BR: A New Method for Computing All Typical Testors

Alexsey Lias-Rodríguez[1] and Aurora Pons-Porrata[2]

[1] Computer Science Department
lias@csd.uo.edu.cu
[2] Center for Pattern Recognition and Data Mining
Universidad de Oriente, Santiago de Cuba, Cuba
aurora@cerpamid.co.cu

Abstract. Typical testors are very useful in Pattern Recognition, especially for Feature Selection problems. The complexity of computing all typical testors of a training matrix has an exponential growth with respect to the number of features. Several methods that speed up the calculation of the set of all typical testors have been developed, but nowadays, there are still problems where this set is impossible to find. With this aim, a new external scale algorithm *BR* is proposed. The experimental results demonstrate that this method clearly outperforms the two best algorithms reported in the literature.

Keywords: typical testors, feature selection.

1 Introduction

One of the problems in Pattern Recognition is Feature Selection, which consists on finding the features that provide relevant information in the classification process. In the Logical Combinatorial Pattern Recognition [1] feature selection is commonly carried out by using Testor Theory [2]. In this theory, a testor is defined as a set of features that distinguishes the objects of different classes. A testor is called irreducible (typical) if none of its proper subsets is a testor. When we refer to typical testors (TT), we restrict us to typical Zhuravlev's testors, where classes are crisp and disjoint sets, the comparison criteria for features are Boolean and the similarity measure assumes two objects as different if they are so in at least one of the features.

Typical testors have been widely used to evaluate the feature relevance [3] and as support sets in classification algorithms [4]. In Text Mining, they have also been used for text categorization [5] and document summarization [6]. Several algorithms have been developed to calculate the typical testors. They can be classified according to its computational strategy into two categories: external and internal scale algorithms. The first perform the TT calculation by generating elements of the power set of features in a predefined order, but trying to avoid the analysis of irrelevant subsets. The second ones explore the internal structure of the training matrix and find conditions that guarantee the testor property. In this paper, we focus on the first strategy.

The complexity of computing all typical testors has an exponential growth with respect to the number of features. Methodologies that speed up the calculation of typical testors have been developed, but nowadays, there are still problems where the set of

E. Bayro-Corrochano and J.-O. Eklundh (Eds.): CIARP 2009, LNCS 5856, pp. 433–440, 2009.
© Springer-Verlag Berlin Heidelberg 2009

all typical testors is impossible to find. Therefore, it is very important to develop better algorithms for obtaining typical testors. The external scale methods *LEX* [7] and *CT-EXT* [8] are reported to be the most efficient ones.

With this aim, we propose *BR*, a new external scale method that avoids the analysis of a greater number of irrelevant subsets and efficiently verifies the testor property by profiting from the computer bit operations. The method name is due to these Binary operations and its Recursive nature. The experimental results demonstrate that this method clearly outperforms the two best algorithms reported in the literature [8].

2 Basic Concepts

Before presenting our method, we review the main definitions of the Testor Theory and we define the basic concepts of this method.

Let *TM* be a training matrix containing m objects described in terms of n features $\Re=\{X_1,\ldots,X_n\}$ and distributed into r classes $\{C_1,\ldots,C_r\}$. Each feature X_i takes values in a set D_i, i=1,…,n. A comparison criterion of dissimilarity $\psi_i : D_i \times D_i \rightarrow \{0,1\}$ is associated to each X_i (0=similar, 1=dissimilar). Applying these comparison criteria for all possible pairs of objects belonging to different classes in *TM*, a Boolean dissimilarity matrix, denoted by *DM*, is built. Notice that the number of rows in *DM* is $m' = \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} |C_i||C_j|$, where $|C_i|$ denotes the number of objects in the class C_i.

Let p and q be two rows of *DM*. p is a *subrow of* q if in all columns where p has 1, q has also it. A row p of *DM* is called *basic* if no row in *DM* is a subrow of p. The submatrix of *DM* containing all its basic rows (without repetitions) is called a *basic matrix* (*BM*). Then, a *testor* is a subset of features $\tau=\{X_{i_1},\ldots,X_{i_s}\}$ of *TM* for which a whole row of zeros does not appear in the remaining submatrix of *BM*, after eliminating all columns corresponding to the features in $\Re\backslash\tau$. τ is a typical testor if there is no proper subset of τ that meets the testor condition [2]. Commonly, algorithms used for computing typical testors make use of *BM* instead of *DM* due to the substantial reduction of rows.

