property in common: either their *Testor Error index*, or their *Typical Error index* evaluates to zero. By following that line of reason, it quickly becomes clear that finding a *Testor* immediately rules out the analysis of any of its supersets, while finding a *Typical* rules out the analysis of any of its subsets.
Any known typical testor-finding algorithm can be adapted to incorporate the proposed strategy. Figure 3 presents a tiny example where the proposed learning strategy is used to analyze several feature subsets and generate their descendants.

4 Pseudo-Code and Sample Experiments

In this section we present the algorithmic form of the proposed strategy as well as some experiments performed with it.

4.1 Pseudo-Code

Let B be a basic matrix with columns labeled with the elements of set R (a complete feature set). We call *descendants* of τ any feature subset within the search space whose analysis is not ruled out after analyzing τ. Also, let $Incomp(\tau)$ be a procedure that finds and returns all pairwise incompatibilities in subset τ. Figure 2 outlines the proposed learning strategy in the form of a function that receives a $\tau \subset R$ as input, and returns its filtered descendants. Two auxiliary functions were used for clearly stating the pseudocode in Figure 2, those functions are:

1. $Classify()$: Receives a feature subset as input, and outputs the corresponding class label (See Figure 1).
2. $FilterWithGlobalLearning()$: Receives a family of feature subsets as input, and uses any previously stored global learning information to filter the set.

```
Learning Strategy.
Input: A feature subset $\tau$
Output: The filtered descendants of $\tau$

label = $Classify(\tau)$
Case label of
    Testor do
        Store: $\tau$ and $e_{Ty}(\tau)$
        Descendants = $\{\sigma \subset \tau \mid e_{Ty}(\sigma) < e_{Ty}(\tau)\}$
    Typical do
        Store: $\tau$ and $e_T(\tau)$
        Descendants = $\{\sigma \supset \tau \mid e_{Ty}(\sigma) = 0 \ \wedge \ e_T(\sigma) < e_T(\tau)\}$
    Typical Testor do
        Store: $\tau$
        Descendants = $\{\sigma \in \wp(R) \mid \sigma \nsubseteq \tau \ \wedge \ \sigma \nsupseteq \tau)\}$
    Incompatible do
        Store: $\tau$, $e_T(\tau)$, $e_{Ty}(\tau)$, and $Incomp(\tau)$
        Descendants = $\{\sigma \in \wp(R) \mid e_T(\sigma) \leq e_T(\tau) \}$
endCase
Descendants = $FilterWithGlobalLearning$(Descendants)
```

Fig. 2. Pseudocode for the proposed learning strategy

In order to ilustrate the mechanics of the learning strategy, Table 1 shows a tiny basic matrix (called the *Ms* matrix). Some feature subsets from the *Ms* matrix are analyzed following the the pseudo-code in Figure 1. The results are summarized in Table 2.

Table 1. The *Ms* matrix used for illustrating the learning strategy

a	*b*	*c*	*d*	*e*	*f*
1	0	0	0	1	0
1	1	0	0	0	1
0	0	1	0	0	1
1	0	0	1	0	1

4.2 Sample Experiments

We adapted the well-known BR-algorithm [5] to use the proposed learning strategy, and compared the number of tested subsets (labeled as *Hits*) and the execution time between the original algorithm and the adapted one. Input matrices for these experiments were designed by applying the ϕ, θ, and γ operators, over the *Ms* matrix, following the specifications and method described in [2]. Table 3 summarizes the experiments performed showing the number of rows, columns and typical testors (*TTestors*) of each input matrix.

Table 2. Some examples of four different cases

τ	$e_T(\tau), e_{Ty}(\tau)$	*Label*	*Learning*	*Descendants*
$\tau_1 = \{a,c,e\}$	$e_T(\tau_1) = 0$ $e_{Ty}(\tau_1) = 1$	*Testor*	$[\{a,c,e\},1]$	$\{\{a,c\}\}$
$\tau_2 = \{b,c\}$	$e_T(\tau_2) = 2$ $e_{Ty}(\tau_2) = 0$	*Typical*	$[\{b,c\},2]$	$\{\{b,c,d\},\{b,c,e\},\{b,c,d,e\},\{b,c,d,f\}\}$
$\tau_3 = \{a,f\}$	$e_T(\tau_3) = 0$ $e_{Ty}(\tau_3) = 0$	*Typical Testor*	$[\{a,f\}]$	$\{\{b\},\{c\},\{d\},\{e\},\{a,b\},\{a,c\},\{a,d\},$ $\{a,e\},\{b,c\},\{b,d\},\{b,e\},\{b,f\},\{c,d\},$ $\{c,e\},\{c,f\},\{d,e\},\{d,f\},\{e,f\},\{a,b,c\},$ $\{a,b,d\},\{a,b,e\},\{a,c,d\},\{a,c,e\},$ $\{a,d,e\},\{b,c,d\},\{b,c,e\},\{b,c,f\},$ $\{b,d,e\},\{b,d,f\},\{b,e,f\},\{c,d,e\},$ $\{c,d,f\},\{c,e,f\},\{d,e,f\},\{a,b,c,d\},$ $\{a,b,c,e\},\{a,b,d,e\},\{a,c,d,e\},$ $\{b,c,d,e\},\{b,c,d,f\},\{b,c,e,f\},$ $\{b,d,e,f\},\{c,d,e,f\},\{a,b,c,d,e\},$ $\{b,c,d,e,f\}\}$
$\tau_4 = \{b,c,d,f\}$	$e_T(\tau_4) = 1$ $e_{Ty}(\tau_4) = 4$	*Incompatible*	$[\{b,c,d,f\},1,4$ $(b,f),(c,f),(d,f)]$	$\{\{a,c\},\{a,f\},\{e,f\},\{a,b,c\},\{a,c,d\},$ $\{a,c,e\},\{a,e,f\},\{a,b,c,d\},\{a,b,c,e\},$ $\{a,c,d,e\},\{b,c,d,e\},\{a,b,c,d,e\}\}$

