Fast *k* Most Similar Neighbor Classifier for Mixed Data Based on a Tree Structure and Approximating-Eliminating

Selene Hernández-Rodríguez, J.A. Carrasco-Ochoa,
and J. Fco. Martínez-Trinidad

Computer Science Department
National Institute of Astrophysics, Optics and Electronics
Luis Enrique Erro No. 1, Sta. María Tonantzintla, Puebla, CP: 72840, México
{selehdez,ariel,fmartine}@ccc.inaoep.mx

Abstract. The *k* nearest neighbor (*k-NN*) classifier has been extensively used as a nonparametric technique in Pattern Recognition. However, in some applications where the training set is large, the exhaustive *k-NN* classifier becomes impractical. Therefore, many fast *k-NN* classifiers have been developed to avoid this problem. Most of these classifiers rely on metric properties, usually the triangle inequality, to reduce the number of prototype comparisons. However, in soft sciences, the prototypes are usually described by qualitative and quantitative features (mixed data), and sometimes the comparison function does not satisfy the triangle inequality. Therefore, in this work, a fast *k* most similar neighbor (*k-MSN*) classifier, which uses a Tree structure and an Approximating and Eliminating approach for Mixed Data, not based on metric properties (Tree AEMD), is introduced. The proposed classifier is compared against other fast *k-NN* classifiers.

Keywords: k-NN Classifier, Fast k-NN Classifiers, Mixed Data.

1 Introduction

The *k-NN* [1] classifier has been extensively used as a nonparametric technique in Pattern Recognition. In order to decide the class of a new prototype, the *k-NN* classifier performs an exhaustive comparison between the prototype to classify (query) and the prototypes in the training set (*T*). However, when *T* is large, the exhaustive comparison is expensive. To avoid this problem, many fast *k-NN* classifiers have been developed. Some of these classifiers are exact methods, because they find the same *k-NN* that would be found using the exhaustive search. Some others are approximate methods, because they do not guarantee to find the *k-NN* to a query prototype among the training set, but they find an approximation faster than the exact methods.

To avoid prototype comparisons during the search of the *k-NN*, different approaches such as: approximating-eliminating [2-6] and tree-based [5-9] have been developed.

J. Ruiz-Shulcloper and W.G. Kropatsch (Eds.): CIARP 2008, LNCS 5197, pp. 364–371, 2008.

Tree-based classifiers consist of two phases: preprocessing and classification. In the preprocessing phase, the prototypes in *T* are organized in a tree structure. In the classification phase, the tree is traversed to find the *k-NN*. The speed up is obtained while the exploration of some parts of the tree is avoided, applying pruning rules derived from the triangle inequality property. One of the first fast *k-NN* classifiers, that uses a tree structure, was proposed by Fukunaga and Narendra [7]. Some improvements, over the pruning rule used in the Fukunaga and Narendra's (FN) classifier, have been proposed by Moreno-Seco et al. (MS classifier) [8] and Oncina et al. (ONC classifier) [9].

The first classifier based on the approximating-eliminating approach was AESA (Approximating Eliminating Search Algorithm), proposed by Vidal [2]. Some improvements over AESA, have been developed; for example: LAESA [3], iAESA [4], probabilistic iAESA [4], TLAESA [5] and ModTLAESA [6].

All these classifiers were designed to work with quantitative data when the prototype comparison function satisfies the triangle inequality. However, in soft sciences as Medicine, Geology, Sociology, etc., the prototypes are commonly described by mixed data. In these cases, sometimes the comparison function does not satisfy the triangle inequality and therefore, we can not use most of the classifiers proposed for quantitative prototype descriptions. Thus, if a metric is not available but a comparison function that evaluates the similarity between a pair of prototypes could be defined, the objective would be to find the *k* most similar neighbors (*k-MSN*) and use them for classifying. For this reason, in this paper we introduce a fast approximate *k-MSN* classifier, based on a tree structure and a new approximating and eliminating approach for mixed data (Tree AEMD). The aim of our algorithm is not to find the exact *k-MSN* but fast finding an approximation of them, which allows obtaining classification accuracy close to that obtained by the exact ones.

This paper is organized as follows: in Section 2 the comparison function used in this work is described. In Section 3 Tree AEMD is introduced. Finally, we report experimental results in Section 4 and conclusions in Section 5.