Let $(a_1,\ldots,a_u)$ be a binary u-tuple of elements, $a_i \in \{0,1\}$, i=1,…,u. We call *cardinal of a binary u-tuple* to the number of its elements (i.e., u). The column corresponding to a feature X in *BM* is a binary u-tuple, whose cardinal is the number of rows in *BM*. We will denote this u-tuple by c_X. We also define logical operations on binary u-tuples as follows:

$(a_1, a_2, \ldots, a_u) \vee (e_1, e_2, \ldots, e_u) = (a_1 \vee e_1, a_2 \vee e_2, \ldots, a_u \vee e_u)$

$(a_1, a_2, \ldots, a_u) \wedge (e_1, e_2, \ldots, e_u) = (a_1 \wedge e_1, a_2 \wedge e_2, \ldots, a_u \wedge e_u)$

$\neg(a_1, a_2, \ldots, a_u) = (\neg a_1, \neg a_2, \ldots, \neg a_u)$

$(a_1, a_2, \ldots, a_u) \otimes (e_1, e_2, \ldots, e_u) = (a_1 \otimes e_1, a_2 \otimes e_2, \ldots, a_u \otimes e_u)$, where $\otimes$ denotes the XOR operation.

(1,…,1) and (0,…,0) represent binary u-tuples in which all elements are one and zero, respectively.

The notation $[X_1,\ldots,X_s]$, $X_i \in \Re$, is used to represent an ordered list of features and *last*$([X_1,\ldots,X_s])$ denotes the last element in the list, i.e. X_s. A list does not contain features is denoted as []. We call *length of a list l*, denoted as $|l|$, to the number of its features. All basic operations of the set theory (difference, intersection, subset or

sublist, etc.) can be defined on ordered lists of features in a similar way. With the symbol + we denote the concatenation between ordered lists of features.

Let l be an ordered list of features. The notation $[l]$ represents a unitary list composed by the list l. Hereafter, by list we will understand an ordered list.

Definition 1. Let $l = [X_1,\ldots,X_s]$ be a feature list. We call *acceptance mask of l*, denoted as am_l, to the binary u-tuple in which the ith element is 1 if the ith row in *BM* has at least a 1 in the columns corresponding to the features of l and it is 0 otherwise.

Definition 2. Let $l = [X_1,\ldots,X_s]$ be a feature list. We call *compatibility mask of l*, denoted as cm_l, to the binary u-tuple in which the ith element is 1 if the ith row in *BM* has an only 1 in the columns corresponding to the features of l and it is 0 otherwise.

Notice that the cardinal of both am_l and cm_l is the number of rows in *BM*.

Example 1. Let $l_1 = [X_1, X_2]$, $l_2 = [X_5,X_6,X_7,X_8,X_9]$ and $l_3 = [X_1, X_2, X_8]$ be feature lists of a basic matrix *BM*. Its acceptance and compatibility masks are the following:

$$BM = \begin{pmatrix} X_1 & X_2 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{aligned} am_{l_1} &= (1,1,0,0,1) \\ am_{l_2} &= (1,1,1,1,1) \\ am_{l_3} &= (1,1,0,1,1) \\ cm_{l_1} &= (1,1,0,0,1) \\ cm_{l_2} &= (1,1,0,0,0) \\ cm_{l_3} &= (0,1,0,1,1) \end{aligned}$$

Proposition 1. A feature list $l = [X_1,\ldots,X_s]$ is a testor if and only if $am_l = (1,\ldots,1)$.

Definition 3. Let $l = [X_1,\ldots,X_s]$ be a feature list and $X \in \Re$. A row p in *BM* is a *typical row of X with respect to l* if it has a 1 in the column corresponding to X and zero in all the columns corresponding to the features in $l \setminus [X]$.

Notice that, by typical testor definition, a feature list l is a typical testor if l is a testor and satisfies the typicity property, i.e. for every feature $X \in l$ there is at least a typical row of X with respect to l.

Proposition 2. Let $l = [X_1,\ldots,X_s]$ be a feature list and $X \notin l$ a feature of *BM*. The acceptance mask of the list $l + [X]$ is calculated as follows: $am_{l+[X]} = am_l \vee c_X$.

