Available online at www.sciencedirect.com

ScienceDirect

Pattern Recognition 40 (2007) 3721–3727

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/pr

A SVM-based cursive character recognizer

Francesco Camastra*

Department of Applied Science, University of Naples Parthenope, Via A. De Gasperi 5, 80133 Napoli, Italy

Received 1 September 2005; received in revised form 20 February 2007; accepted 21 March 2007

Abstract

This paper presents a cursive character recognizer, a crucial module in any cursive word recognition system based on a segmentation and recognition approach. The character classification is achieved by using *support vector machines* (SVMs) and a *neural gas*. The neural gas is used to verify whether lower and upper case version of a certain letter can be joined in a single class or not. Once this is done for every letter, the character recognition is performed by SVMs. A database of 57 293 characters was used to train and test the cursive character recognizer. SVMs compare notably better, in terms of recognition rates, with popular neural classifiers, such as learning vector quantization and multi-layer-perceptron. SVM recognition rate is among the highest presented in the literature for cursive character recognition.
© 2007 Pattern Recognition Society. Published by Elsevier Ltd. All rights reserved.

Keywords: Support vector machines; Neural gas; Learning vector quantization; Multi-layer-perceptron; Crossvalidation; Cursive character recognition

1. Introduction

Off-line cursive word recognition has many applications such as the reading of postal addresses and the automatic processing of forms, checks and faxes [1,2]. The main approaches [3,4] for off-line cursive word recognition can be divided into segmentation-based and holistic one. The former is based on the word segmentation into letters [5,6] and the recognition of individual letters; the latter tries to recognize the word image as a whole [2].

In the segmentation-based strategy for cursive word recognition, no method is available to achieve a perfect segmentation. Hence the word is first oversegmented, i.e. fragmented into primitives that are characters or parts of them, to ensure that all appropriate letter boundaries have been dissected. To find the optimal segmentation, a set of segmentation hypotheses is tested by merging neighboring primitives and invoking a classifier to score the combination. Finally, the word with the optimal score is generally found by applying dynamic programming techniques [7].

A crucial module in the segmentation-based approach is a cursive character recognizer for scoring individual characters. It has to cope with the high variability of the cursive letters and their intrinsic ambiguity (letters like *e* and *l* or *u* and *n* can have the same shape).

In this paper, we present a cursive character recognizer where the character classification is achieved by using *support vector machines* (SVMs) and a *neural gas* (NG). The NG is used to verify when the upper and lower case versions of a letter can form a common class. This happens when the two characters (e.g. *o* and *O*) are similar in shape and their vectors in the feature space occupy neighboring or even overlapping regions. By grouping the characters in this way, the number of classes is reduced and a more suitable representation of the data is obtained. The classifier, based on SVMs, provides for the character the class attribution. To our best knowledge, the use of SVMs in the cursive character recognition represents a novelty.

The paper is organized as follows: in Section 2 the method for extracting features for character representation is presented; a review of SVM and NG is provided in Sections 3 and 4, respectively; in Section 5 reports some experimental results; in Section 6 some conclusions are drawn.

* Tel.: +39 338 4447991.
E-mail address: francesco.camastra@uniparthenope.it.

0031-3203/$30.00 © 2007 Pattern Recognition Society. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2007.03.014

2. Feature extraction

Most character recognizers do not work on the raw image, but on a suitable compact representation of the image by means of a vector of features. Since cursive characters present high variability in shapes, a feature extractor should have negligible sensitivity to local shifts and distortions. Therefore feature extractors that perform local averaging are more appropriate than others that yield an exact reconstruction of the pattern (e.g. Zernike polynomials, moments) as shown in Ref. [8]. The feature extractor, fed with the binary image of an isolated cursive character, generates local and global features. The local features are extracted from subimages (*cells*) arranged in a regular grid covering the whole image, as shown in Fig. 1. A fixed set of operators is applied to each cell. The first operator is a counter that computes the percentage of foreground pixels in the cell (*gray feature*) with respect to the total number of foreground pixels in the character image. If n_i is the number of foreground pixels in cell i and M is the total number of foreground pixels in the pattern, then the gray feature related to cell i is n_i/M. The other operators try to estimate to which extent the black pixels in the cell are aligned along some directions. For each direction of interest, a set of N, equally spaced, straight lines are defined, that span the whole cell and that are parallel to the chosen direction. Along each line $j \in [1, N]$ the number n_j of black pixels is computed and the sum $\sum_{j=1}^{N} n_j^2$ is then obtained for each direction. The difference between the sums related to orthogonal directions is used as feature. In our case, the directions of interest were 0^{o} and 90^{o} and the computation of the directional feature becomes easier. If we indicate with h and w, respectively, the height and the width in pixels of the cell, the directional feature d is given by

$$d = \frac{1}{2}\left(1 + \frac{1}{hw^2}\sum_{j=1}^{h} n_j^2 - \frac{1}{h^2 w}\sum_{i=1}^{w} n_i^2\right), \qquad (1)$$

where n_j and n_i indicate, respectively, the number of the black pixels along the jth row and the ith column.

