A New Editing Scheme Based on a Fast Two-String Median Computation Applied to OCR

José Ignacio Abreu Salas[1] and Juan Ramón Rico-Juan[2]

[1] Universidad de Matanzas, Cuba
jose.abreu@umcc.cu
[2] Dpto Lenguajes y Sistemas Informáticos, Universidad de Alicante, Spain
juanra@dlsi.ua.es

Abstract. This paper presents a new fast algorithm to compute an approximation to the median between two strings of characters representing a 2D shape and its application to a new classification scheme to decrease its error rate. The median string results from the application of certain edit operations from the minimum cost edit sequence to one of the original strings. The new dataset editing scheme relaxes the criterion to delete instances proposed by the Wilson Editing Procedure. In practice, not all instances misclassified by its near neighbors are pruned. Instead, an artificial instance is added to the dataset expecting to successfully classify the instance on the future. The new artificial instance is the median from the misclassified sample and its same-class nearest neighbor. The experiments over two widely used datasets of handwritten characters show this preprocessing scheme can reduce the classification error in about 78% of trials.

1 Introduction

Dataset editing has received considerable attention from the seminal works of Wilson [17] about the *edited near neighbor rule* (ENN) because this technique can be useful to improve *nearest neighbor classifiers* response. Mainly, these algorithms focus on deleting wrong tagged instances from a set, which will be used as training set for a given classifier. Several modifications have been proposed such as [2][4][13] and [15][18] more recently, facing with some problems of basic Wilson procedure as statistically dependence of estimations over each instance [10]. Another group of algorithms also changes some instances tag while editing, such [6] and [14].

In many problems, patterns do not have a vectorial representation, instead another syntactic coding such strings and trees are commonly used. Methods cited before mainly concern with these vectorial representations and distances. Therefore, when dealing with strings a suitable distance has be selected. In our case, the widely used Levenshtein edit distance [7] is choosen. In addition, some approaches find representatives instances as centroids [12] or prototypes, so this concepts need to be extended to the new coding schemes.

Several authors [1] [3] [5] have described algorithms to get a prototype representing a set of strings, as the centroid, the median string or an approximation. Most of this works build the desired string by successive refinements of a initial string or by some *ad-hoc* procedure.

This work proposes new Wilson based approach to edit a dataset of instances that have been encoded by some string representation. The inclusion of a prototype representing both, an instance and its same-class nearest neighbor inside a k-neighborhood if exist, when the instance is misclassified, is the main difference from others described in the literature. Besides, this paper presents a fast algorithm to compute the prototype representing two strings suitable for requirements of described editing procedure. Section 2 provides a detailed explanation of the algorithm to build the prototype and some useful concepts. Section 3 describes the proposed editing procedure and several considerations related to the computational complexity of proposed algorithms. Finally, section 4 illustrates with different experiments the behavior of proposed methods.

2 Prototype Construction

To compute the prototype representing two strings, this case defined as the median string, the proposed approach focus on information gathered from calculation of the distance between those strings. This section contains a glance of the selected distance measure, the Levenshtein [7] edit distance. Latter a procedure to compute the median string is covered.

2.1 Edit Distance

Let Σ be an alphabet and $S_1 = \{S_{11}, S_{12}..S_{1m}\}$, $S_2 = \{S_{21}, S_{22}..S_{2n}\}$ two strings over Σ where $m, n \geq 0$, the edit distance between S_1 and S_2, $D(S_1, S_2)$, is defined in terms of elementary edit operations which are required to transform S_1 into S_2. Usually three edit operations are considered:

- *substitution* of a symbol $a \in S_1$ by a symbol $b \in S_2$, denoted as $w(a, b)$
- *insertion* of a symbol $b \in S_2$ in S_1, denoted as $w(\varepsilon, b)$
- *deletion* of a symbol $a \in S_1$, denoted as $w(a, \varepsilon)$.

where ε denotes an empty string. Let $Q^{Sj}_{Si} = \{q_1, q_2, ..., q_k\}$ be a sequence of edit operations transforming S_i into S_j, if each operation has cost $e(q_i)$ the cost of Q^{S1}_{S2} is $E_{Q^{S1}_{S2}} = \sum_{i=1}^{k} e(q_i)$ and the edit distance $D(S_1, S_2)$ is defined as:

$$D(S_1, S_2) = argmin\{ E_{Q^{S1}_{S2}}\} .$$

Strings involved in this work are Freeman Chaincodes, for that reason substitution costs are computed as follows:

$$e(w(a, b)) = argmin\{|a - b|, 8 - |a - b|\}$$

In the case of the insertions and deletions the cost 2 was chosen which is half of the maximum cost of the substitution operation; this same fixed number is used in [11]. The dynamic programming algorithm exposed by Wagner [16] allows to compute $D(S_1, S_2)$ in $\mathcal{O}(L_{S1} \times L_{S2})$ time, where L_S denotes the length of string S.

