Transforming Strings to Vector Spaces Using Prototype Selection

Barbara Spillmann[1], Michel Neuhaus[1], Horst Bunke[1],
Elżbieta Pękalska[2], and Robert P.W. Duin[2]

[1] Institute of Computer Science and Applied Mathematics, University of Bern,
Neubrückstrasse 10, CH-3012 Bern, Switzerland
[2] Faculty of Electrical Engineering, Mathematics and Computer Science, Mekelweg 4,
2628 CD Delft, Delft University of Technology, The Netherlands
{spillman, mneuhaus, bunke}@iam.unibe.ch,
e.m.pekalska@tudelft.nl, r.duin@ieee.org

Abstract. A common way of expressing string similarity in structural pattern recognition is the edit distance. It allows one to apply the kNN rule in order to classify a set of strings. However, compared to the wide range of elaborated classifiers known from statistical pattern recognition, this is only a very basic method. In the present paper we propose a method for transforming strings into n-dimensional real vector spaces based on prototype selection. This allows us to subsequently classify the transformed strings with more sophisticated classifiers, such as support vector machine and other kernel based methods. In a number of experiments, we show that the recognition rate can be significantly improved by means of this procedure.

1 Introduction

Strings are one of the fundamental representation formalisms in structural pattern recognition [1]. Using a sequence of symbols rather than a vector of features often has some advantages. For example, the number of symbols in a string is variable and depends on the individual pattern under consideration, while in a feature vector we are forced to always use the same number of features, no matter how simple or complex a pattern is. In fact, strings have been successfully used in a number of applications, including digit recognition [2], shape classification [3,4], and bioinformatics [5].

In many tasks, one needs to measure distances between patterns. In case of string representations the standard distance function is the edit distance. This distance function is based on the minimum number of edit operations, such as insertion, deletion and substitution of symbols, required to transform one of two given strings into the other [6]. This distance can be computed in quadratic time with respect to the lengths of the two strings under consideration. Based on the edit distance one can easily implement classifiers of the nearest-neighbor type. However, more sophisticated classifiers, such as Bayes classifier, neural net, or

D.-Y. Yeung et al. (Eds.): SSPR&SPR 2006, LNCS 4109, pp. 287–296, 2006.

support vector machine, are not applicable in the domain of strings [7,8]. This is a serious drawback and restriction of string based pattern representation.

In the present paper we propose a transformation that maps elements from the domain of strings into real vector spaces. This transformation is intended to maintain the convenience and representational power of strings, but makes available, at the same time, the rich repository of classification tools developed in statistical pattern recognition. A transformation of graphs into vector spaces has been proposed recently [9]. In [10] general properties of embedding transformations have been discussed from various points of view. The method proposed in this paper is closely related to the dissimilarity based approach to pattern recognition proposed in [11,12]. However, while the main focus in [11,12] is on the transformation of feature vectors into dissimilarity spaces, and the possible gain in recognition accuracy obtained from this transformation, the main motivation of our approach is to build a bridge between structural and statistical pattern recognition by making the large spectrum of classifiers known from statistical pattern recognition available to string representations.

In the next section, we will introduce our terminology. Then, in Section 3, we will show how strings are transformed to n-dimensional real vector spaces, $\mathbb{R}^n$, based on various prototype selection procedures. Experimental results of the proposed method, applied to handwritten digit recognition using nearest-neighbor classifiers and support vector machines, are reported in Section 4. Finally, in Section 5, we present concluding remarks.

