Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

NEUROCOMPUTING

ELSEVIER

Neurocomputing 51 (2003) 147–159

www.elsevier.com/locate/neucom

Combining neural gas and learning vector quantization for cursive character recognition

Francesco Camastra[a,*], Alessandro Vinciarelli[b]

[a] *INFM - DISI, University of Genova, Via Dodecaneso 35 - 16146 Genova, Italy*
[b] *IDIAP - Institut Dalle Molle d'Intelligence Artificielle Perceptive Rue du Simplon 4, CP592 - 1920 Martigny, Switzerland*

Received 12 June 2001; accepted 8 May 2002

Abstract

This paper presents a cursive character recognizer, a crucial module in any Cursive Script Recognition system based on a segmentation and recognition approach.

The character classification is achieved by combining the use of neural gas (NG) and learning vector quantization (LVQ). NG 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, it is possible to find an optimal number of classes maximizing the accuracy of the LVQ classifier.

A database of 58000 characters was used to train and test the models. The performance obtained is among the highest presented in the literature for the recognition of cursive characters.

Keywords: Learning vector quantization; Neural gas; Self-organizing map; Crossvalidation; Cursive character recognition

1. Introduction

Off-line cursive script recognition (CSR) has several industrial applications such as the reading of postal addresses and the automatic processing of forms, checks and faxes [13,15]. Among other CSR approaches [14,8] one attempts to segment words into letters [3,5]. Since no method is available to achieve a perfect segmentation, a word is first oversegmented, i.e. fragmented into primitives that are characters or parts of them (a perfect segmentation into letters is extremely difficult), then neighboring

* Corresponding author.
E-mail addresses: camastra@disi.unige.it (F. Camastra), alessandro.vinciarelli@idiap.ch (A. Vinciarelli).

0925-2312/03/$ - see front matter
PII: S0925-2312(02)00613-6

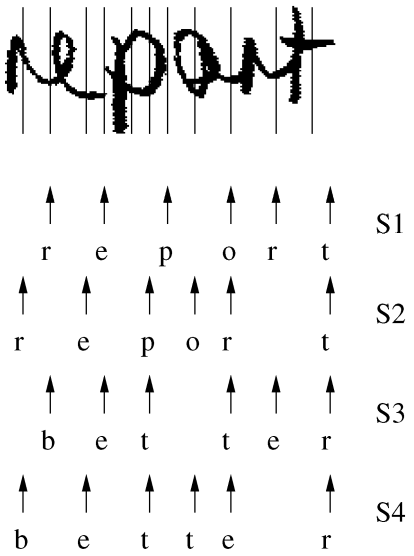

Fig. 1. Score calculation. Different combinations of primitive aggregations. Each combination gives a different score. In our case, the score is the distance between the pattern enclosed by two arrows and the closest LVQ prototype labelled with the same character as the corresponding word letter. It is possible to have several combinations for the same word and several transcriptions of the same combination.

primitives are joined together in all possible combinations (a limit on the number of consecutive fragments that can form a character is usually experimentally determined).

Given a combination where n aggregations of primitives appear, a matching score with all the n-letter long words in the lexicon is calculated. A common way to calculate it is to average over the scores of classifying each aggregation of primitives as the corresponding letter of the lexicon entry under examination (see Fig. 1). In our case the score is the distance between a feature vector extracted from the aggregation of primitives and the closest *learning vector quantization* (LVQ) prototype. The word with the optimal score is found by applying Dynamic Programming techniques [2].

The role of the cursive character recognizer in the above described architecture is crucial. 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 combining the use of *neural gas* (NG) and LVQ. 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 LVQ was selected as classifier because, being a vector quantizer, it yields for each pattern the cost of assigning to a given letter class (in terms of distance from the closest prototype of the class).

This paper is organized as follows: in Section 2 the method for extracting features for character representation is presented; a review of LVQ and NG is provided in

Sections 3 and 4, respectively; Section 5 reports some experimental results; in Section 6 some conclusions are drawn.

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 [4]. 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 [1] covering the whole image. 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 $\sum_i^N n_j^2$ 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° and 90°.