2.2 Fast Median String Computation

The median of a set T of strings can be briefly defined as the string R which realizes:

$$argmin_R\{\sum D(R,S_i)|S_i \in T\} \tag{1}$$

As explained above, the proposed approach computes the median R of two strings S_1 and S_2 by applying a subset of edit operations from the minimum cost edit sequence Q_{S2}^{S1} to S_1. To choose those editions that will be applied, each element at Q_{S2}^{S1} is tested to estimate how it will affect $D(S_1,R)$ and $D(S_2,R)$, since the algorithm seeks for a string R satisfying (1) and (2). This additional requirement means that an R near of the halfway between S_1 and S_2 will be preferred.

$$argmin_R\{|D(R,S_1) - D(R,S_2)|\} \tag{2}$$

The idea behind the algorithm is that each operation q_i in Q_{S2}^{S1} affects the future $D(R,S_1)$ and $D(R,S_2)$ since can be guessed that a rejected operation keeps R similar to S_1 while accepted editions makes R resembles to S_2. A close examination of each possible operation help to explain this conjecture.

For insertions, let b_{S2}^k be the k-esime symbol from S_2. An operation $q_i = w(\varepsilon, b_{S2}^k)$ from $Q_{S_1}^{S2}$ indicates the insertion of this symbol into S_1 to obtain R. Suppose q_i is accepted, thus can be expected that Q_R^{S2} does not involves an insertion of a symbol in R to be matched with b_{S2}^k in S_2 since it was done before. A similar reasoning led to guess this symbol is market to deletion from R when $D(R,S_1)$ is computed. In turn, a symbol $b_{S_1}^k$ deleted from S_1 to get R will be pointed to be inserted again while computing distance from R to S_1.

Substitutions $w(b_{S_1}, b_{S_2})$ will always applied, but whenever possible a symbol m will be placed in R instead b_{S_1} or b_{S_2}. The choice of m tries to make R similar to both S_1 and S_2, thus must satisfies:

$$e(w(b_{S_1}, b_{S_2})) = e(w(b_{S_1}, m)) + e(w(m, b_{S_2})). \tag{3}$$

$$argmin_m\{|e(w(m, b_{S_1})) - e(w(m, b_{S_2}))|\}. \tag{4}$$

Previous assumptions allow estimating how applying or not an operation q_i will affect distances from R to S_1 and S_2. Chosen insertions of b_{S2} into S_1 contributes with $e(w(b_{S2},\varepsilon))$ to $D(R,S_1)$ since the inserted symbol need to be deleted, now q_i has not an effect on $D(R,S_2)$. If the insertion is rejected implies $D(R,S_1)$ does not change, but $D(R,S_2)$ will grow by $e(w(\varepsilon,b_{S2}))$. Unlike the insertion occurs in the case of the deletion operation. If a deletion is discarded $D(R,S_2)$ increases by $e(w(b_{S1},\varepsilon))$ or $D(R,S_1)$ by $e(w(\varepsilon,b_{S1})$, if operation is accepted. Substitutions make both $D(R,S_1)$ and $D(R,S_2)$ grow by $e(w(b_{S1},m))$ and $e(w(m,b_{S2}))$ respectively.

For example, let $S_1 = \{a,b\}$, $S_2 = \{d,e\}$ and $e(w(\cdot,\varepsilon)) = e(w(\varepsilon,\cdot)) = 1$. Table 1 shows the substitution cost between symbols, thus $Q_{S1}^{S2} = \{w(a,\varepsilon), w(b,d), w(\varepsilon,e)\}$. Possible options to select or not an operation yields the tree at Figure 1 where each leaf node shows a candidate R. Inside brackets, an estimation of the cumulative contribution of each operation up from the node to $D(R,S1)$ and $D(R,S2)$ respectively. Procedures