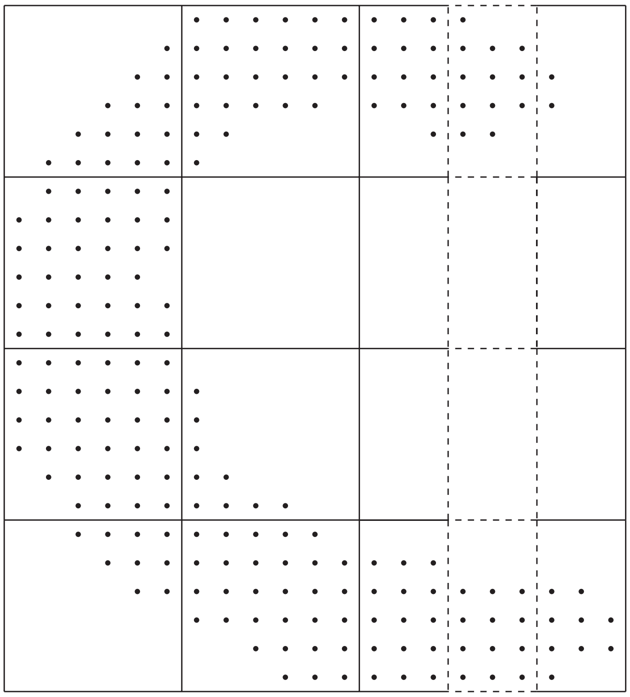
Fig. 1. The image of the character is divided in cells of equal size, arranged in a 4 × 4 grid. The dashed lines indicate the parts of the cells which are overlapped.

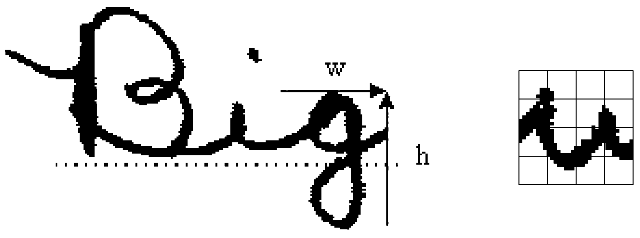

Fig. 2. Global features. The dashed line is the *baseline*, the fraction of h below is used as first global feature. The second global feature is the ratio w/h.

We enriched the local feature set with two global features giving information about the overall shape of the cursive character and about its position with respect to the *baseline* of the cursive word. As shown in Fig. 2, the baseline is the line on which a writer implicitly aligns the word in the absence of rulers. The first global feature measures the fraction of the character below the baseline and detects eventual descenders. The second feature is the *width/height* ratio.

The number of local features can be arbitrarily determined by changing the number of cells or directions examined in each cell. Since classifier reliability can be hard when the number of features is high (*curse of dimensionality*, [9]), we use simple techniques for feature selection in order to keep the feature number as low as possible. Directional features corresponding to different directions were applied and the one having the maximal variance was retained. Therefore the feature set was tested changing the number of cells and the grid giving the best results (4 × 4) was selected.

In the reported experiments we used a feature vector of 34 elements. Two features are global (*baseline* and *width/height ratio*) while the remaining 32 are generated from 16 cells, placed on a regular 4×4 grid; from each cell, the gray feature and one directional feature are extracted. An implementation, in C language, of the feature extraction process is available on request.

3. SVM for classification

Firstly we recall the definition of *Mercer kernel* [10].

Definition 1. Let X be a nonempty set. A function $G : X \times X \to \mathbb{R}$ is called a *Mercer kernel* (or *positive definite kernel*) if and only if is *symmetric* (i.e. $G(x, y) = G(y, x)\ \forall x, y \in X$) and $\sum_{j=1}^{n}\sum_{k=1}^{n} c_j c_k G(x_j, x_k) \geqslant 0$ for all $n \geqslant 2$, $x_1, \ldots, x_n \subseteq X$ and $c_1, \ldots, c_n \subseteq \mathbb{R}$. Each Mercer kernel $G(\cdot)$ can be represented as: $G(x, y) = \langle \Phi(x), \Phi(y)\rangle$ where $\langle \cdot, \cdot \rangle$ is the inner product and $\Phi : X \to \mathscr{F}$, $\mathscr{F}$ is called *feature space*.

An example of Mercer kernel is the *Gaussian* $G(\vec{x},\vec{y}) = \exp(-\frac{\|\vec{x}-\vec{y}\|^2}{\sigma^2})$ where $\sigma \in \mathbb{R}$, $\vec{x},\vec{y} \in \mathbb{R}^n$.

Support vector machine (*SVM*) [11,12] is an algorithm that, after projecting data in the Feature space, computes an *optimal separating hyperplane*.