2 Comparison Functions for Mixed Data

In this work, the function *D* [10], which does not fulfil the triangle inequality and allow us to compare prototypes described by mixed data, was used. Let us consider a set of prototypes $\{P_1, P_2, \ldots, P_N\}$, each of them described by *d* attributes $\{x_1, x_2, \ldots, x_d\}$. Each feature could be quantitative or qualitative. The function *D* is defined as follows:

$$D(P_1, P_2) = 1 - \frac{|\{x_i \mid C_i(x_i(P_1), x_i(P_2)) = 1\}|}{d} \tag{1}$$

For qualitative data $C_i(x_i(P_1), x_i(P_2))$ is the following:

$$C_i(x_i(P_1), x_i(P_2)) = \begin{cases} 1 & \text{If } x_i(P_1) = x_i(P_2) \text{ and neither } x_i(P_1) \text{ nor } x_i(P_2) \\ & \quad \text{is a missing value} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

For quantitative data $C_i(x_i(P_1), x_i(P_2))$ is the following:

$$C_i(x_i(P_1),x_i(P_2))=\begin{cases}1 & \text{If } |x_i(P_1)-x_i(P_2)|<\sigma_i \text{ and neither } x_i(P_1) \text{ nor } x_i(P_2) \\ & \quad \text{is a missing value} \\ 0 & \text{otherwise}\end{cases} \quad (3)$$

Where, σ_i is the standard deviation of the attribute x_i in T. In this work, the Similarity $S(P_1,P_2)=1-D(P_1,P_2)$ was used for our algorithm.

3 Proposed Classifier

The proposed classifier (Tree AEDM) consists of two phases: preprocessing and classification, which are described in the next subsections.

3.1 Preprocessing Phase

In the preprocessing phase, a tree structure, a similarity matrix, a representative prototype per class and a similarity threshold are computed as follows:

1. Tree structure (*TS*)*:* This is created using the prototypes in T. In this case, the training set is hierarchically decomposed to create the tree structure. For the decomposition, the *C-Means with Similarity Functions* algorithm (*CMSF*) [10], which is an extension of the *C-Means* algorithm to work with mixed data and any prototype comparison function, is used. In the original *C-Means* the mean of the prototypes of a cluster is considered as the centre of it, meanwhile in *CMSF* a representative prototype in the sample is used as the centre of the cluster. See [10] for details.

The node 0 (root of *TS* tree) contains the whole training set. In order to create the following levels of the tree, each node p of the tree is divided in C clusters, in such a way that each cluster represents a descendant node of p. Each node p of the tree contains three features which are: S_p the set of prototypes that belong to p; N_p the number of prototypes in p and Rep_p a representative prototype of p, which is the most similar in average to the prototypes contained in the node.

Each node is divided again and this process is repeated until a stop criterion is fulfilled. In this work, we used a stop criterion proposed in [11]. This criterion takes into account not only the number of prototypes in the node, but also the class distribution of these prototypes. In this case, if certain percentage (*PercThres*) of the prototypes in a cluster belongs to the same class, two nodes are created, for replacing the original one. Using the prototypes that belong to the majority class, a leaf node is created and marked with the majority class. The remaining prototypes are assigned to a second node. In the second node, the size is considered to decide if the node is a leaf (if $N_p \leq NoP$) or if the node will be divided again. In the nodes where a majority class, can not be recognized, only the size of the nodes is considered to create leaves (if $N_p \leq NoP$ then the node is a leaf).

2. Similarity matrix (*SM*). A matrix containing the similarities per attribute between all the representative prototypes of the nodes in the tree is computed. In this case, $SM(Rep_a,Rep_b,x_i)=1$ if, according to certain criterion, we can conclude that the representative prototypes Rep_a and Rep_b ($a,b=1,\ldots,NoNod$; $NoNod$ is the number of nodes in the tree) are similar considering the attribute x_i ($i=1,\ldots,d$) and

$SM(Rep_a,Rep_b,x_i)$=0, otherwise. In this work, the similarity criteria described in Section 2, were used.

The required space to store *SM* is *NoNod* x *NoNod* x *d* but each element is a bit, therefore, *NoNod* x *NoNod* words of *d* bits are needed for storing *SM*.