Proposition 3. Let $l = [X_1,\ldots,X_s]$ be a feature list and $X \notin l$ a feature of *BM*. The compatibility mask of the list $l + [X]$ is calculated as follows:

$$cm_{l+[X]} = ((cm_l \otimes c_X) \wedge cm_l) \vee (\neg am_l \wedge c_X)$$

Notice that propositions 2 and 3 allow the updating of acceptance and compatibility masks, respectively when a new feature is added to a feature list.

Proposition 4. Let $l = [X_1,\ldots,X_s]$ be a feature list and $X \notin l$ a feature of *BM*. If at least one of the following conditions is satisfied:

1. $am_{l+[X]} = am_l$
2. $\exists X_i \in l$ such that $cm_{l+[X]} \wedge c_{X_i} = (0,\ldots,0)$

Then, X does not form a typical testor with l. In this case, we will say that X is *exclusive with l*.

The condition 1 means that X has no typical rows with respect to l and the second one indicates that X_i loses all its typical rows due to X.

Notice that X_6 is exclusive with l_1 in the Example 1, since X_2 holds that $cm_{l_1+[X_6]} \wedge c_{X_2} = (1,0,1,0,1) \wedge (0,1,0,0,0) = (0,0,0,0,0)$. Notice also that X_8 is non-exclusive with l_1, because $am_{l_1+[X_8]} \neq am_{l_1}$ ($l_3 = l_1+[X_8]$), $cm_{l_3} \wedge c_{X_1} = (0,0,0,0,1)$ and $cm_{l_3} \wedge c_{X_2} = (0,1,0,0,0)$.

Proposition 5. Let $l = [X_1,\ldots,X_s]$ be a feature list and $X \notin l$ a feature of *BM*. $l + [X]$ is a typical testor if and only if X is non-exclusive with l and $am_{l+[X]} = (1,\ldots,1)$.

The first condition means that all features of l+[X] have at least a typical row with respect to l+[X]. The second one guarantees that l+[X] is a testor, by proposition 1.

Definition 4. Let $l = [X_1,\ldots,X_s]$ be a feature list, p an integer such that $1 \leq p \leq s+1$ and $X \notin l$ a feature of *BM*. We call *substitution of X in l according to p*, denoted as *subst*(l,X,p), to the list $l' = [X_1,\ldots,X_{p-1},X]$. If $l = []$ then *subst*(l,X,1) = [X].

Notice that if $p = s+1$, *subst*(l,X,p) is the list l+[X].

Definition 5. Let $l=[X_{i_1},\ldots,X_{i_p}]$ and $l'=[X_{j_1},\ldots,X_{j_q}]$ be feature lists such that $l \cap l' = []$. We call *non-exclusive list of l with respect to l'*, denoted as *nonExcl*(l,l'), to the list composed by the features $X_{i_k} \in l$ such that X_{i_k} is non-exclusive with l' and $l' + [X_{i_k}]$ is not a typical testor.

For instance, in the basic matrix of Example 1, *nonExcl*(l_2, l_1) = [X_7,X_8]. Notice that X_5 and X_9 are non-exclusive with l_1, but [X_1, X_2, X_5] and [X_1, X_2, X_9] are typical testors.

Definition 6. Let $l=[X_{i_1},\ldots,X_{i_p}]$ and $l'=[X_{j_1},\ldots,X_{j_q}]$ be feature lists such that $l \cap l' = []$. We call *typical list of l with respect to l'*, denoted as *TypL*(l,l') to the list composed by the lists $l' + [X_{i_k}]$ such that $X_{i_k} \in l$ and $l' + [X_{i_k}]$ is a typical testor.

For instance, in the basic matrix of Example 1, *TypL*(l_2,l_1) = [[X_1,X_2,X_5],[X_1,X_2,X_9]].

3 BR Method

The proposed method firstly rearranges the rows and columns of *BM* in order to reduce the search space of typical testors. The row with the minimum number of 1's and the maximum number of 1's in the columns of *BM* where it has a 1 is put as the first row (see Steps 1a and 1b). In the Example 1, the two first rows have two 1's, but the first row stays there, since it has four 1's in the columns where it has a 1 (X_1, X_8). The rearrangement of columns (see Step 1c) allows the algorithm finishes as soon as possible, i.e., when the feature to be analyzed has a zero in the first row of *BM*. Notice that all possible combinations of the remaining features will not be testors. The rearrangement of columns also attempts to reduce the likelihood of the features to be analyzed being non-exclusive with a feature list, and therefore, to minimize the length of the feature lists that must be examined.