3.1. Optimal separating hyperplane

In what follows, we assume we are given a set $\mathscr{D}$ of patterns $\vec{x}_i \in \mathbb{R}^n$ with $i = 1,\ldots,\ell$. Each pattern $\vec{x}_i$ belongs to either of two classes and thus is given a label $y_i \in \{-1,+1\}$. The patterns with output $+1$ are called *positive patterns*, while the others are called *negative patterns*. The goal is to establish the equation $\vec{w}\cdot\vec{x}+b$ ($\vec{w},\vec{x} \in \mathbb{R}^n$, $b \in \mathbb{R}$) of the *optimal hyperplane* that divides $\mathscr{D}$ leaving all points of the same class on the same side while maximizing the distance between the two classes and the hyperplane. This can be expressed by the constraints: $y_i(\langle\vec{w}\cdot\vec{x}_i\rangle + b) \geqslant 1$, $i = 1,\ldots,\ell$.

In most practical problems, a separating hyperplane may not exist. Due to presence of noise, an overlapping between classes may exist. Hence it has to allow that the constraints can be violated by some examples [11,12]. Using *slack variables* $\xi_i \geqslant 0$ the constraints can be relaxed in:

$$y_i \cdot (\langle\vec{w}\cdot\vec{x}_i\rangle + b) \geqslant 1 - \xi_i, \quad i = 1,\ldots,\ell. \tag{2}$$

Therefore we can construct a classifier, *SVM* solving the following optimization problem:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{\ell}\xi_i \\ &\text{subject} \quad y_i\cdot(\langle\vec{w}\cdot\vec{x}_i\rangle + b) \geqslant 1-\xi_i, \quad i=1,\ldots,\ell, \\ &\qquad\qquad \xi_i \geqslant 0, \quad i = 1,\ldots,\ell. \end{aligned}$$

The classifier controls at the same time the margin ($\|w\|$) and the number of training errors. The regularization constant $C \geqslant 0$ determines the trade-off between the two terms. The conditional optimization problem can be solved by introducing Lagrange multipliers $\alpha_i \geqslant 0$ and a Lagrangian function $\mathscr{L}$

$$\begin{aligned} \mathscr{L}(\vec{w},b,\alpha) = {} & \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{\ell}\xi_i \\ & - \sum_{i=1}^{\ell}\alpha_i(y_i(\langle\vec{x}_i\cdot\vec{w}\rangle + b) - 1 + \xi_i). \end{aligned} \tag{3}$$

The Lagrangian $\mathscr{L}$ has to be minimized with respect to the *primal variables* $\vec{w}$ and b and maximized with respect to the *dual variables* α_i, i.e. a saddle point has to be found. The condition at the saddle point implies that the derivatives of $\mathscr{L}$ with respect to the primal variables must vanish, that is $\frac{\partial\mathscr{L}(\vec{w},b,\alpha)}{\partial b} = \frac{\partial\mathscr{L}(\vec{w},b,\alpha)}{\partial\vec{w}} = 0$. These conditions lead to

$$\sum_{i=1}^{\ell}\alpha_i y_i = 0, \quad \vec{w} = \sum_{i=1}^{\ell}\alpha_i y_i\vec{x}_i. \tag{4}$$

Hence the solution vector $\vec{w}$ is an expansion in terms of patterns of the training set whose $\alpha_i \neq 0$. These patterns are called *support vectors* (*SV*). Kuhn–Tucker theorem implies that α_i must satisfy the *Karush–Kuhn–Tucker* conditions:

$$\alpha_i \cdot [y_i\langle\vec{x}_i\cdot\vec{w}_i\rangle + b - 1] = 0, \quad i = 1,\ldots,\ell. \tag{5}$$

This condition implies that the SV lie on the margin. All remaining samples of the training set are irrelevant for the optimization since their α_i is null. Therefore Eq. (4) can be written as

$$\vec{w} = \sum_{\alpha_i \in SV}^{\ell}\alpha_i y_i\vec{x}_i. \tag{6}$$

Plugging Eq. (4) into $\mathscr{L}$, one eliminates the primal variables and the optimization problem becomes:

$$\begin{aligned} &\text{maximize} \quad \sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\alpha_i\alpha_j y_i y_j\langle\vec{x}_i\cdot\vec{x}_j\rangle \\ &\text{subject to} \quad 0 \leqslant \alpha_i \leqslant C, \quad i = 1,\ldots,\ell, \\ &\qquad\qquad \sum_{i=1}^{\ell}\alpha_i y_i = 0. \end{aligned}$$

Therefore the hyperplane function can be written as

$$f(x) = \sum_{i=1}^{\ell}\alpha_i y_i\langle\vec{x}_i\cdot\vec{x}_j\rangle + b. \tag{7}$$