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*, [1]), 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

[1] Small translations of the input patterns can significantly change the distribution of the pixels across the cells. In order to smooth this effect, the cells are partially overlapped.

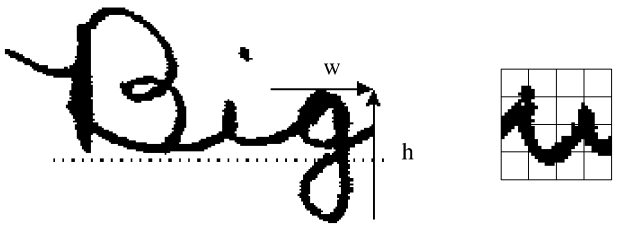

Fig. 2. Global features. The dashed line is the *baseline*, the fraction of h below is used as first global feature. The second global feature is the ratio w/h.

from 16 cells, placed on a regular 4×4 grid; from each cell, the gray feature and one directional feature are extracted.

3. Learning vector quantization

LVQ is a supervised version of vector quantization and generates codevectors to produce "*near-optimal decision boundaries*" [9].

LVQ consists of the application of three consecutive different learning techniques, i.e. LVQ1, LVQ2, LVQ3.[2] LVQ1 uses for classification the nearest-neighbour decision rule; it chooses the class of the nearest codebook vector.

LVQ1 learning is performed in the following way: if $\bar{m}_t^c$[3] is the nearest codevector to the input vector $\bar{x}$, then

$$\begin{aligned} \bar{m}_{t+1}^c &= \bar{m}_t^c + \alpha_t[\bar{x} - \bar{m}_t^c] \quad \text{if } \bar{x} \text{ is classified correctly,} \\ \bar{m}_{t+1}^c &= \bar{m}_t^c - \alpha_t[\bar{x} - \bar{m}_t^c] \quad \text{if } \bar{x} \text{ is classified incorrectly,} \\ \bar{m}_{t+1}^i &= \bar{m}_t^i \qquad\qquad\qquad i \neq c, \end{aligned} \tag{1}$$

where α_t is the learning rate at time t.

Since LVQ1 tends to push codevectors away from the decision surfaces of the Bayes rule, it is necessary to apply to the codebook generated a successive learning technique called LVQ2.

LVQ2 tries harder to approximate the Bayes rule by pairwise adjustments of codevectors belonging to adjacent classes. If $\bar{m}^s$ and $\bar{m}^p$ are nearest neighbours of different classes and the input vector $\bar{x}$, belonging to the $\bar{m}^s$ class, is closer to $\bar{m}^p$ and falls into a zone of values called *window*,[4] the following rule is applied:

$$\bar{m}_{t+1}^s = \bar{m}_t^s + \alpha_t[\bar{x} - \bar{m}_t^s], \bar{m}_{t+1}^p = \bar{m}_t^p - \alpha_t[\bar{x} - \bar{m}_t^p]. \tag{2}$$

Since the application of LVQ2 tends to overcorrect the class boundaries, it is necessary to include additional corrections that ensure that the codebook continues approximating the class distributions. In order to assure that, it is necessary to apply a further algorithm (LVQ3).

[2] LVQ2 and LVQ3 were proposed, on empirical basis, in order to improve LVQ1 algorithm.

[3] $\bar{m}_t^c$ stands for the value of $\bar{m}^c$ at time t.

[4] The window is defined around the midplane of $\bar{m}^s$ and $\bar{m}^p$.

If $\bar{m}^i$ and $\bar{m}^j$ are the two closest codevectors to input $\bar{x}$ and $\bar{x}$ falls in the window, the following rule is applied: [5]