FMSC and *FindOp* outlined below allow searching through the tree for those edit operations which yields an R satisfying established requirements.

```
Let:
$Q_{S1}^{S2}$:minimum cost edit sequence to transform $S_1$ into $S_2$.
$d$: difference between cumulative $S1$ and $S2$.
$r$: the better consecutive symbols corresponding to $d$ difference.
function FindOp($op_i$, $a_{S1}$, $a_{S2}$) : ($d$, $r$)
/* $op_i$: index of the operation $op \in Q_{S1}^{S2}$ to analyze if is applied or not */
/* $a_{S1} = 0$: cumulative distance of applied editions over $D(R, S1)$ */
/* $a_{S2} = 0$: cumulative distance of applied editions over $D(R, S2)$ */
/* $better = (\infty, \emptyset)$: local better result */
if ($op_i == 0$) then
    $better = (a_{S1} - a_{S2}, \emptyset)$
else
    case $Q_{S1}^{S2}[op_i]$ :
      - $w(b_{S1}, \varepsilon, )$: /* Deletion */
            /* Rejected */
            $(d_{no}, r_{no})$ = FindOp($op_i - 1$, $a_{S1}$, $a_{S2} + e(w(b_{S1}, \varepsilon))$)
            /* Accepted */
            $(d_{yes}, r_{yes})$ = FindOp($op_i - 1$, $a_{S1} + e(w(\varepsilon, b_{S1}))$, $a_{S2}$)
            if ($|d_{yes}| < |d_{no}|$) then
                $better = (d_{yes}, r_{yes} \cup \{b_{S1}\})$
            else
                $better = (d_{no}, r_{no})$
            end if
      - $w(\varepsilon, b_{S2})$: /* Insertion */
            /* Rejected */
            $(d_{no}, r_{no})$ = FindOp($op_i - 1$, $a_{S1}$, $a_{S2} + e(w(\varepsilon, b_{S2}))$)
            /* Accepted */
            $(d_{yes}, r_{yes})$ = FindOp($op_i - 1$, $a_{S1} + e(w(b_{S2}, \varepsilon))$, $a_{S2}$)
            if ($|d_{yes}| < |d_{no}|$) then
                $better = (d_{yes}, r_{yes} \cup \{b_{S2}\})$
            else
                $better = (d_{no}, r_{no})$
            end if
      - $w(b_{S1}, b_{S2})$: /* Substitution */
            foreach symbol $m$ satisfying (3) and (4)
                $(d, r)$ = FindOp($op_i - 1$, $a_{S1} + e(w(m, b_{S1}))$, $a_{S2} + e(w(m, b_{S2}))$)
                if ($|d| < |better|$) then
                   $better = (d, r \cup \{m\})$
                end if
            end foreach
    end case
end if
return $better$
end function

procedure FMSC($S_1$,$S_2$)
/* $S_1$ and $S_2$: strings to compute its median $R$
  - compute $D(S_1, S_2)$ to get $Q_{S1}^{S2}$
  - $(d,r)$ = FindOp($L_{Q_{S1}^{S2}}$,0,0)
  - return $r$
end procedure
```

Table 1. Substitution cost between two symbols. $e(w(\cdot,\cdot))$

	a	b	c	d	e
a	0	1	2	3	4
b	1	0	1	2	3
c	2	1	0	1	2
d	3	2	1	0	1
e	4	3	2	1	0

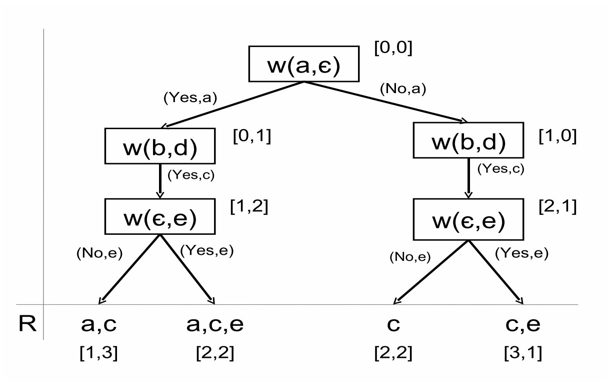

Fig. 1. Each branch represents a possible set of operations to get R from S_1

3 Editing Algorithm

Let T a set of instances. Wilson [17] based editing procedures such [4][13] remove all misclassified instances, t_i, by its k-nearest neighbors (K-NN). This kind of editing cleans interclass overlapping regions while the boundaries between classes are smoothed. A K-NN classifier that uses the edited set as training set could be improved its classification results respect the classification through the original dataset.

As point Wilson [18], in some cases editing has be done carefully because the algorithm may remove a lot of instances, while spoiling the generalization capabilities.