3. A representative prototype per class (RP_c). Taking advantage of the class information, in order to obtain a first approximation during the classification phase, we propose to use a representative prototype per class (RP_c). Let $Class_c$ be the set of prototypes in *T*, belonging to the class *c*. Then, for each prototype $P_a \in Class_c$, the average similarity between P_a and the remaining prototypes belonging to the same class is computed as follows:

$$AvgSim(P_a) = \frac{\sum_{P_b \in Class_c, b \neq a} S(P_a, P_b)}{|Class_c| - 1} \tag{4}$$

In this way, the representative prototype for the class *c* (RP_c) will be the one most similar on average:

$$RP_c = ArgMax(AvgSim(P_a)), \quad \forall P_a \in Class_c \tag{5}$$

This process is repeated for every c=1,…,*NC*, where *NC* is the number of classes in the training set.

4. Similarity threshold between prototypes (*SimThres*). This value is used as a confidence threshold to make decisions during the classification phase. In order to obtain *SimThres*, for each class *c*, the average value of similarity among the prototypes belonging to the same class is computed as follows:

$$ClassAvgSim(c) = \frac{\sum_{P_a \in Class_c} AvgSim(P_a)}{|Class_c|} \tag{6}$$

Finally, the similarity threshold (*SimThres*) is computed as the average value of *ClassAvgSim* (*c*), for all c=1,..,*NC*.

3.2 Classification Phase of Tree AEDM

Given a new prototype *Q* to classify, *TS*, *SM*, RP_c and *SimThres*, computed during the preprocessing phase, are used to avoid prototype comparison. In this phase, a classification algorithm, using new approximating and eliminating steps (which are not based on the triangle inequality), is proposed as follows:

1. *Initial approximation step*. At the beginning of the algorithm, the prototype *Q* is compared against the representative prototypes (RP_c, c=1,...,*NC*) to obtain a first approximation to the *MSN*:

$$MSN = ArgMin(D(Q, RP_c)),\ c = 1,...,NC \tag{8}$$

2. *Tree traversal algorithm*. In order to update the *k-MSN*, the tree structure is traversed. Two algorithms to traverse the tree are proposed:

- Tree traversal algorithm, using Depth First Search (*DFS*).
- Tree traversal algorithm, using Best First Search (*BFS*).

3. *Classification step*. Finally, the majority class of the *k-MSN* is assigned to Q.

In the next subsections, the tree traversal algorithms are presented.

3.2.1 Tree Traversal Algorithm, Using *DFS*

1. *CurrentNode*=root
2. *Approximating step*: All the direct descendant nodes (n) of *CurrentNode*, neither traversed nor eliminated by the *Eliminating step*, are compared to Q, in order to select a new node (*MSNod*) to continue the tree traversal:

$$MSNod = \text{ArgMin}(D(Q, Rep_n)) \tag{9}$$

The list of the *k-MSN* is updated during the tree traversal, *MSNod* is marked as already traversed and *CurrentNode=MSNod*.

3. If $D(Q, Rep_{MSNod}) \geq SimThres$, go to step 4, in other case, go to step 5.

The elimination step will be applied only if the current Rep_{MSNod} is similar enough to Q, which is determined using *SimThres.*

4. *Eliminating step*: Since Rep_{MSNod} is very similar to Q, if a prototype is not similar to Rep_{MSNod} in at least the same attributes where Rep_{MSNod} is similar to Q, that prototype will not be more similar to Q than Rep_{MSNod}. Therefore, the representative prototype of *MSNod* (Rep_{MSNod}) is used to eliminate nodes of the tree. In this step, *BR* containing the similarity per attribute, between Q and Rep_{MSNod}, is created as follows:

$$BR_i(Q, Rep_{MSNode}) = C_i(x_i(Q), x_i(Rep_{MSNode})),\ i = 1,..,d \tag{10}$$

Thus, $BR_i(Q, Rep_{MSNod})=1$ if Q and Rep_{MSNod} are similar in the attribute x_i and $BR_i(Q, Rep_{MSNod})=0$, in other case. Using *BR*, and *SM* matrix, those nodes n in the tree (not yet traversed or eliminated), having a representative prototype (Rep_n), which is not similar to Rep_{MSNod}, at least in the same attributes where Rep_{MSNod} is similar to Q, are eliminated.