$$
\begin{aligned}
\bar{m}^i_{t+1} &= \bar{m}^i_t && \text{if } C(\bar{m}^i) \neq C(\bar{x}) \wedge C(\bar{m}^j) \neq C(\bar{x}),\\
\bar{m}^j_{t+1} &= \bar{m}^j_t && \text{if } C(\bar{m}^i) \neq C(\bar{x}) \wedge C(\bar{m}^j) \neq C(\bar{x}),\\
\bar{m}^i_{t+1} &= \bar{m}^i_t - \alpha_t[\bar{x}_t - \bar{m}^i_t] && \text{if } C(\bar{m}^i) \neq C(\bar{x}) \wedge C(\bar{m}^j) = C(\bar{x}),\\
\bar{m}^j_{t+1} &= \bar{m}^j_t + \alpha_t[\bar{x}_t - \bar{m}^j_t] && \text{if } C(\bar{m}^i) \neq C(\bar{x}) \wedge C(\bar{m}^j) = C(\bar{x}),\\
\bar{m}^i_{t+1} &= \bar{m}^i_t + \alpha_t[\bar{x}_t - \bar{m}^i_t] && \text{if } C(\bar{m}^i) = C(\bar{x}) \wedge C(\bar{m}^j) \neq C(\bar{x}),\\
\bar{m}^j_{t+1} &= \bar{m}^j_t - \alpha_t[\bar{x}_t - \bar{m}^j_t] && \text{if } C(\bar{m}^i) = C(\bar{x}) \wedge C(\bar{m}^j) \neq C(\bar{x}),\\
\bar{m}^i_{t+1} &= \bar{m}^i_t + \varepsilon\alpha_t[\bar{x}_t - \bar{m}^i_t] && \text{if } C(\bar{m}^i) = C(\bar{m}^j) = C(\bar{x}),\\
\bar{m}^j_{t+1} &= \bar{m}^j_t + \varepsilon\alpha_t[\bar{x}_t - \bar{m}^j_t] && \text{if } C(\bar{m}^i) = C(\bar{m}^j) = C(\bar{x}),
\end{aligned}
\tag{3}
$$

where $\varepsilon \in [0,1]$ is a fixed parameter.

LVQ3 is self-stabilizing, i.e. the optimal placement of the codebook does not change while continuing learning.

4. Neural gas

NG in Martinetz et al. [10] is an unsupervised version of vector quantization. In neural gas model, in contrast to SOM, no topology of a fixed dimensionality is imposed on the network. Neural Gas consists of a set of M units: $A = (c_1, c_2, \ldots, c_M)$. Each unit c_i has an associated *reference vector* $w_{c_i}(w_{c_i} \in R^n)$ indicating its position or *receptive field center* in input space. The learning algorithm of the neural gas is the following:

(1) Initialize the set A to contain units c_i, with $w_{c_i} \in R^n$, chosen randomly according to input distribution $p(\xi)$. Besides, initialize the time parameter t, to 0.
(2) Generate at random an input ξ according to $p(\xi)$.
(3) Order all elements of A according to the distance of their reference vectors to ξ, e.g., find the sequence of indices $S = (i_0, i_1, \ldots, i_{M-1})$ such that w_{i_0} is the reference vector closest to ξ, w_{i_1} is the second vector closest to ξ etc. Let $k_i(\xi, A)$ the rank associated with w_i. [6]
(4) Adapt the reference vectors according to

$$\Delta w_i = \varepsilon(t) h_\lambda(k_i(\xi, A))(\xi - w_i),$$

where

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

[5] $C(\bar{q})$ stands for the class of $\bar{q}$.
[6] w_i stands for w_{c_i}. This convention is also adopted in the following formulae.

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

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

(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 [6,10,11].

5. Experiments and results

A combination of NG and LVQ is shown to improve the performance of a cursive character recognizer.

The letters of the database are both upper and lower case and this suggests that the number of classes could be 52. On the other hand, not all the characters have a different shape in the two versions. In some cases (e.g. *o* and *O*) the two letters differ only for their size. When the feature extraction process is (as in our system) not sensitive to the size, such letters could be joined in a single class. Their vectors are in fact expected to be distributed in the same region of the feature space.

Since trying to separate overlapped classes can determine overfitting, finding a suitable class representation is helpful to achieve good performances.

In our experiments, we tried two clustering algorithms (NG and SOM) to find the classes that could be joined. The clustering tries to model the data distribution and it is a suitable technique for finding the classes sharing the same region of the feature space.

More details about the steps of the above described process are given in the following three subsections. 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 [7] database [7]. 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 character recognizer is embedded in a cursive word recognition system using a segmentation and recognition approach. For most of

[7] Center of Excellence in Document Analysis and Recognition, State University of New York at Buffalo (USA). All the words belonging to directories `train/cities` and `train/states` were used.