The underlying idea of *BR* method is firstly to generate feature lists that satisfy the typicity property and secondly to verify the testor condition. Like *LEX* and *CT-EXT* algorithms, our method explores the power set of features starting from the first feature in *BM* and generates candidate feature lists to be typical testors. Once a candidate feature list has been generated, the typicity and testor properties are verified by using propositions 1, 4 and 5. Notice that these propositions are based on acceptance and compatibility masks.

Given a candidate feature list L, *BR* method builds the list *LP* composed by the features X_i that are non-exclusive and do not form a typical testor with L. It means that $L+[X_i]$ needs more features to form a typical testor. Unlike previous algorithms, which attempt to find these features in *BM*, our method restricts the search to the features in *LP*. This fact is based on the following proposition:

Proposition 6. Let $l=[X_1,\ldots,X_s]$ be a feature list and $X \notin l$ a feature of *BM*. If X is exclusive with l, then it will also be exclusive with any list l', such that $l \subseteq l'$ and $X \notin l'$.

Notice that the features X_i that form a typical testor with L are not included in *LP*. In this case, these typical testors are stored in *TTR*. Then, the first feature in *LP* is added to L and the remaining features in *LP* that are non-exclusive with L are selected again. This process is repeated until all typical testors containing the first feature in *BM* are found. Then, the algorithm starts from the second feature in *BM* and repeats all steps until the feature to be analyzed has a zero in the first row of *BM* (see Step 3c). Notice that the process of generating candidate feature lists and removing features from the lists is recursive (*TL* acts as a stack in which feature lists are added or removed in order to be reused in the analysis of new feature combinations).

The proposed method is described as follows:

Input: A basic matrix *BM*.
Output: The set of all typical testors of *BM*.

1. Sorting rows and columns of *BM*:
   a. Let *F* be the set of rows that have the minimum number of 1's.
   b. For each row $f \in F$ obtain the number of 1's in all columns of *BM* that contain a 1 in *f*. Put the row with the maximum number as the first row in *BM*. If there is more than one row with the maximum value, then take any one of them.
   c. Let C^1 (C^0) be the set of columns with a 1 (0) in the first row of *BM*. Rearrange the columns such that columns in C^1 are on the left and columns in C^0 are on the right. Sort, in descending order, the columns in C^1 according to its number of 1's. The columns in C^0 are sorted in the same way.
2. Initialization:
   a. $L = []$
   b. Let *TTR* be the list of typical testors, $TTR = []$. Notice that *TTR* is a list of lists.
   c. Let *R* be the list of all features in *BM* and $TL = [R]$. Notice that *TL* is also a list of lists.
3. Process:
   a. Let *RL* be the last list of features in *TL*, i.e. $RL = last(TL)$.
   b. Let *X* be the first feature of *RL*.
   c. If $|TL| = 1$ then
      If the column corresponding to *X* (c_X) has a zero in the first row of *BM*, then return *TTR* and *END*
      else, if $c_X = (1,\ldots,1)$ then $TTR = TTR + [[X]]$, $RL = RL \setminus [X]$ and go to Step 3b.
   d. $L = subst(L, X, |TL|)$
   e. Remove the last element (list) from *TL*, i.e. $TL = TL \setminus [last(TL)]$.
   f. $RL = RL \setminus [X]$
   g. $LP = nonExcl(RL, L)$
   h. $TTR = TTR + TypL(RL, L)$
   i. If $|RL| > 1$, then
      $TL = TL + [RL]$
      If $|LP| > 1$, then $TL = TL + [LP]$
   j. Go to Step 3.