For example, suppose that Rep_1, Rep_2, Q and Rep_{MSNod}, are such that $BR(Q, Rep_{MSNod})=[1,1,0,1,1,1,0,0]$, $SM(Rep_{MSNod}, Rep_1)=[1,1,1,1,1,1,0,1]$ and $SM(Rep_{MSNod}, Rep_2)=[1,0,0,0,0,1,0,1]$. Then, according to this criterion, Rep_1 is not eliminated because it is similar to Rep_{MSNod} in the same attributes, where Rep_{MSNod} is similar to Q (attributes 1, 2, 4, 5 and 6). But Rep_2 is eliminated, because Rep_2 is not similar in the same attributes, where Rep_{MSNod} is similar to Q (Rep_{MSNod} is similar to Q in attribute 2, but Rep_2 is not similar to Rep_{MSNod} in this attribute). The similarity per attribute between Rep_{MSNod} and Rep_1 ($SM(Rep_{MSNod}, Rep_1)$) and between Rep_{MSNod} and Rep_2 ($SM(Rep_{MSNod}, Rep_2)$) are known, since these similarities were computed in the preprocessing phase. The similarity between Rep_{MSNod} and Q, has already been computed and stored in *BR*.

5. If *CurrentNode* is a leaf, then go to step 6; in other case, go to step 2.
6. When a leaf node l is reached:

- If l has been marked with the majority class, given that, most of the times the most similar prototype in the node will be in the majority class, and will be close to the representative prototype, only the representative prototype Rep_l is considered to update the list of *k-MSN*.
- If l has not been marked with the majority class, then, as there are few prototypes in the node ($N_p \leq NoP$), an exhaustive comparison between Q and the prototypes contained in l is done, for updating the list of *k-MSN*.

7. The tree traversal gets back one level of the tree to evaluate the remaining nodes.
8. If there are still neither traversed nor eliminated nodes in the tree, go to step 9. In other case, go to step 10.
9. Assign to *CurrentNode* the most similar node to Q in the corresponding level and go to step 5.
10. End of the algorithm.

3.2.2 Tree Traversal Algorithm, Using *BFS*

1. *CurrentNode*=root, L =Ø.
2. *Approximating step*: All the direct descendant nodes of *CurrentNode*, not yet traversed or eliminated by the *Eliminating step*, are compared to Q and added to the list L, which is sorted in such a way that the most similar node to Q is in the first place ($L(1)$=*MSNod*). This node (*MSNod*) is selected to continue the tree traversal. The node $L(1)$ is eliminated from the list (because, it is already traversed). The *k-MSN* is updated. The node *MSNod* is marked as node already traversed and *CurrentNode=MSNod*.
3. If $D(Q, Rep_{MSNod}) \geq SimThres$, go to step 4, in other case, go to step 5.
4. *Eliminating step*: this step is the same described for the tree traversal algorithm, using *DFS* (step 4, section 3.2.1).
5. If *CurrentNode* is a leaf, then go to step 6; otherwise, go to step 2.
6. When a leaf node l is reached; this step is the same described for the tree traversal algorithm, using *DFS* (step 6, section 3.2.1).
7. If the list L is not empty; then go to step 8. Otherwise, go to step 10.
8. If the first element in the list L has not been eliminated yet, then go to step 9. In other case, eliminate this element from the list and go to step 7.
9. *CurrentNode*=$L(1)$ and go to step 5.
10. End of the algorithm.

4 Experimental Results

In order to evaluate Tree AEMD classifier, it was compared against the exhaustive *k-NN* [1], FN [7], MS [8], ONC [9], AESA [2], LAESA ($|BP|$=20% of the prototypes in the dataset) [3], iAESA [4], Probabilistic iAESA (using 70% as percentage threshold of the data set) [4], TLAESA [5] and ModTLAESA [6] classifiers. The comparison function D, described in Section 2 was used.

For the experiments, 10 datasets from the UCI repository [12] were used (4 mixed datasets: Hepatitis, Zoo, Flag and Echocardiogram. 3 qualitative datasets: Hayes, Soybean-large and Bridges. 3 quantitative: Glass, Iris and Wine).