Table 3. Performance of the adapted BR-algorithm

Matrix	Rows	Cols	TTestors	Original BR		Adapted BR	
				Hits	Time	Hits	Time
Id_5	5	5	1	16	0.001	16	0.001
Id_{15}	15	15	1	16384	1.582	16384	1.580
$\phi^2(Ms)$	4	12	28	180	0.002	68	0.001
$\phi^3(Ms)$	4	18	108	2361	0.03	222	0.006
$\theta^2(Ms)$	16	12	8	617	0.011	216	0.013
$\theta^3(Ms)$	64	18	12	30979	3.986	4196	1.157
$\gamma^2(Ms)$	8	12	16	352	0.007	252	0.037
$\gamma^3(Ms)$	16	24	256	166252	152.617	111132	22.305

As it can be seen, the proposed strategy cannot improve the search process in the case of identity matrices. However, the number of hits is notoriously reduced in all other cases, effectively reducing the problem's search space. Also note that the execution time is not always reduced proportionally, since the internal structure of the input matrix can sometimes severely complicate the calculation of masks and error indices.

5 Conclusions

A general mixed learning strategy for finding typical testors in large datasets was presented. The proposed strategy uses both globally and locally learned information to calculate the descendants of the currently analyzed feature subset, effectively reducing the search space for any problem. The host search algorithm however, must decide the order in which the search space is analyzed, as well as the size of its population. The interaction and dependence between the host algorithm and the learning module determines, for the most part, the performance yield by any experiment. Since there are no current means for predicting the optimum order for traversing a search space or for maximizing the use of learned information, the intuition that no single algorithm can be found to optimally solve any problem instance is strengthened.

Evidently the problem of finding typical testors still stands as not solvable in polynomial time; however, the proposed strategy is enough to cut out from the analysis all feature subsets that neither have real possibilities of being typical testors, nor contribute to the rest of the search process.

References

1. Alba-Cabrera, E., Santana, R., Ochoa-Rodriguez, A., Lazo-Corts, M.: Finding typical testors by using an evolutionary strategy. In: Proceedings of the 5th Ibero American Symposium on Pattern Recognition, p. 267 (2000)
2. Alba-Cabrera, E., Ibarra-Fiallo, J., Godoy-Calderon, S.: A theoretical and practical framework for assessing the computational behavior of typical testor-finding algorithms. In: Ruiz-Shulcloper, J., Sanniti di Baja, G. (eds.) CIARP 2013, Part I. LNCS, vol. 8258, pp. 351–358. Springer, Heidelberg (2013)
3. Diaz-Sanchez, G., Piza-Davila, I., Sanchez-Diaz, G., Mora-Gonzalez, M., Reyes-Cardenas, O., Cardenas-Tristan, A., Aguirre-Salado, C.: Typical testors generation based on an evolutionary algorithm. In: Yin, H., Wang, W., Rayward-Smith, V. (eds.) IDEAL 2011. LNCS, vol. 6936, pp. 58–65. Springer, Heidelberg (2011)
4. Lazo-Cortés, M., Ruiz-Shulcloper, J., Alba-Cabrera, E.: An overview of the evolution of the concept of testor. Pattern Recognition **34**(4), 753–762 (2001)
5. Lias-Rodríguez, A., Pons-Porrata, A.: BR: a new method for computing all typical testors. In: Bayro-Corrochano, E., Eklundh, J.-O. (eds.) CIARP 2009. LNCS, vol. 5856, pp. 433–440. Springer, Heidelberg (2009)
6. Martnez-Trinidad, J.F., Guzmán-Arenas, A.: The logical combinatorial approach to pattern recognition, an overview through selected works. Pattern Recognition **34**(4), 741–751 (2001)
7. Sanchez-Diaz, G., Lazo-Cortes, M., Piza-Davila, I.: A fast implementation for the typical testor property identification based on an accumulative binary tuple. International Journal of Computational Intelligence Systems **5**(6), 1025–1039 (2012)
8. Sanchez-Diaz, G., Diaz-Sanchez, G., Mora-Gonzalez, M., Piza-Davila, I., Aguirre-Salado, C., Huerta-Cuellar, G., Reyes-Cardenas, O., Cardenas-Tristan, A.: An evolutionary algorithm with acceleration operator to generate a subset of typical testors. Pattern Recognition Letters **41**, 34–42 (2014)
9. Santiesteban-Alganza, Y., Pons-Porrata, A.: LEX: A new algorithm for calculating typical testors. Revista Ciencias Matematicas **21**(1), 85–95 (2003)