2 Basic Notation

Let A be a finite alphabet of symbols and A^* be the set of all strings over A. Furthermore, let ϵ denote the empty symbol. We can replace a symbol $a \in A \cup \{\epsilon\}$ by $b \in A \cup \{\epsilon\}$ and call this action an edit operation. More precisely, we refer to $a \to b$ as a substitution, $a \to \epsilon$ a deletion and $\epsilon \to a$ an insertion. In order to measure the dissimilarity of strings, a cost c is assigned to these edit operations: $c(a \to b)$, $c(a \to \epsilon)$ and $c(\epsilon \to a)$. Given a sequence $S = e_1, \ldots, e_n$ of edit operation, its cost is defined as $c(S) = \sum_{i=1}^{n} c(e_i)$. Considering two strings $x, y \in A^*$ and all sequences of edit operations that transform x into y, the edit distance, $d(x, y)$, of x and y is the sequence with minimum cost. The edit distance can be computed by dynamic programming in $O(nm)$ time and space, where n and m are the lengths of the two strings under consideration.

With the notation introduced above, the *set median string* and the *set marginal string* of a given set of strings can be defined as follows. If we denote a set of strings by $\mathcal{X}$, the set median string of $\mathcal{X}$, median $(\mathcal{X})$, is defined as the string $x_{mdn} \in \mathcal{X}$ that satisfies $x_{mdn} = \text{argmin}_{y \in \mathcal{X}} \sum_{x \in \mathcal{X}} d(x, y)$. It is a popular approximation of the generalized median string [13]. Similar to the set median we define the set marginal string, marginal $(\mathcal{X})$, of X as the string $x_{mrg} \in \mathcal{X}$ for which the sum of the edit distances to the remaining elements in $\mathcal{X}$ is maximal: $x_{mrg} = \text{argmax}_{y \in \mathcal{X}} \sum_{x \in \mathcal{X}} d(x, y)$. Obviously, set median and set

marginal strings can be easily obtained by first computing all pairwise distances and then selecting the string with the minimum and maximum average distance, respectively.

3 Transforming Strings to Real Vector Spaces

The idea of our transformation approach is to select a number of prototypes out of a given set of strings. By characterizing an arbitrary string in terms of its edit distances to the prototypes, we obtain a vectorial description of the string. More precisely, a string can be transformed into a vector by calculating the edit distances to all the prototypes, where each distance represents one vector component. Formally, if we denote a set of strings (over an alphabet A) by $\mathcal{X} \subseteq A^*$ and a set of prototypes by $\mathcal{P} = \{p_1, \ldots, p_n\} \subseteq \mathcal{X}$, the transformation $t_n^{\mathcal{P}} : \mathcal{X} \to \mathbb{R}^n$ is defined as a (not necessarily injective) function, where $t_n^{\mathcal{P}}(x) = (d(x, p_1), \ldots, d(x, p_n))$ and $d(x, p_i)$ is the edit distance between the strings x and p_i. Obviously, the dimension of the vector space equals the number of prototypes.

3.1 Prototype Selection Methods

In the previous paragraph, the basic idea of our transformation from the string domain to a vector space has been described. However, no concrete prototype selection strategies have been considered. In the current subsection we will discuss possible algorithms for selecting prototypes from a given set of patterns.

Intuitively, a good selection strategy should satisfy the following three conditions. First, if some prototypes are similar—that is, if they are close in the space of strings—their distances to a sample string should vary only little. Hence, in this case, some of the respective vector components are redundant. Consequently, a selection algorithm should *avoid redundancies*. Secondly, to include as much structural information as possible in the prototypes, they should be *uniformly distributed* over the whole set of patterns. Thirdly, since outliers are likely to introduce noise and distortions, a selection algorithm should *disregard outliers*.

In this paper we will focus on four different class-independent selection algorithms, which we call *center prototype selector*, *border prototype selector*, *spanning prototype selector* and *k-medians prototype selector*. In the following, we will describe these selection algorithms and discuss them in terms of the above mentioned criteria.