In order to compare the different classifiers, the accuracy (*Acc*) and the percentage of comparisons between prototypes (*Comp*), were considered. The accuracy was computed as follows:

$$Acc = \frac{NoCorrectObj}{NoTestObj} \tag{11}$$

Where, *NoCorrectObj* is the number of correctly classified prototypes in the test set and *NoTestObj* is the size of the test set. The percentage of comparisons between prototypes was computed as follows:

$$Comp = \frac{NoCompFastClass * 100}{NoTrainingObj} \tag{12}$$

Where, *NoCompFastClass* is the number of comparisons done by the fast classifier, and *NoTrainingObj* is the size of the training set. According to (12), for the exhaustive classifier, the 100% of the comparisons is done.

In all the experiments, k=1 (for *k-MSN*) was used. In [11] different experiments for choosing a value of the parameter C and *PercThres* were done. In our experiments, C=3, *NoP*=20, and *PercThres*=100% were used, since in [11], the fast classifiers reached their best results with these values.

In tables 1 and 2, the corresponding results (*Acc* and *Comp*) obtained with the different classifiers, are shown. From these tables, we can observe that when the comparison function does not satisfy the triangle inequality, FN, ONC, AESA, LAESA, iAESA, TLAESA and ModTLAESA classifiers become approximate methods. However, the percentage of comparisons is, on average, reduced from 100%, done by the exhaustive search, to 58.17% (FN), 33.79% (ONC), 53.47 % (TLAESA), 30.10% (ModTLAESA), 28.79 % (LAESA), 25.82 % (AESA) and 23.88% (iAESA). The classifier proposed in this work (Tree AEMD), did the smallest number of prototype comparisons, using *DFS* (9.39%) and *BFS* (8.81%).

The experiments were repeated, using k=3 and k=5 and the performance of the classifiers was similar.

Table 1. Obtained results using different classifiers

Datasets	Exhaustive *k-NN* classifier		FN		MS		ONC		AESA		LAESA	
	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp
Hep.	81,66	100	82,00	132,9	81,36	75,98	80,71	68,59	80,57	52,96	80,54	61,86
Zoo	96,00	100	95,41	47,53	94,35	35,44	94,18	23,90	96,00	23,50	96,00	15,26
Flag	54,67	100	52,26	48,26	50,21	45,93	50,39	32,98	51,45	28,02	51,45	25,73
Echo.	82,44	100	81,60	103,8	80,15	85,41	80,10	73,40	81,77	62,04	81,77	68,23
Hayes	81,24	100	81,24	42,65	81,27	27,54	81,81	20,29	81,24	24,82	81,24	23,32
Soyb. L	85,40	100	84,26	35,65	84,65	42,98	84,95	23,25	85,40	2,51	85,40	4,49
Bridges	57,85	100	57,84	62,57	55,84	31,58	54,00	31,55	57,85	25,62	57,85	36,10
Glass	68,26	100	68,02	40,26	68,25	23,85	67,29	21,80	66,45	14,02	67,92	20,83
Iris	93,30	100	93,30	25,98	92,54	21,32	92,67	21,37	93,30	9,22	93,30	6,86
Wine	90,90	100	90,10	42,05	90,10	20,45	90,49	20,73	89,01	15,46	90,90	25,26
Avg.	**79,17**	100	**78,60**	58,17	**77,87**	41,05	**77,66**	33,79	**78,30**	25,82	**78,64**	28,79

Table 2. Obtained results using different classifiers

Datasets	i AESA		Probabilistic i AESA		TLAESA		ModTLAESA		PROPOSED CLASSIFIER			
									Tree AEMD (using DFS)		Tree AEMD (using BFS)	
	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp
Hep.	81,03	52,8	80,64	32,44	81,33	87,54	81,66	72,65	80,51	14,6	80,01	13,9
Zoo	96,0	19,4	94,00	17,51	96,00	42,74	96,00	23,95	95,65	9,26	94,22	8,35
Flag	51,36	27,7	49,62	26,41	52,84	48,41	52,09	32,95	52,64	10,2	52,35	9,46
Echo.	81,05	63,6	80,06	63,08	81,77	71,58	82,44	44,62	81,77	13,1	81,40	14,9
Hayes	80,77	17,6	80,07	16,74	80,54	46,42	81,06	24,05	80,95	11,3	80,34	12,4
Soyb. L	85,4	1,96	82,15	2,04	85,40	47,51	85,40	16,85	84,57	2,40	81,51	2,94
Bridges	57,85	25,1	56,95	25,06	56,74	46,75	57,23	38,74	54,00	6,88	54,00	7,93
Glass	66,34	12,6	66,21	12,06	67,92	62,47	67,72	22,85	67,37	8,33	65,54	7,53
Iris	93,3	7,54	93,30	8,01	93,30	41,51	93,30	11,65	93,30	8,62	93,30	6,55
Wine	90,01	10,6	90,90	10,54	90,90	39,75	90,90	12,64	90,90	9,26	90,10	4,09
Avg.	**78,31**	23,9	**77,39**	21,39	**78,67**	53,47	**78,78**	30,10	**78,17**	9,39	**77,28**	8,81