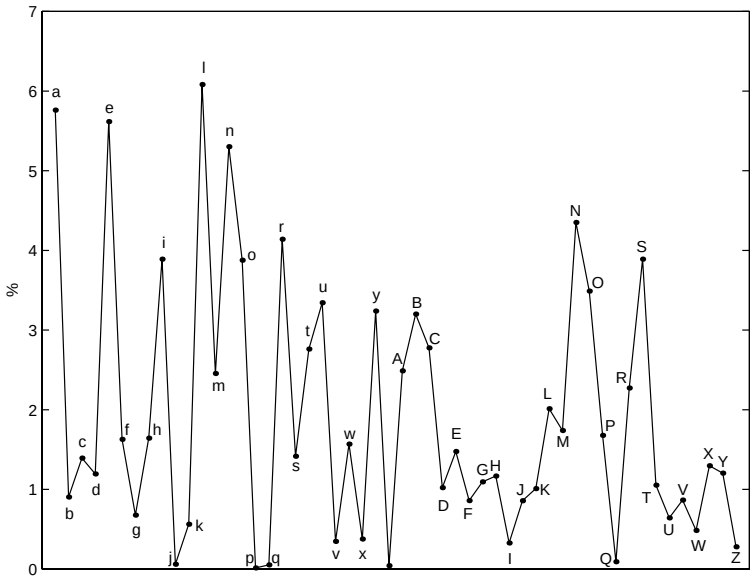

Fig. 3. Letter distribution in the test set.

the attempted segmentations, the word fragments to be recognized do not correspond to actual letters (see Fig. 1). The score at the word level is calculated by averaging over the scores corresponding to classifying each fragment as a letter of the word. Because the word scores are expected to be lower when the segmentation is bad, no rejection mechanism was added at the letter level.

The characters are extracted from the words through a segmentation process performed by the system in which the recognizer is embedded. Before being segmented, the words are desloped and deslanted following the scheme described in [12]. The resulting character database contains 58000 samples. 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 is split with a random process into training, validation and test set containing respectively 25 000, 13 000 and 20 000 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.

Table 1
Quantization error of SOM and NG for different reference vector numbers

Neurons	SOM (q.error)	NG (q.error)
1300	0.197290	0.162117
900	0.204126	0.167981
600	0.209687	0.175800
400	0.217515	0.182672
200	0.227662	0.195116
100	0.242464	0.208706

Clustering was performed by means of NG and self-organizing map (SOM). In Table 1 the performances of different SOM and NG maps, measured in terms of *quantization error*[8] on the whole character database, are reported. Given a number of reference vectors, the NG always performs better than the SOM and is, for this reason, selected.

The number of reference vectors in the map can be set up by finding an optimal tradeoff between quantization and generalization error. The quantization error can be decreased by simply adding reference vectors (in the extreme case, each training sample can be a reference vector, resulting in a null quantization error), but after a certain limit, this leads to overfit the map over the training set.

To avoid this problem, the generalization error must be estimated [17]. The recognition error (the percentage of characters misclassified) on the test set can provide an estimate of the generalization error. In our experiments, the optimal number of reference vectors is 1300.

The reference vectors were labelled with a kNN technique[9] and 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 is an index of the overlapping of the classes of the uppercase and lowercase versions of the letter. This information can be used to represent the data with a number of different classes ranging from 26 (uppercase and lowercase always joined in a single class) to 52 (uppercase and lowercase 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) uppercase and lowercase versions are joined in a single class.

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

[8] Using the notation adopted in Section 4, the quantization error Q_E is defined as follows: $Q_E = \sum_{i=1}^{N} \sum_{\xi_j \in V(w_i)} \|\xi_j - w_i\|_{L_2}$ where $V(w_i)$ is the Voronoi set of codevector w_i.

[9] Each node is labelled with the classes of the k closest feature vectors.