Notice that the list *LP* includes the features of *RL* that are non-exclusive with *L* but do not form typical testors with *L,* whereas *TypL*(*RL*, *L*) contains the features of *RL* that constitute typical testors with the features in *L*. Then, the Steps 3g and 3h can be performed simultaneously as follows:

For each feature *X'* of *RL*:
1. Calculate $am_{L+[X']}$ from am_L by using the Proposition 2.
2. If $am_{L+[X']} \neq am_L$ (see condition 1 of Proposition 4) then
   a. Calculate $cm_{L+[X']}$ from cm_L by using the Proposition 3.
   b. If $cm_{L+[X']} \wedge cx_i \neq (0,\ldots,0)\ \forall X_i \in L$ (see condition 2 of Proposition 4) then
      If $am_{L+[X']} = (1,\ldots,1)$ then
         Add $L+[X']$ to *TTR* ($L+[X']$ is a typical testor by Proposition 5)
      else, Add *X'* to *LP*

The characteristics that allow *BR* method to avoid the analysis of irrelevant feature subsets are the following:

- The algorithm directly examines the feature combinations generated from the features in *L* and those belonging to *LP* (non-exclusive ones with *L*), avoiding the analysis of the remaining combinations.
- Since features that constitute a typical testor with *L* are not included in *LP*, the algorithm disregards all supersets of a typical testor.

CT-EXT method firstly generates testors and secondly verifies the typicity property, whereas *LEX* and *BR* methods firstly generate feature subsets satisfying the typicity property and then, verify the testor condition. *CT-EXT* attempted to reduce the cost of verifying the typicity property of *LEX*, but at expense of generating a greater number of feature subsets.

Example 2. The following table shows the feature subsets generated by *LEX*, *CT-EXT* and *BR* methods for the basic matrix *BM* until the subset $\{X_2,X_6\}$ (represented as 26 in the table) is generated. Typical testors are highlighted in boldface. As we can see, *BR* generates the least number of feature subsets. Notice that $LP=[X_3,X_4]$ when the first 6 subsets are generated. Therefore, *BR* can jump to $\{X_1,X_3,X_4\}$ disregarding the remaining combinations that include X_1. However, *LEX* is not able to disregard these combinations. On the other hand, *CT-EXT* examines several subsets including X_1 and X_2 even though neither of them constitutes a typical testor. The definition of that a feature *X* contributes to a subset [8] only verifies that *X* has at least a typical row, but disregards that features in the subset can lose its typical rows due to *X*. Notice that X_2 contributes to $\{X_1\}$, but X_1 lost its typical row (the first one) due to X_2.

LEX		*CT-EXT*			*BR*	
1	146	1	135	**23**	1	**23**
12	**15**	12	136	**25**	12	24
13	**16**	123	14	**26**	13	**25**
134	2	124	145		14	**26**
135	**23**	125	146		**15**	
136	24	126	**15**		**16**	
14	**25**	13	**16**		**134**	
145	**26**	**134**	2		2	

$$BM = \begin{pmatrix} X_1 & X_2 & X_3 & X_4 & X_5 & X_6 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

4 Experimental Results

In order to evaluate the performance of the proposed method, we compare the time spent to obtain all typical testors by our method and the two best algorithms reported in the literature: *LEX* and *CT-EXT*. It is worth mentioning that the source code of the *CT-EXT* algorithm was provided by the authors. To ensure a fair comparison all methods are carried out on an Intel Pentium Dual Core 1.6 GHz, 1 Gb RAM.

For this comparison we use five data sets obtained from UCI Machine Learning Repository[1]. For each one, we generated the basic matrices regarding the strict equality as comparison criterion for all features. Table 1 shows the run time of the methods for the basic matrices of real data sets and two basic matrices randomly generated. Notice that these matrices have different dimensions (see Column 3). The last column (NTT) indicates the number of calculated typical testors.

Table 1. Run times (h:m:s:ms) of the algorithms for several basic matrices

Data set	Class	*BM*	*LEX*	*CT-EXT*	*BR*	NTT
Zoo (101 x 17)	7	14 x 17	0:0:00:15	0:0:0:718	0:0:00:00	34
Mushroom (8124 x 22)	2	30 x 22	0:0:00:16	0:0:0:750	0:0:00:00	292
Chess (3196 x 36)	2	29 x 36	0:2:22:16	0:8:01:67	0:0:00:12	4
Dermatology (366 x 34)	6	1124 x 34	0:25:45:7	1:43:15:6	0:0:58:22	115556
Promoter (106 x 57)	2	2761 x 57	1:07:27:5	4:24:23:8	0:3:18:51	7456943
Random	-[2]	150 x 70	0:55:45:3	2:06:30:4	0:4:02:67	44165054
Random	-[2]	100 x 100	2:22:01:9	> 20 hrs	0:10:30:1	183051234