3.2. SVM construction

To construct SVMs, an optimal hyperplane in some feature space has to be computed. Hence it is adequate to substitute each training example $\vec{x}_i$ with its corresponding image in the feature space $\Phi(\vec{x}_i)$. The weight vector (4) becomes an expansion of vectors in the feature space

$$\vec{w} = \sum_{i=1}^{\ell}\alpha_i y_i\Phi(\vec{x}_i). \tag{8}$$

Therefore the weight vector is not directly computable when the mapping Φ is unknown. Since $\Phi(\vec{x}_i)$ occur only in scalar products, scalar products can be substituted by an appropriate Mercer kernel $G(\cdot)$, leading to a generalization of the hyperplane function (7)

$$\begin{aligned} f(x) &= \sum_{i=1}^{\ell}\alpha_i y_i\langle\Phi(\vec{x}_i)\cdot\Phi(\vec{x}_j)\rangle + b \\ &= \sum_{i=1}^{\ell}\alpha_i y_i G(\vec{x}_i,\vec{x}_j) + b \end{aligned} \tag{9}$$

and the following quadratic problem to optimize:

$$\begin{aligned}&\text{maximize} && \sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{i=1}^{\ell}\alpha_i\alpha_j y_i y_j G(\vec{x}_i,\vec{x}_j)\\ &\text{subject to} && \alpha_i \geqslant 0, \quad i=1,\ldots,\ell,\\ &&& \sum_{i=1}^{\ell}\alpha_i y_i = 0.\end{aligned}$$

The threshold b can be computed by exploiting the fact that for all SVs $\vec{x}_i$ with $\alpha_i < C$, the slack variable ξ_i is zero, therefore

$$\sum_{j=1}^{\ell} y_j\alpha_j G(\vec{x}_i,\vec{x}_j) + b = y_i. \tag{10}$$

3.3. Multiclass SVMs

In order to use SVM when the number of classes K is larger than 2, a few different strategies have been suggested [13]. In our experiments we have adopted *one-versus-rest* (*o-v-r*) method [14,15]. The method learns one classifier for each of the K classes against all the other classes. More formally, the method consists in training K SVM classifiers f_j by labelling all training points having $y_i = j$ with $+1$ and $y_i \neq j$ with -1 during the training of the jth classifier. In the test stage, the final decision function $F(\cdot)$ is given by

$$F(\vec{x}) = \arg\max_j f_j(\vec{x}). \tag{11}$$

4. Neural gas

NG is a unsupervised version of vector quantization. In NG model no topology of a fixed dimensionality is imposed on the network. NG consists of a set of M units: $A = (c_1, c_2, \ldots, c_m)$. Each unit c_i has an associated *reference vector* $\vec{w}_{c_i}$ ($\vec{w}_{c_i} \in \mathbb{R}^n$) indicating its position or *receptive field center* in input space. The learning algorithm of NG is the following:

(1) Initialize the set A to contain units c_i, with $\vec{w}_{c_i} \in \mathbb{R}^n$, chosen randomly according to input distribution $p(\vec{\zeta})$. Besides, initialize the time parameter t, to 0.
(2) Generate at random an input $\vec{\zeta}$ according to $p(\vec{\zeta})$.
(3) Order all elements of A according to the distance of their reference vectors to $\vec{\zeta}$ e.g. find the sequence of indices $S = (i_0, i_1, \ldots, i_{M-1})$ such that $\vec{w}_{i_0}$ is the reference vector closest to $\vec{\zeta}$, $\vec{w}_{i_1}$ is the second vector closest to $\vec{\zeta}$, etc. Let $k_i(\vec{\zeta}, A)$ the rank associated with $\vec{w}_i$.[1]
(4) Adapt the reference vectors according to

$$\Delta\vec{w}_i = \varepsilon(t)h_\lambda(k_i(\vec{\zeta}, A))(\vec{\zeta} - \vec{w}_i),$$

where

$$h_\lambda(k_i(\vec{\zeta}, A)) = \exp\left(-\frac{k_i}{\lambda(t)}\right),$$

$$\lambda(t) = \lambda_i\left(\frac{\lambda_f}{\lambda_i}\right)^{t/t_f},$$

$$\varepsilon(t) = \varepsilon_i\left(\frac{\varepsilon_f}{\varepsilon_i}\right)^{t/t_f}.$$

(5) Increase the time parameter t : $t = t + 1$.
(6) If $t < t_f$ continue with step 2.

For the time dependent parameters suitable initial values λ_i, ε_i and final values λ_f, ε_f have to be chosen. For the above-mentioned parameter in our work, we adopted the values suggested in Refs. [16–18].

5. Experiments and results

The combined use of NG and SVM is shown to improve the performance of a cursive character classifier.

The letters are present in the database in both upper and lower case version. In some cases, the two versions are different and must be considered as separate classes. In some other cases, the two versions are similar and can be joined in a single class. NG is used to measure the overlapping in the feature space of the vectors corresponding to the two versions of each character. When the overlapping is high enough, upper and lower case versions of the letter are joined in a single class. This improves the performance of the SVM over such classes and results in a better accuracy of the overall character classifier.