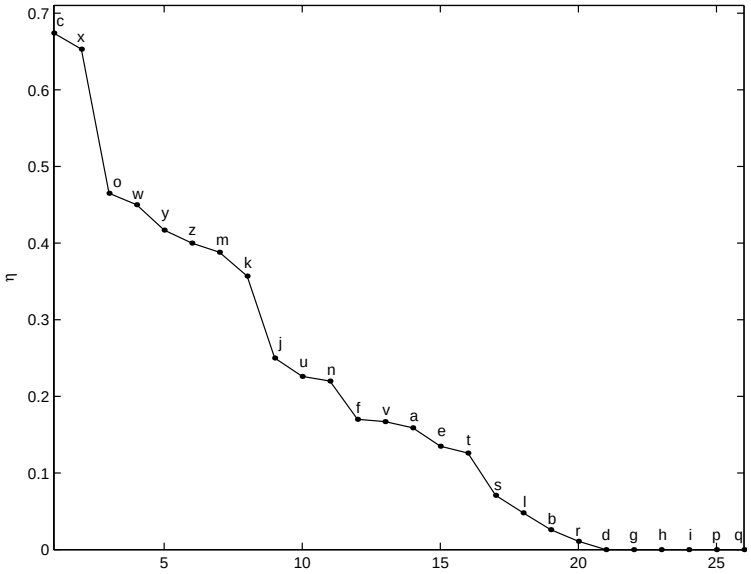

Fig. 4. Value of η for each letter.

both upper and lower case versions. We trained LVQ nets with different number of classes. In each trial, the number of codevectors and the learning rate were selected by means of cross-validation [16] and the learning sequence LVQ1+LVQ2+LVQ3 was adopted. The number of LVQ codevectors, assigned to each class, was proportional to the a-priori class probability.

In Table 3, 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 39). 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.

This is confirmed by looking at the performance of the recognizer over the single characters. Table 2 reports the change in recognition rate of each character when passing from 52 to 39 classes. The characters are ordered following the value of η. The first 13 classes show in most cases a significant accuracy improvement. The other classes show smaller changes in both positive and negative directions.

The effect on the overall performance of the recognizer is influenced by the letter distribution. In our case, the letters showing the highest improvements are not enough represented to significantly affect the overall accuracy, but in other cases, the distribution can be different and the improvement of the recognition rate much higher.

Our best result in terms of recognition rate is 84.52%. The only result we know [18], obtained on a smaller test, is approximately 75%. In Fig. 5 the confusion matrix is shown. The cumulative probability function of the correct classification is reported in Fig. 6. The probabilities of classification of a character correctly top three and top

Table 2
Change in the recognition rate when passing from 52 (lower and upper case versions of the letter never joined in a single class) to 39 classes (lower and upper case versions of the letter joined in a single class for the first 13 letters)

Class	52 classes	39 classes
c	88.99	88.47
x	84.40	88.23
o	90.19	91.76
w	71.39	77.60
y	87.94	88.23
z	65.90	85.71
m	72.54	81.84
k	59.86	62.25
j	70.73	75.70
u	90.55	90.55
n	88.13	87.53
f	80.50	80.71
v	73.81	76.82
a	83.62	84.01
e	82.30	84.36
t	87.94	90.55
s	81.87	83.34
l	83.25	82.22
b	88.15	87.89
r	85.80	83.34
d	80.94	80.47
g	80.58	80.00
h	83.11	82.56
i	75.98	75.46
p	91.38	88.61

The characters are ordered following the value of η.

Table 3
Recognition rates on the Test Set, in absence of rejection, for several class numbers

Class number	Performance
52	83.74
46	83.91
42	84.25
41	84.27
39	84.52
36	84.38
26	84.27

twelve positions are, respectively, 95.76% and 99.50%. In our opinion, the fundamental sources of misclassification for our classifier are two. The first one (for the most rare letters) is the low number of available samples. The second is the intrinsic ambiguity in cursive characters. In fact, some couples of letters (e.g. e/l or a/o) are very difficult to be distinguished. This is confirmed by the confusion matrix and by the high recognition rate in the top three positions.