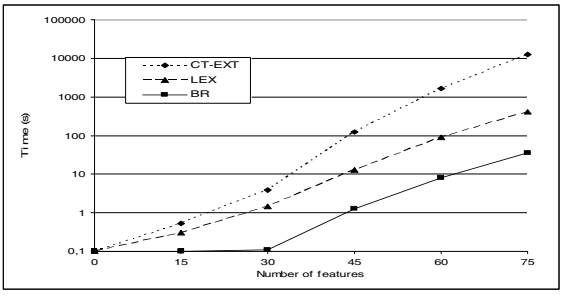

Fig. 1. Run times (in seconds) for basic matrices of 50 rows varying the number of features

As we can observe, the higher dimension of basic matrix, the greater time is needed to calculate typical testors in all methods. It is important to notice also that *BR* achieves considerable time reductions with respect to *LEX* and *CT-EXT*. Unlike the reported results in [8], our experiments revealed that *CT-EXT* actually performs worse than *LEX*.

In order to study the behavior of the algorithms, we show in Figure 1 the run times (in seconds) of the methods for basic matrices of 50 rows varying the number of

[1] http://archive.ics.uci.edu/ml/

[2] The number of classes is disregarded, because we randomly generate matrices of 0's and 1's.

features from 15 to 75. As we expected, the time of all methods grows exponentially when the number of features is increased. However, notice that our method runs about 10 times faster than the best competitor, *LEX* and about 100 times faster than *CT-EXT*.

Thus, we can conclude that *BR* is significantly more efficient than other algorithms.

5 Conclusions

In this paper, a new external scale algorithm *BR* to calculate all typical testors of a training matrix has been proposed. The experimental results demonstrate that this method significantly outperforms the two best algorithms reported in the literature. The main contributions that ensure the speed up in the calculation of the set of all typical testors are: a new method for verifying typicity and testor properties which is based on binary logic and profits from the computer bit operations, the introduction of a generation mechanism of candidate feature subsets that avoids the analysis of a greater number of irrelevant subsets, and a prior ordering of the basic matrix that guarantees that the method finishes as soon as possible.

Future work includes extending our method in order to obtain other generalizations of the typical testors not restricted to Zhuravlev's testors (e.g. ε-testors and fuzzy testors [2]). We also plan to conduct additional experiments with basic matrices of different densities to evaluate the performance of the proposed method.

References

1. Martínez-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)
2. Lazo-Cortés, M., Ruiz-Shulcloper, J., Alba-Cabrera, E.: An overview of the evolution of concept testor. Pattern Recognition 34(4), 753–762 (2001)
3. Ortiz-Posadas, M.R., Martínez-Trinidad, J.F., Shulcloper, J.R.: A new approach to diferential diagnosis of diseases. Int. J. Biomed. Compu. 40(3), 179–185 (1996)
4. De la Vega-Doria, L.A., Carrasco-Ochoa, J.A., Shulcloper, J.R.: Fuzzy KORA-W algorithm. In: 6th European Congress on Intelligent Techniques and Soft Computer, Aachen, Germany, pp. 1190–1194 (1998)
5. Pons-Porrata, A., Gil-García, R., Berlanga-Llavori, R.: Using Typical Testors for Feature Selection in Text Categorization. In: Rueda, L., Mery, D., Kittler, J. (eds.) CIARP 2007. LNCS, vol. 4756, pp. 643–652. Springer, Heidelberg (2007)
6. Pons-Porrata, A., Ruiz-Shulcloper, J., Berlanga-Llavori, R.: A Method for the Automatic Summarization of Topic-Based Clusters of Documents. In: Sanfeliu, A., Ruiz-Shulcloper, J. (eds.) CIARP 2003. LNCS, vol. 2905, pp. 596–603. Springer, Heidelberg (2003)
7. Santiesteban-Alganza, Y., Pons-Porrata, A.: LEX: a New Algorithm for Computing Typical Testors. Revista Ciencias Matemáticas 21(1), 85–95 (2003)
8. Sánchez Díaz, G., Lazo Cortés, M.: CT-EXT: An Algorithm for Computing Typical Testor set. In: Rueda, L., Mery, D., Kittler, J. (eds.) CIARP 2007. LNCS, vol. 4756, pp. 506–514. Springer, Heidelberg (2007)