Section 5.1 describes the database used in the experiments, Sections 5.2 and 5.3 show how the optimal class representation was found and the recognition experiments, respectively.

5.1. The character database

The cursive characters used to train and test the recognizer were extracted from the handwritten words belonging to two different data sets. The first one is the CEDAR[2] database [19]. The second one is a database of handwritten samples collected by the United States Postal Service. In both cases, the data were collected in a postal plant by digitizing handwritten addresses. The characters were extracted from the words through a segmentation process performed by the system in which the recognizer is embedded. Before being segmented, the words were desloped and deslanted following the scheme described in Ref. [20]. The resulting character database *C-Cube* [21] contains 57 293 elements. C-Cube can be downloaded from `ccc.idiap.ch`. The letter distribution, shown in Fig. 3, reflects the prior distribution of the postal plants where the handwritten words were collected. For this reason, some letters are very frequent while others are almost absent. The database was split with a random process into training, validation and

[1] $\vec{w}_i$ stands for $\vec{w}_{c_i}$. This convention is also adopted in the following formulae.

[2] Center of Excellence in Document Analysis and Recognition, State University of New York at Buffalo (USA).

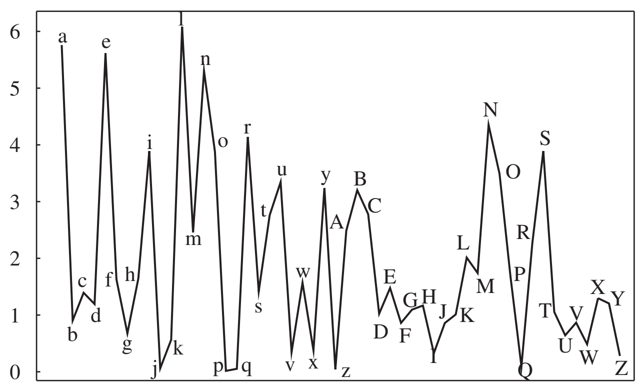

Fig. 3. Letter distribution in the test set.

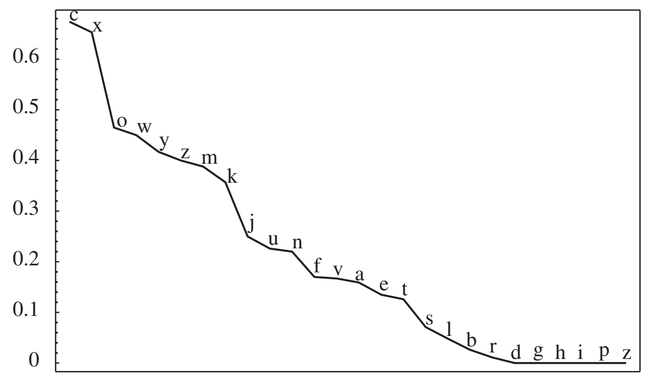

Fig. 4. Value of η for each letter.

test set[3] containing, respectively, 25 440, 12 720 and 19 133 characters.

5.2. Optimal number of classes finding

Clustering allows to verify whether vectors corresponding to the upper and lower case versions of the same letter are distributed in neighboring regions of the feature space or not. The more the two versions of the letter are similar in shape the more their vectors are overlapping (e.g. like *o* and *O*) and can be joined in a single class. On the other hand, when the two versions of a character are very different (e.g. *g* and *G*), it is better to consider them as separate classes.

Clustering was performed by means of NG. We trained different NG maps, selecting the one with minimal *empirical quantization error* [18]. The map neurons were labelled with a kNN technique, namely each node was labelled with the classes of the *k* closest feature vectors. Then the neurons were divided into 26 subsets collecting all the nodes showing at least one version of each letter α among the *k* classes in the label. For each subset, the percentage η_α of nodes having upper and lower case versions of the letter α in the label was calculated. The results are reported, for every subset, in Fig. 4. The percentage can be interpreted as an index of the overlapping of the classes of the upper and lower case versions of the letter. This information can be used to represent the data with a number of different classes ranging from 26 (upper and lower case always joined in a single class) to 52 (upper and lower case always in separate classes). For example, a class number equal to 46 means that, for the six letters showing the highest values of η (i.e. c, x, o, w, y, z) upper and lower case versions are joined in a single class.

[3] The training (*data58train.ptg*) and the test set (*data58test.ptg*) can be downloaded from ccc.idiap.ch. The first two thirds of the first file was used for training, the remaining third for the validation set.