5 Conclusions

In this work, a fast approximate *k-MSN* classifier for mixed data, based on a tree structure and an approximating-eliminating approach, not based on metric properties, was proposed. In order to compare our classifier, FN, MS, ONC, AESA, LAESA, iAESA, probabilistic iAESA, TLAESA, and ModTLAESA classifiers were implemented using the same prototype comparison function for mixed data. Based on our experimental results, it is possible to conclude that, the proposed classifier obtained a similar accuracy, but it needed the smallest number of prototype comparisons.

As future work, we are going to look for other elimination criteria, which could improve the performance of the proposed classifier.

References

1. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. Trans. Information Theory 13, 21–27 (1967)
2. Vidal, P.E.: An algorithm for finding nearest neighbours in (approximately) constant average time complexity. Pattern Recognition Letters 4, 145–157 (1986)
3. Micó, L., Oncina, J., Vidal, E.: A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing-time and memory requirements. Pattern Recognition Letters 15, 9–17 (1994)
4. Figueroa, K., Chávez, E., Navarro, G., Paredes, R.: On the least cost for proximity searching in metric spaces. In: Àlvarez, C., Serna, M. (eds.) WEA 2006. LNCS, vol. 4007, pp. 279–290. Springer, Heidelberg (2006)
5. Mico, L., Oncina, J., Carrasco, R.: A fast Branch and Bound nearest neighbor classifier in metric spaces. Pattern Recognition Letters 17, 731–739 (1996)
6. Tokoro, K., Yamaguchi, K., Masuda, S.: Improvements of TLAESA nearest neighbor search and extension to approximation search. In: ACSC 2006: Proceedings of the 29th Australian Computer Science Conference, pp. 77–83 (2006)
7. Fukunaga, K., Narendra, P.: A branch and bound algorithm for computing k-nearest neighbors. IEEE Trans. Comput. 24, 743–750 (1975)
8. Moreno-Seco, F., Mico, L., Oncina, J.: Approximate Nearest Neighbor Search with the Fukunaga and Narendra Algorithm and its Application to Chromosome Classification. In: Sanfeliu, A., Ruiz-Shulcloper, J. (eds.) CIARP 2003. LNCS, vol. 2905, pp. 322–328. Springer, Heidelberg (2003)
9. Oncina, J., Thollard, F., Gómez-Ballester, E., Micó, L., Moreno-Seco, F.: A Tabular Pruning Rule in Tree-Based Fast Nearest Neighbor Search Algorithms. In: Martí, J., Benedí, J.M., Mendonça, A.M., Serrat, J. (eds.) IbPRIA 2007. LNCS, vol. 4478, pp. 306–313. Springer, Heidelberg (2007)
10. García-Serrano, J.R., Martínez-Trinidad, J.F.: Extension to C-Means Algorithm for the use of Similarity Functions. In: Żytkow, J.M., Rauch, J. (eds.) PKDD 1999. LNCS (LNAI), vol. 1704, pp. 354–359. Springer, Heidelberg (1999)
11. Hernández-Rodríguez, S., Martínez-Trinidad, J., Carrasco-Ochoa, A.: Fast k Most Similar Neighbor Classifier for Mixed Data Based on a Tree Structure. In: Rueda, L., Mery, D., Kittler, J. (eds.) CIARP 2007. LNCS, vol. 4756, pp. 407–416. Springer, Heidelberg (2007)
12. Blake, C., Merz, C.U.: Repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, CA (1998), `http://www.uci.edu/mlearn/databases/`