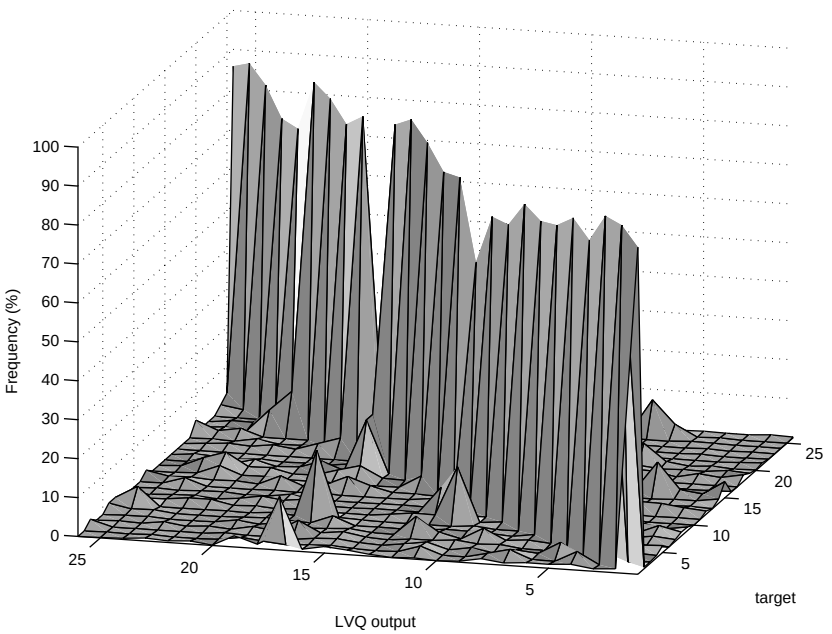

Fig. 5. Confusion matrix of the LVQ classifier. The letters are mapped to the numbers from 1 to 26. The letter a is mapped to 1, b is mapped to $2,\ldots,z$ to 26.

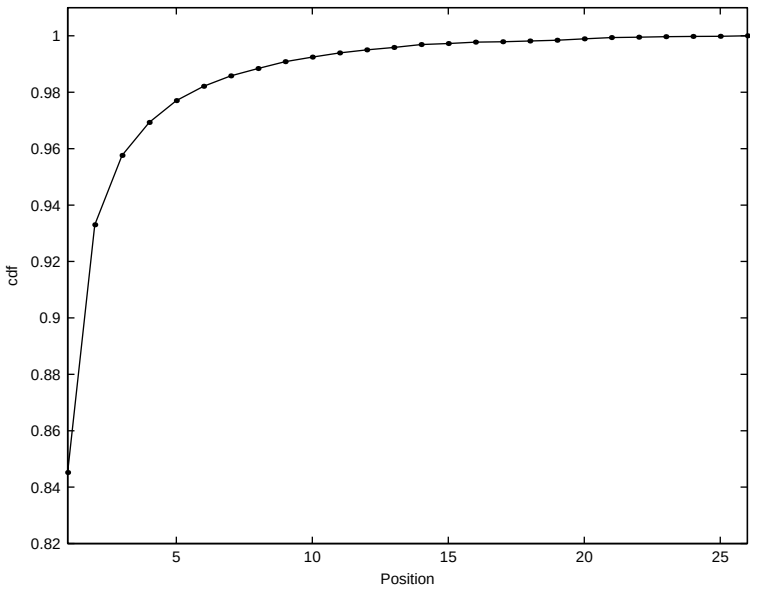

Fig. 6. Cumulative probability function of the correct classification of LVQ classifier.

6. Conclusion

We have presented a cursive character recognizer, a crucial module in Cursive Script Recognition systems based on a segmentation and recognition approach. An improvement of the correct classification rate is obtained by combining a clustering algorithm with a classifier. Since in our experiments NG performs better than SOM, it is selected as clustering algorithm. LVQ is used as classifier. The optimal representation of classes is obtained by evaluating the overlapping in the feature space of the vectors corresponding 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.

The accuracy achieved is the highest presented, to our knowledge, in the literature.

Acknowledgements

This work was funded by the Italian Ministry of University and Technological Research under the grant to Parco Scientifico e Tecnologico dell'Elba, Project No. 62413 "Dispositivi con reti neurali e sensori per riconoscimento voci e teleoperazioni varie".