Center Prototype Selector. As its name indicates, the *center prototype selector (c-ps)* selects prototypes situated in the center of a given set of strings. Considering the set median string to be the most central string, the set of i prototypes $\mathcal{P}_i \subseteq \mathcal{X}, i = 0, \ldots, |\mathcal{X}|$, selected by the *c-ps*, is iteratively constructed as:

$$\mathcal{P}_i = \begin{cases} \emptyset & \text{if } i = 0, \\ \mathcal{P}_{i-1} \cup \{p_i\} & \text{if } 0 < i \leq |\mathcal{X}|, \qquad \text{where } p_i = \text{median}\,(\mathcal{X} \setminus \mathcal{P}_{i-1}). \end{cases}$$

For an intuitive illustration using points on the two-dimensional plane see Fig. 1a. Due to their central position all prototypes are structurally similar. Hence, many redundant prototypes occur. On the other hand, strings at the border are not considered, and thus, the set of prototypes is not negatively influenced by outliers. Obviously, the property of uniform distribution is not satisfied.

Border Prototype Selector. The *border prototype selector (b-ps)* acts just contrary to the *c-ps*. It selects prototypes from the border and is therefore based on marginal strings. The set of i prototypes $\mathcal{P}_i \subseteq \mathcal{X}, i = 0, \ldots, |\mathcal{X}|$, selected by the *border prototype selector* is defined as:

$$\mathcal{P}_i = \begin{cases} \emptyset & \text{if } i = 0, \\ \mathcal{P}_{i-1} \cup \{p_i\} & \text{if } 0 < i \leq |\mathcal{X}|, \end{cases} \quad \text{where } p_i = \text{marginal}\,(\mathcal{X} \setminus \mathcal{P}_{i-1}).$$

An illustration is given in Fig. 1b. Obviously, only few redundant prototypes are selected. However, there are no prototypes located in the center and the condition of uniform distribution is only partially fulfilled. Furthermore, it is to be expected that there are outliers among the prototypes selected by the *b-ps*.

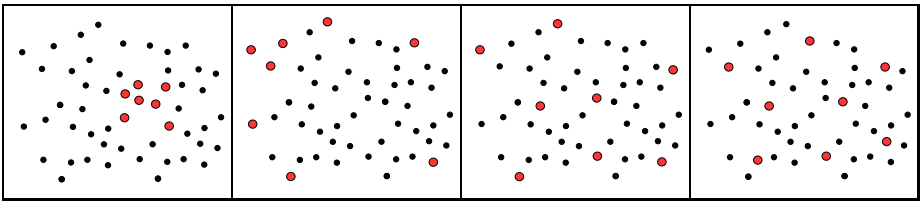
Spanning Prototype Selector. A set of prototypes selected by the *spanning prototype selector (s-ps)* is given by the following iterative procedure. The first prototype is the set median string. Every further prototype is the string with the largest distance to the set of previously selected prototypes. Analog algorithms have been proposed for k-means initialization [14,15]. Formally, the set of i prototypes $\mathcal{P}_i \subseteq \mathcal{X}, i = 0, \ldots, |\mathcal{X}|$, selected by the *spanning prototype selector (s-ps)* is defined as

$$\mathcal{P}_i = \begin{cases} \emptyset & \text{if } i = 0, \\ \text{median}\,(\mathcal{X}) & \text{if } i = 1, \\ \mathcal{P}_{i-1} \cup \{p_i\} & \text{if } 1 < i \leq |\mathcal{X}|, \end{cases} \quad \text{where } p_i = \underset{x \in \mathcal{X} \setminus \mathcal{P}_{i-1}}{\text{argmax}} \min_{p \in \mathcal{P}_{i-1}} d(x, p).$$

Each additional prototype selected by the *s-ps* is the string located the furthest away from the already selected prototypes. Thus, the case that two prototypes are very close is avoided and hence also redundant prototypes are prevented. It is in the nature of this algorithm that new prototypes are selected from an area which hasn't been considered before. This leads to a good distribution of the prototypes. However, since outliers have a large distance to the other patterns, there is a certain chance for them to be selected. Fig. 1c illustrates the behavior of the *s-ps*.