Table 1
SVM recognition rates on the test set, in absence of rejection, for some class numbers

Class number	Performance (%)
52	89.20
38	90.05
26	89.61

Table 2
SVM, LVQ, MLP recognition rates on the test set, in absence of rejection

Model	Class number	Performance (%)
SVM	38	90.05
LVQ	39	84.52
MLP	26	71.42

5.3. Recognition experiments

The percentage η was used to look for the optimal number of classes. The letters showing the highest values of η were represented by a single class containing both upper and lower case versions. We trained SVMs with different number of classes. In each trial we used the gaussian kernel and the variance σ and the regularization constant C were selected by means of *crossvalidation* [22]. In Table 1, for different class numbers, the performances on the test set, measured in terms of recognition rate in absence of rejection, are reported. The performance is shown to be improved by decreasing the number of classes when this is higher than an optimal value, in this case 38. A further reduction of the number of classes results in a lower accuracy. The η parameter is then reliable in estimating the optimal number of classes. In Fig. 5 the confusion matrix of the SVM classifier is shown.

We compared SVM against *learning vector quantization* (*LVQ*) [23,24] and *multi-layer-perceptron* (*MLP*) [25]. SVM and LVQ trials were performed using, respectively, *SVMLight* [26] and *LVQ-pak* [27] software packages. In LVQ trials the learning sequence LVQ1 + LVQ2 + LVQ3 was adopted and the number of codevectors and learning rates were setup using crossvalidation. In MLP trials learning rates and the number of hidden neurons were setup by crossvalidation. LVQ, MLP and SVM recognition rates, in absence of rejection, are reported in Table 2. As shown in Fig. 6, SVM recognizes better than LVQ, 25 letters on 26. The cumulative probability functions of the correct classification for LVQ and SVM are reported in Fig. 7. The probabilities of classification of a character correctly

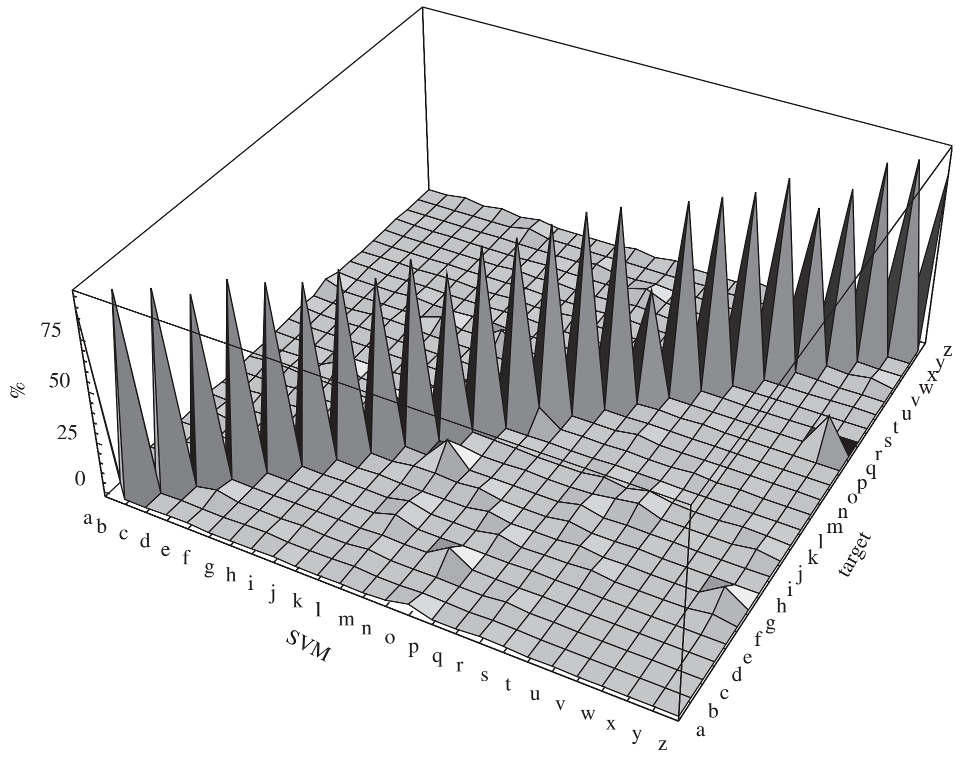

Fig. 5. Confusion matrix of the SVM classifier.

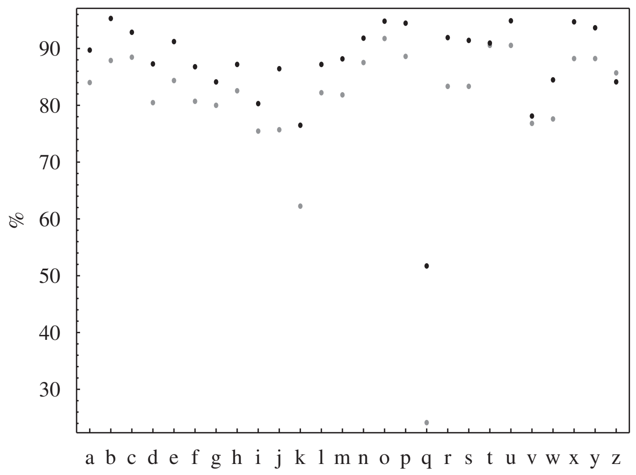

Fig. 6. LVQ (gray points) and SVM (black points) recognition rate, in absence of rejection, for each letter.

in the top, top two and top three positions[4] for LVQ are, respectively, 84.52%, 93.30% and 95.76%; whereas for SVM are 90.05%, 95.73% and 97.31%.