When $k \geq 1$, a wrong classification of t_i does not mean that no one k-nearest neighbors belongs to the same class of t_i. Thus, it is reasonable to think that t_i does not need to be an outlier, but can be a boundary instance useful for next classifications.

The proposed approach aims to face successfully this problem by adding to T an artificial instance R computed from t_i and its same-class nearest neighbor, t_j, if it belongs to k-nearest neighbors. If R, tagged as t_i, satisfies $D(R, t_i) \leq D(t_i, t_j)$ and (2). Its inclusion boosts the chance t_i will be correctly reclassified since by definition R lies in the t_i k-neighborhood. Moreover, this can be viewed as some space regions poorly covered from the original instances become better represented. From these assumptions can be guessed this editing scheme leads to lower classification errors versus the original dataset. Artificial instance R will be computed by the procedure **F**ast **M**edian **S**tring **C**omputation, FMSC for short, described at section 2.2, as $R = FMSC(t_i, t_j)$. The algorithm sketched below allows compute the edited set.

```
Let:
$T$:instance set to edit.
$K$:number of near neighbors.
foreach instance $t_i$ in $T$

 - classify $t_i$ by its k-near neighbors in $T - t_i$.
   if wrong classified then
    - find $t_j$, the k-near same-class neighbor of $t_i$.
      if exist $t_j$ then
       - build $R = FMSC(t_i, t_j)$.
       - make $T = T \cup R$.
      else
       - mark $t_i$ to deletion.
      end if
   end if

end foreach
- delete from $T$ all market instances.
```

3.1 Computational Cost Analysis

Computing the median R from strings S_1 and S_2 involves the calculation of $D(S_1, S_2)$, which can be accomplished in $\mathcal{O}(L_{S1} \times L_{S2})$ as was pointed at subsection 2.1. From definition of this distance $L_{Q_{S1}^{S2}} = L_{S1} + L_{S2}$ for the worst case, i.e when there are no substitutions.

Searching through the tree with *FindOp* can be viewed as evaluating all possibilities to assigning $\{accepted/rejected\}$ to every q_i in Q_{S1}^{S2}, so there are $2^{L_{Q_{S1}^{S2}}}$ chances, this is, the number of branches on the tree. Denoting a rejected operation by $\sim q_i$, let $Op = \{q_1, \sim q_2, \sim q_3, ..., q_n\}$ be a possible assignation and q_i, $i < n$, an arbitrary operation. Clearly, the estimate a_{S1} to $D(R, S_1)$ associated with Op can be decomposed as

$a_{S_1} = a_{S_1}^{0..i} + a_{S_1}^{i+1..n}$ where $a_{S_1}^{k..h}$ denotes the cumulative contribution of those operations $q_k, .., q_h$ to a_{S1}, this holds also for a_{S2}.

Be Op' a new assignment derived from Op by changing the $\{accepted/rejected\}$ tag to some operations $\{q_0, .., q_i\}$ while $\{q_{i+1}, .., q_n\}$ gets unchanged. Consequently the value a'_{S1} related to Op' is $a'_{S_1} = a'^{0..i}_{S_1} + a^{i+1..n}_{S_1}$, an expression which is partially calculated before, similarly can be computed a'_{S2}. Moreover, if those assignations to $\{q_{i+1}, .., q_n\}$ minimizes $|a_{S_1}^{i+1..n} - a_{S_2}^{i+1..n}|$ thus, the optimal sequence of assignations is one which have $\{q_{i+1}, .., q_n\}$ as subsequence.

Considerations above allows to speed up computation of *FindOp* procedure by applying a dynamic programming approach leading a $\mathcal{O}(max\{L_{S1} \times L_{S2}, L_{Q_{S1}^{S2}} \times D_s\})$, where $D_s = D(S_1, S_2)$.

The editing procedure needs to classify every instance at T, which requires an $\mathcal{O}(|T|^2)$ time. A second steps involves computing *FMSC* for every wrong classified instance having a same-class k-near neighbor. The worst case, all $|T|$ instances will need to be processed, so this step entails $\mathcal{O}(|T| \times$ *FindOp*$)$ time.

4 Experimental Results

The behavior of the proposed algorithms was analyzed using two sets of strings (Freeman Chaincodes). Digits and character contours from the NIST 3 DATABASE with 26 and 10 classes respectively.

To evaluate the editing algorithm a sample of 80 instances per class was drawn and each set splits in 4-fold to use a crossvalidation technique. At a first stage, for a fixed value of K, all training sets were edited by the Wilson procedure and each test set classified by the K-NN rule using the respective edited set. Latter, each original training set was edited but this time by our proposed approach classifying again the test sets. As

Table 2. Average error rate (4-folds) as percent for classification with different edited sets. (Characters set).