K-Medians Prototype Selector. The *k-medians prototype selector (km-ps)* is based on the k-means clustering algorithm [16]. The idea is to find n clusters in the given set of data and to declare each cluster center, i.e. the set median of each cluster, to be a prototype. An illustration can be found in Fig. 1d.

The advantage of the prototypes selected by the *km-ps* is that they are evenly spread over the whole set of data. Similar strings are represented by the same



a) b) c) d)

Fig. 1. Illustration of the a) *c-ps*, b) *b-ps*, c) *s-ps* and d) *km-ps* algorithms. The larger dots represent the selected prototypes.

prototype. Hence, redundant prototypes are mostly avoided. Furthermore, outliers usually aren't positioned in the center of a k-medians cluster and thus the chance for them to be selected as a prototype is small.

4 Experimental Results

This section provides experimental classification results using the transformation schema introduced in Section 3. The prototype selection algorithms are tested on the *Pendigits* database described in [17] (original, unnormalized version). The original version contains 10,992 instances of handwritten digits 0 to 9, where 7,494 are used for training and 3,498 for testing (see Fig. 2). Each digit is originally given as a sequence of two-dimensional points. To obtain a suitable string representation, each digit curve is first approximated by a sequence $s = z_1, \ldots, z_n$ of vectors of constant length $|z_i| = l$, such that the start and end points of all z_i lie on the original curve.

A string can be generated by one of the following two methods. Either the sequence s of vectors is directly regarded as a string. Then the costs of the edit operations are defined as follows. A substitution has the costs $c(z_i \rightarrow z_j) = \|z_i - z_j\|^{q_v}$, where q_v is a positive real value; for the costs of insertion and deletion we take the arithmetic mean of the extremal values (0 and $(2l)^{q_v}$) of the substitution costs, which is $2^{q_v-1}l^{q_v}$. This cost function is referred to as *vector cost function*. Another way of generating a string is to consider the sequence $\alpha_1, \ldots, \alpha_{n-1}$ of angles, where α_i is the angle between vectors z_i and z_{i+1}. In that case, the costs assigned to the edit operations are constantly set to $0 \leq q_a \leq \frac{\pi}{2}$ in case of angle insertions and deletions, and for substitutions the costs are given by the absolute difference of the two involved angles α_i and α_j, $c(\alpha_i \rightarrow \alpha_j) = |\alpha_i - \alpha_j|$. We call this cost function *angle cost function*.

To find a suitable string representation, the values of parameters l, q_v and q_a are optimized on a validation set, which consists of one fifth of the original training set. For this purpose we generate string representations for various combinations of parameter values and classify the validation set with a k-nearest-neighbor classifier, using the original training set minus the validation set as the set of labeled training items. Finally, we select the parameter combination of

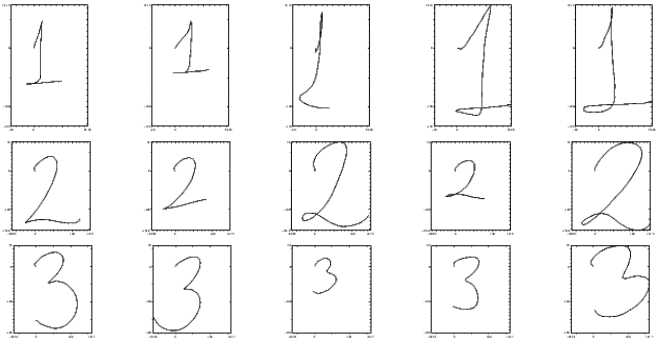


Fig. 2. Example patterns of the classes "1", "2" and "3"

l, q_v, q_a and k, that leads to the highest recognition rate. The value of parameter k is used at the same time to build a k-nearest-classifier in the string domain, which we use as a reference classifier for vector space classifiers.