It is difficult to compare results in handwriting recognition due to different databases used in the experimentations. That being said, SVM recognition rate (89.01%) compares significatively better with other results[5] [28–34] on cursive character recognition. Finally, SVM recognition rate is comparable with

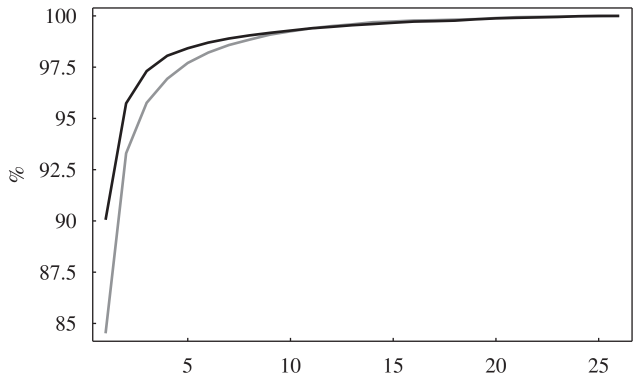

Fig. 7. Cumulative probability function of the correct classification of SVM (black curve) and LVQ (gray curve) classifiers.

the result (90.24%) obtained by Liu and Blumenstein [35] on a smaller test (∼ 2000 characters) than the one (∼ 19 000 characters) used in our experiments.

6. Conclusion

We have presented a cursive character recognizer, a crucial module in cursive word recognition systems based on a segmentation and recognition strategy. The character classification is achieved by using support vector machines (SVMs) and a neural gas (NG). NG allows to obtain a suitable representation of classes, SVMs perform the character recognition. The optimal representation of classes is obtained by evaluating the overlapping in the feature space of the vectors corresponding

[4] For MLP are, respectively, 71.42%, 82.56% and 88.60%.

[5] The best result [28] is 89.01%.

to upper and lower case versions of each letter. When the degree of overlapping is high enough, the two versions can be joined in a single class resulting in an improvement of the classifier performance. SVMs, tested on a large database ($\sim 19\,000$ characters), compare favorably with LVQ (more than 5.50%) and MLP (more than 18%). Moreover, SVM recognition rate is among the highest presented in the literature for cursive character recognition. Finally, the database of cursive characters is available for further comparisons and investigations.

Acknowledgments

First the author wishes to thank the anonymous reviewer for his valuable comments. The author thanks A. Vinciarelli for comments and useful discussions. M. Spinetti is gratefully acknowledged for collecting the character database. The work is dedicated to Carmen Conte for her patience and support.