		K on Edition															
		K=3		K=5		K=7		K=9		K=11		K=13		K=15		K=17	
K on Classif.	Not Edited	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson
1	13.7	16.5	14.6	15.8	13.8	16.3	13.8	16.8	13.8	17.1	**13.4**	17.1	**13.4**	17.5	**13.2**	17.8	**13.1**
3	14.7	17.6	15.3	17.6	14.6	17.5	**14.4**	17.8	**14.2**	18.8	**14.0**	18.9	**13.9**	19.6	**13.8**	19.9	**13.3**
5	15.4	17.6	**15.2**	17.9	**14.4**	17.9	**14.1**	18.5	**14.1**	18.9	**14.2**	19.5	**14.1**	19.8	**13.8**	20.2	**13.7**
7	16.0	19.4	16.2	19.6	**15.1**	19.9	**15.2**	19.8	**14.7**	20.2	**14.2**	20.7	**14.0**	20.8	**14.0**	21.3	**14.1**
9	17.1	19.5	**16.3**	20.1	**15.5**	20.3	**15.3**	20.5	**14.8**	21.0	**14.8**	21.6	**14.5**	21.8	**14.6**	22.2	**14.6**
11	17.7	20.0	17.7	20.8	**16.5**	20.8	**16.1**	21.3	**15.4**	21.6	**15.0**	22.3	**15.0**	22.3	**15.2**	23.1	**15.0**
13	18.3	21.0	**18.2**	21.2	**17.1**	21.7	**16.4**	22.1	**16.5**	22.5	**15.8**	22.8	**15.5**	23.3	**15.5**	23.7	**15.7**
15	18.6	22.0	18.9	21.6	**18.0**	22.3	**17.1**	22.6	**16.7**	23.3	**16.2**	23.5	**16.3**	23.7	**16.0**	24.2	**16.0**
17	19.6	22.1	**18.8**	22.7	**18.0**	23.0	**17.5**	23.8	**17.4**	24.0	**16.9**	24.0	**16.5**	24.6	**16.5**	24.8	**16.4**

Table 3. Average error rate (4-folds) as percent for classification with different edited sets. (Digits set).

		K on Edition															
		K=3		K=5		K=7		K=9		K=11		K=13		K=15		K=17	
K on Classif.	Not Edited	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson	Wilson	JJWilson
1	1.8	2.8	1.9	2.6	1.8	2.5	1.8	2.9	**1.6**	2.8	**1.6**	2.8	**1.6**	2.8	**1.5**	2.8	**1.5**
3	2.0	3.1	2.3	2.9	2.3	3.0	2.0	3.3	**1.9**	3.4	2.0	3.4	2.0	3.8	**1.9**	3.9	**1.8**
5	3.0	3.6	**2.9**	3.6	**2.8**	3.8	**2.8**	4.3	**2.6**	4.3	**2.6**	4.1	**2.5**	4.3	**2.3**	4.4	**2.5**
7	3.5	4.3	3.5	4.3	**2.9**	4.3	**2.9**	4.5	**2.6**	4.6	**2.6**	4.8	**2.6**	5.0	**2.5**	5.1	**2.4**
9	3.6	4.3	**3.5**	4.1	**3.3**	4.3	**3.0**	4.6	**2.9**	4.9	**2.8**	4.9	**2.8**	5.1	**2.8**	5.4	**2.8**
11	4.1	4.5	**2.9**	4.5	**2.9**	4.6	**2.9**	4.6	**3.0**	4.6	**3.6**	4.8	**3.5**	4.9	**3.5**	5.3	**3.4**
13	4.4	4.8	**3.1**	4.8	**3.3**	5.0	**3.1**	5.0	**3.3**	5.3	**4.0**	5.6	**3.5**	5.9	**3.5**	5.9	**3.4**
15	4.8	5.1	**3.8**	5.1	**3.8**	5.4	**3.6**	5.5	**3.8**	5.5	**4.5**	6.1	**4.3**	6.3	**4.1**	6.3	**3.9**
17	4.9	5.1	**4.1**	5.1	**4.0**	5.4	**3.8**	5.5	**3.9**	5.5	**4.6**	6.1	**4.2**	6.0	**4.3**	6.1	**4.3**

a baseline, the original training sets classficantion is used. At each fold, editing was repeated for odds values of K from 3 to 17, while in the classification stage, the range was from 1 to 17. Remaining a total of 288 trials on each dataset. As distance, the Levenshtein distance was chosen which is described in the section 2.1.