Once the dataset is prepared, i.e. all the elements are represented as strings, the prototypes are selected from the training set that is made up of the remaining four fifths of the original training set (i.e. the part of the training set that is not used for validation). The prototypes are exclusively used for the purpose of mapping the data from the string to the vector space. Once the prototypes are selected, the complete dataset is mapped into the vector domain, without losing the partitioning into training, validation and test set. That is, each set still represents the same objects as in the string domain. After the dataset has been mapped to the vector domain, any classifier known from statistical pattern recognition can be trained by using the transformed validation and training sets, as described in the following.

The number of prototypes, i.e. the dimensionality of the vector space, and the prototype selection strategy as well as classifier parameters are determined on the validation set. That is, a number of possible vector space dimensions are considered for each selection strategy and one individual classifier is built for each combination of possible dimensionality and selection strategy. Then the validation set is classified with each of these classifiers. Finally, the parameter values for the dimensionality, the selection strategy, and the classifier leading to best performance on the validation set are selected. Then this classifier is taken to classify the test set. An overview of the classifiers we used in our experiments is given in the following.

First of all, we apply a kNN classifier not only in the string domain, but also in the vector space. The distance measure we use is the Minkowski metric $L_p(\boldsymbol{x}, \boldsymbol{y}) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$, where $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and $\boldsymbol{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$. In case of this classifier, both parameters k and p are optimized on the validation set and the training set (excluding the validation set) is used for finding the nearest neighbors.

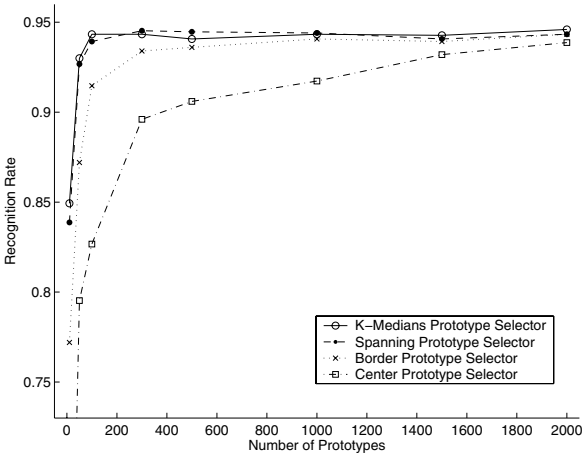


Fig. 3. Comparison of the four prototype selection strategies *c-ps*, *b-ps*, *s-ps* and *km-ps*: recognition rates of a 3NN classifier on the validation set depending on the number of prototypes

Another possibility is to apply the kNN classifier in a higher-dimensional feature space. This method uses kernel theory [18] which has become a popular subject in statistical pattern recognition. Instead of directly classifying the transformed strings, the patterns are mapped by a non-linear function $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^m (m > n)$ to a higher-dimensional real vector space $\mathbb{R}^m$, called feature space, in which the kNN classification is performed with the Euclidean distance L_2 as distance measure. In the feature space $\mathbb{R}^m$ an inner product $< \boldsymbol{x}, \boldsymbol{y} >$ exists and $\mathbb{R}^m$ is complete with respect to the norm $\|\boldsymbol{x}\| = \sqrt{< \boldsymbol{x}, \boldsymbol{x} >}$, defined by the inner product. This fact allows us to define kernel functions $k_\Phi(\boldsymbol{x}, \boldsymbol{y}) := \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle , (\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n)$. The kernel function $k_\Phi(\boldsymbol{x}, \boldsymbol{y})$ can then be regarded as a similarity measure in the vector space $\mathbb{R}^n$, and the Euclidean distance in the feature space $\mathbb{R}^m$ can be derived from it. With the use of this method, the explicit application of the mapping Φ can be avoided. In our experiments we used the following standard kernel functions: radial basis function, polynomial function, and sigmoid function [18].