References

[1] T. Steinherz, E. Rivlin, N. Intrator, Off-line cursive script word recognition—a survey, Int. J. Doc. Anal. Recognition 2 (2) (1999) 1–33.
[2] R. Plamondon, S. Srihari, On-line and off-line handwriting recognition: a comprehensive survey, IEEE Trans. Pattern Anal. Mach. Intell. (2000) 63–84.
[3] A. Senior, A. Robinson, An off-line cursive handwriting recognition system, IEEE Trans. Pattern Anal. Mach. Intell. 20 (3) (1998) 309–321.
[4] G. Kim, V. Govindaraju, A lexicon driven approach to handwritten word recognition for real time applications, IEEE Trans. Pattern Anal. Mach. Intell. 19 (4) (1997) 366–379.
[5] R. Bozinovic, S. Srihari, Off-line cursive script word recognition, IEEE Trans. Pattern Anal. Mach. Intell. 11 (1) (1989) 69–83.
[6] S. Edelman, T. Flash, S. Ullman, Reading cursive handwriting by alignment of letter prototypes, Int. J. Comput. Vision 5 (3) (1990) 303–331.
[7] R. Bellman, S. Dreyfus, Applied Dynamic Programming, Princeton University Press, Princeton, NJ, 1962.
[8] F. Camastra, A. Vinciarelli, Cursive character recognition by learning vector quantization, Pattern Recognition Lett. 22 (6) (2001) 625–629.
[9] R. Bellman, Adaptive Control Processes: A Guided Tour, Princeton University Press, Princeton, NJ, 1961.
[10] C. Berg, J. Christensen, P. Ressel, Harmonic Analysis on Semigroups, Springer, New York, 1984.
[11] C. Cortes, V. Vapnik, Support vector networks, Mach. Learn. 20 (1995) 1–25.
[12] V. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.
[13] R. Herbrich, Learning Kernel Classifiers, MIT Press, Cambridge, USA, 2004.
[14] B. Scholkopf, A. Smola, Learning with Kernels, MIT Press, Cambridge, USA, 2002.
[15] J. Shawe-Taylor, N. Cristianini, Kernels Methods for Pattern Analysis, Cambridge University Press, Cambridge, 2004.
[16] T. Martinetz, S. Berkovich, K. Schulten, "Neural Gas" network for vector quantization and its application to time-series prediction, IEEE Trans. Neural Networks 4 (4) (1993) 558–569.
[17] T. Martinetz, K. Schulten, Topology representing networks, Neural Networks 3 (1994) 507–522.
[18] B. Fritzke, Some competitive learning methods, Technical Report, Ruhr University Bochum, April 1997.
[19] J. Hull, A database for handwritten text recognition research, IEEE Trans. Pattern Anal. Mach. Intell. 16 (5) (1994) 550–554.
[20] G. Nicchiotti, C. Scagliola, Generalised projections: a tool for cursive character normalization, in: Proceedings of the Fifth International Conference on Document Analysis and Recognition, IEEE Press, New York, 1999, pp. 729–732.
[21] F. Camastra, M. Spinetti, A. Vinciarelli, Offline cursive character challenge: a new benchmark for machine learning and pattern recognition algorithms, in: Proceedings of the 18th International Conference on Pattern Recognition (ICPR06), IEEE Press, New York, 2006, pp. 913–916.
[22] M. Stone, Cross-validatory choice and assessment of statistical prediction, J. R. Statist. Soc. 36 (1) (1974) 111–147.
[23] T. Kohonen, Learning vector quantization, in: The Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, MA, 1995, pp. 537–540.
[24] T. Kohonen, Self-Organizing Maps, Springer, Berlin, 1997.
[25] C. Bishop, Neural Networks for Pattern Recognition, Cambridge University Press, Cambridge, UK, 1995.
[26] T. Joachim, Making large-scale SVM learning practical, in: Advances in Kernel Methods—Support Vector Learning, MIT Press, Cambridge, MA, 1999, pp. 169–184.
[27] T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen, K. Torkkola, Lvq-pak: the learning vector quantization program package, Technical Report A30, Helsinki University of Technology, Laboratory of Computer and Information Science, 1996.
[28] M. Blumenstein, X. Liu, B. Verma, An investigation of the modified direction feature for cursive character recognition, Pattern Recognition 40 (2) (2007) 376–388.
[29] B. Verma, M. Blumenstein, M. Ghosh, A novel approach for structural feature extraction: contour vs. direction, Pattern Recognition Lett. 25 (9) (2004) 975–988.
[30] H. Yamada, Y. Nakano, Cursive handwritten word recognition using multiple segmentation determined by contour analysis, IEICE Trans. Inf. Syst. 79 (5) (1996) 464–470.
[31] F. Kimura, N. Kayahara, Y. Miyake, M. Sridhar, Machine and human recognition of segmented characters from handwritten words, in: Proceedings of the Fourth International Conference on Document Analysis and Recognition, IEEE Press, New York, 1997, pp. 866–869.
[32] P. Gader, M. Mohamed, J.-H. Chiang, Handwritten word recognition with character and inter-character neural networks, IEEE Trans. Syst. Man Cybern. Part B: Cybern. 27 (1997) 158–164.
[33] S. Singh, M. Hewitt, Cursive digit and character recognition on cedar database, in: Proceedings of International Conference on Pattern Recognition (ICPR 2000), IEEE Press, New York, 2000, pp. 569–572.
[34] M. Blumenstein, X. Liu, B. Verma, A modified direction feature for cursive character recognition, in: Proceedings of International Joint Conference on Neural Networks 2004, IEEE Press, New York, 2004, pp. 2303–2309.
[35] X. Liu, M. Blumenstein, Experimental analysis of the modified direction feature for cursive character recognition, in: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition, IEEE Press, New York, 2004, pp. 353–358.

About the Author—FRANCESCO CAMASTRA holds a M.Sc. in Physics at University of Milan in 1985 and a Ph.D. in Computer Science at the University of Genova in 2004. From May 1987 to January 2006 he was in R&D Department of Elsag in Genova. In 2005, he was Adjoint Professor at University of Pisa. Since February 2006 he has been Assistant Professor with the Department of Applied Science of the University of Naples "Parthenope". He is the recipient of Eduardo R. Caianiello Award 2005, promoted by the Italian Neural Network Society, for the best Ph.D. Thesis on neural networks and machine learning. He published about 40 papers on peer-reviewed journals and proceedings of conferences. He served as a reviewer for: Control and Intelligent Systems, Cytokine, IEEE Transactions on Neural Networks, IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Transactions on Systems, Man and Cybernetics-B, Information Sciences, Neural Processing Letters, Pattern Recognition and Pattern Recognition Letters. He is member of the IEEE, IEEE Evolutionary Computation Society and Italian Neural Network Society. His research interests are: Machine Learning, Kernel Methods, Manifold Learning, Time Series Prediction.