Tables 2 and 3 show some results for the 4-fold experiments when the test set uses: the original training set, different edited sets for the characters and the digits datasets, respectively. Through these experiments, Wilson procedure never reduces the baseline error rate (classification with original tranining sets), while our proposed approach, labeled as JJWilson, is to able to improve by 79.1% of the trials in the case of characters dataset and by 77.7% in the case of digits dataset. These improvements are highlighted in bold type in the tables of results. So, the experiments in both datasets show that the proposed algorithm for editing outperforms the Wilson approach, with respect to the error rate reduction.

5 Conclusions and Future Work

A novelty method was presented to edit a dataset of contours encoded by Freeman Chaincodes. In addition, a new fast procedure to compute the median between two strings based on a string edit distance is explained. Experiments show that the edit scheme behaves well on the studied datasets. Further investigations can be addressed to revise the method to identify the misclassified instance, and consider other near neigbor belonging to the same class instead the nearest one to build the new prototype. Also, others datasets could be studied and compared with additional edit methods. Moreover, our fast median string algorithm between two examples could be extended to compute the average of N examples.

Acknowledgements

This work is partially supported by the Spanish CICYT under project DPI2006-15542-C04-01, the Spanish MICINN through project TIN2009-14205-CO4-01 and by the Spanish research programme Consolider Ingenio 2010: MIPRCV (CSD2007-00018).

References

1. Cárdenas, R.: A Learning Model for Multiple-Prototype Classification of Strings. In: 17th International Conference on Pattern Recognition, vol. 4, pp. 420–442 (2004)
2. Devijver, I., Kittler, J.: On the edited nearest neighbour rule. In: 5th Int. Conf. on Pattern Recognition, pp. 72–80 (1980)
3. Duta, N., Jain, A., Dubuisson-Jolly, M.: Automatic Construction of 2D Shape Models. IEEE Transactions on Pattern Analysis and Machine Intelligence 23, 433–446 (2001)
4. Ferri, F., Vidal, E.: Comparison of several editing and condensing techniques for colour image segmentation and object location. Pattern Recognition and Image Analysis (1992)
5. Jiang, X., Schiffmann, L., Bunke, H.: Computation of median shapes. In: 4th Asian Conference on Computer Vision (2000)
6. Koplowitz, J., Brown, T.: On the relation of performance to editing in nearest neighbour rules. Pattern Recognition 13, 251–255 (1981)
7. Levenshtein, V.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics 10, 707–710 (1966)
8. Martínez, C., Juan, A., Casacubierta, F.: Median strings for k-nearest neighbour classification*1. Pattern Recognition Letters 24, 173–181 (2003)
9. Olvera, J., Martínez, F.: Edition schemes based on BSE. In: 10th Iberoamerican Congress on Pattern Recognition, pp. 360–368 (2005)
10. Penrod, C., Wagner, T.: Another look at the edited neares neighbour rule. IEEE Trans. on Systems, Man and Cybernetics 7, 92–94 (1977)
11. Rico-Juan, J.R., Micó, L.: Comparison of AESA and LAESA search algorithms using string and tree-edit-distances. Pattern Recognition Letters 24, 1417–1426 (2003)
12. Sánchez, J., Pla, F., Ferri, F.: Using the nearest centroid neighbourhood concept for editing purposes. In: 7th Symposium National de Reconocimiento de Formas y Análisis de Imágen, vol. 1, pp. 175–180 (1997)
13. Tomek, I.: An experiment with the edit nearest neighbour. IEEE Trans. on Systems, Man and Cybernetics 6, 448–452 (1976)
14. Tomek, I.: A generalization of the k-NN rule. IEEE Trans. on Systems, Man and Cybernetics 6, 121–126 (1976)
15. Vázquez, F., Sánchez, J., Pla, F.: A stochastic approach to Wilson's editing algorithm. In: Marques, J.S., Pérez de la Blanca, N., Pina, P. (eds.) IbPRIA 2005. LNCS, vol. 3523, pp. 35–42. Springer, Heidelberg (2005)
16. Wagner, R., Fischer, M.: The String-to-String Correction Problem. Journal of the ACM 21, 168–173 (1974)
17. Wilson, D.: Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans. on Systems, Man. and Cybernetics 2, 408–421 (1972)
18. Wilson, D., Martínez, T.: Reduction techniques for instance based learning algorithms. Machine Learning 38, 257–286 (2000)