Another classification method using kernel functions is the support vector machine (SVM) [8,19]. The key idea is to find a hyperplane that separates the data into two classes with a maximal margin. Such a hyperplane can only be found if the data are linearly separable. If linear separability is not fulfilled, a weaker definition of the margin, the soft margin, can be used. In this case, the optimal hyperplane is the one that minimizes the sum of errors and maximizes the margin. This optimization problem is usually solved by quadratic programming. In order to improve the classification of non-linearly separable data, an explicit mapping to a higher-dimensional feature space can be performed, or instead, a

the above mentioned kernel function can be applied. For our experiments we use the LIBSVM library [20]. The kernel functions used in our experiments are the linear kernel and the radial basis function.

Fig. 3 illustrates the performance of the four prototype selection methods described in Section 3 with respect to the number of prototypes. It shows the recognition rate of a 3NN classifier on the validation set, where the digit curves are approximated by segments of length 20 and the angle cost function with $q_a = \frac{11}{36}\pi$ is used. (Note, the classifier parameter k is kept constant for this plot.) The number of prototypes n ranges from 10 to 2000. Generally, it can be observed that the recognition rate increases with an increasing number of prototypes. Once the recognition rate has reached a certain value, however, it roughly remains constant. For a small value of n, differences in the quality of each method can be detected, but the recognition rates become incrementally equal for larger n. That is, while the *c-ps* and the *b-ps* clearly perform worse than the *s-ps* and *km-ps* for small n, the difference at $n = 2000$ almost disappears. We observe that selection strategies which uniformly distribute the prototypes, *s-ps* and *km-ps*, have a higher performance for smaller n.

In Tab. 1 the recognition rates on the test set with the above mentioned classifiers are listed. The table shows the results for both angle (`pen ang`) and vector cost function (`pen vec`). Three different partitions of the dataset into a validation, training and test set have been used. The term `pen1` refers to the original partitioning into training and test set. The experiments `pen2` and `pen3` are further setups, where the size of each set is unchanged, but different partitions have been performed. In order to show the performance of the transformation, we use the recognition rate of the kNN classifier in the string space as a reference value. Recognition rates printed in bold face refer to statistically significant better results at a significance level of 0.95.

Table 1. Recognition rates on the Pendigits dataset

	*k*NN string domain	*k*NN Mink. metric	*k*NN RBF kernel	*k*NN poly. kernel	*k*NN sig. kernel	SVM RBF	SVM lin.
`pen1 ang`	88.56	**90.99**	**89.51**	**89.48**	**89.74**	**90.99**	**94.54**
`pen2 ang`	92.48	92.96	91.97	92.00	92.32	**96.16**	**95.25**
`pen3 ang`	92.71	**93.43**	92.36	92.36	91.83	**95.83**	**95.70**
`pen1 vec`	97.48	97.60	97.06	97.06	97.08	**98.34**	97.88
`pen2 vec`	99.33	99.31	99.28	99.28	99.20	**99.68**	**99.57**
`pen3 vec`	99.33	99.25	99.12	99.12	99.04	99.55	99.31

First we observe that the application of the standard kNN classifier in the vector space (column kNN Mink. metric) leads to an improvement of the recognition rate over the kNN classifier in the string domain in four out of six cases. For all other kernel based kNN classifiers (columns kNN RBF kernel, kNN poly. kernel and kNN sig. kernel), an improvement is obtained in only one out of six cases. However, both SVMs demonstrate superior performance. The SVM with radial basis function kernel leads in all six cases to an improvement, five of which are statistically significant, and even the linear kernel SVM shows a higher recognition performance than the classifier in the string domain in all cases but one.

In [21], classification based on two MLP approaches has been performed on the same data. The recognition rates on the test set achieved in [21] are 95.26% and 94.25%, respectively. We note that both recognition rates are already outperformed by our kNN classifier in the string domain using the vector cost function. Nevertheless, a further improvement can be achieved by means of the proposed embedding procedure in conjunction with both SVMs.

5 Conclusion

In this paper we study the representation and classification of strings in n-dimensional real vector spaces. The transformation is accomplished with a prototype selection procedure, where each vector component of a transformed string represents the edit distance to one prototype.