First the authors wish to thank the anonymous reviewers for their valuable comments. The authors also thank S. Bengio, D. Ugolini, and A. Verri for comments and encouragements. M. Spinetti is gratefully acknowledged for collecting the character database.

References

[1] R. Bellman, Adaptive Control Processes: A Guided Tour, Princeton University Press, Princeton, NJ, 1961.
[2] R. Bellman, S. Dreyfus, Applied Dynamic Programming, Princeton University Press, Princeton, NJ, 1962.
[3] R.M. Bozinovic, S.N. Srihari, Off-line cursive script word recognition, IEEE Trans. Pattern Anal. Mach. Intell. 11 (1) (1989) 69–83.
[4] F. Camastra, A. Vinciarelli, Cursive character recognition by Learning Vector Quantization, Pattern Recognition Lett. 22 (6) (2001) 625–629.
[5] S. Edelman, T. Flash, S. Ullman, Reading cursive handwriting by alignment of letter prototypes, Int. J. Comput. Vision 5 (3) (1990) 303–331.
[6] B. Fritzke, Some competitive learning methods, Technical Report, Ruhr University Bochum, April 1997.
[7] J.J. Hull, A database for handwritten text recognition research, IEEE Trans. Pattern Anal. Mach. Intell. 16 (5) (1994) 550–554.
[8] 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.
[9] T. Kohonen, Self-organizing Maps, Springer, Berlin, 1997.
[10] 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.
[11] T. Martinetz, K. Schulten, Topology representing networks, Neural Networks 3 (1994) 507–522.
[12] G. Nicchiotti, C. Scagliola, Generalised projections: a tool for cursive character normalization, in: Proceedings of Fifth International Conference on Document Analysis and Recognition, IEEE Press, New York, 1999.

[13] R. Plamondon, S. Srihari, On-line and off-line handwriting recognition: a comprehensive survey, IEEE Trans. Pattern Anal. Mach. Intell. 22 (1) (2000) 63–84.
[14] A.W. Senior, A.J. Robinson, An off-line cursive handwriting recognition system, IEEE Trans. Pattern Anal. Mach. Intell. 20 (3) (1998) 309–321.
[15] T. Steinherz, E. Rivlin, N. Intrator, Off-line cursive script word recognition—a survey, Int. J. Document Anal. Recognition 2 (2) (1999) 1–33.
[16] M. Stone, Cross-validatory choice and assessment of statistical prediction, J. Roy. Statist. Soc. 36 (1) (1974) 111–147.
[17] V. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.
[18] H. Yamada, Y. Nakano, Cursive handwritten word recognition using multiple segmentation determined by contour analysis, IEICE Trans. Inf. Systems 79 (5) (1996) 464–470.

Francesco Camastra was born in Polignano (Bari), Italy, in 1960. He took an M.Sc. in Physics at University of Milan in 1985. He joined in R&D Department of Elsag, in Genova in 1987. Since 1992 he has researched on Neural Networks. In particular, he applied neural networks to state estimation, predictive control, time series prediction and handwriting recognition. Since 2001, as a Ph.D. student, he is with Dipartimento d' Informatica e Scienze dell' Informazione (DISI) of the University of Genova. He is also an affiliate of Istituto Nazionale Fisica della Materia (INFM unit of Genova) since 2001. He served as a reviewer for Neural Processing Letters and IEEE Transaction on Neural Networks. He is member of European Neural Network Society (ENNS) and Italian Neural Network Society (SIREN). His current research interests are: Machine Learning, Kernel Methods, and Nonlinear Dynamic Theory.

Alessandro Vinciarelli (M.Sc. in Physics, University of Torino, 1994). After working in several companies and laboratories (Andersen Consulting, Polo Nazionale Bioelettronica, International Institute for Advanced Scientific Studies), he is with IDIAP (Martigny, Switzerland) as a Ph.D. student since 1999. He served as a reviewer for Pattern Recognition Letters and the International Journal on Document Analysis and Recognition. His research interests include OCR, Neural Networks, Hidden Markov Models, Machine Learning and Cursive Handwriting Recognition.