We evaluate the transformation on strings extracted from the Pendigits database. The recognition rates of several kNN methods and support vector machines for the transformed strings are compared to a kNN classifier in the original string domain. We show that by means of SVM the recognition rate for strings can significantly be improved. However, the improvement of the correct classification rate in the considered task is just one contribution of this paper. From the general point of view, the methodology proposed in the paper opens new ways of embedding symbolic data structures, i.e. strings, into vector spaces using edit distance and prototype selection. Based on such an embedding, a large number of methods from statistical pattern recognition become available to string representations. In our future work we will study additional classifiers, such as Bayes classifier and neural net, as well as data dimensionality reduction and clustering tasks.

Acknowledgements

This work has been partially supported by the Swiss National Science Foundation NCCR program *Interactive Multimodal Information Management (IM)2* in the Individual Project *Multimedia Information Access and Content Protection* as well as by the Dutch Organization for Scientific Research (NWO).

References

1. Bunke, H., Sanfeliu, A.: Syntactic and Structural Pattern Recognition – Theory and Applications. World Scientific Publ. Co. (1990)
2. Cha, S.H., Shin, Y.C., Srihari, S.N.: Approximate stroke sequence matching algorithm for character recognition and analysis. In: 5th International Conference on Document Analysis and Recognition. (1999) 53–56
3. Bunke, H., Bühler, U.: Applications of approximate string matching to 2D shape recognition. Pattern Recognition **26** (1993) 1797–1812
4. Chen, S.W., Tung, S.T., Fang, C.Y., Cheng, S., Jain, A.K.: Extended attributed string matching for shape recognition. Computer Vision and Image Understanding **70** (1998) 36–50
5. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: Biological sequence analysis. Cambridge University Press, Cambridge, UK (1998)
6. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the ACM **21** (1974) 168–173
7. Duda, R., Hart, P., Stork, D.: Pattern Classification. 2nd edn. Wiley, New York (2001)
8. Vapnik, V.: The Nature of Statistical Learning Theory. 2nd edn. Springer-Verlag (2000) ISBN: 0-387-98780-0.
9. Wilson, R.C., Hancock, E.R., Luo, B.: Pattern vectors from algebraic graph theory. IEEE Transactions on Pattern Analysis and Machine Intelligence **27** (2005) 1112–1124
10. Hjaltason, G.R., Samet, H.: Properties of embedding methods for similarity searching in metric spaces. IEEE Transactions on Pattern Analysis and Machine Intelligence **25** (2003) 530–549
11. Pękalska, E.: Dissimilarity representations in pattern recognition. PhD thesis, Delft University of Technology (2005)
12. Pękalska, E., Duin, R.P., Paclík, P.: Prototype selection for dissimilarity-based classifiers. Pattern Recognition **39** (2006) 189–208
13. Kohonen, T.: Median strings. Pattern Recognition Letters **3** (1985) 309–313
14. Katsavounidis, I., Kuo, C.C.J., Zhang, Z.: A new initialization technique for generalized lloyd iteration. IEEE Signal processing letters **1** (1994) 144–146
15. Juan, A., Vidal, E.: Comparison of four initialization techniques for the k -medians clustering algorithm. In: Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition, London, UK, Springer-Verlag (2000) 842–852
16. Jain, A.K., Dubes, R.C.: Algorithms for clustering data. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1988)
17. Alpaydin, E., Alimoglu, F.: Department of Computer Engineering, Bogaziçi University, 80815 Istanbul Turkey (1998) `ftp://ftp.ics.uci.edu/pub/mlearn/databases/pendigits`.
18. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)
19. Vapnik, V.: Statistical Learning Theory. Wiley-Interscience (1998) ISBN: 0-471-03003-1.
20. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. (2001) `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.
21. Alimoglu, F., Alpaydin, E.: Combining multiple representations for pen-based handwritten digit recognition. Turk J Elec Engin **9** (2001)