A Prototype Classification Method and Its Application to Handwritten Character Recognition*

Fu Chang†, Chien-Hsing Chou†, Chin-Chin Lin‡, and Chun-Jen Chen†
†Institute of Information Science, Academia Sinica, Taipei, Taiwan
‡Dept. of Electrical Engineering, National Taipei University of Technology, Taipei, Taiwan
Email: {fchang, ister, erikson, dean}@iis.sinica.edu.tw

Abstract - *We propose a new prototype classification method that can be combined with support vector machines (SVM) [1] for recognizing handwritten numerals and Chinese characters. This method employs a learning process for determining both the number and location of prototypes. The possible techniques used in this process for adjusting the location of prototypes include the K-means (KM) algorithm and the fuzzy c-means (FCM) algorithm [2]. When the prototype classification method is applied, the SVM method can be used to process top-rank candidates obtained in the prototype learning or matching process. We apply this hybrid method to the recognition of handwritten numerals and Chinese characters. Experiment results show that this hybrid method saves great amount of training and testing time when the number of character types is large, and achieves comparable accuracy rates to those achieved by using SVM solely. Our results also show that the proposed method performs better than the nearest neighbor (NN) classification method. These outcomes suggest that the proposed method can serve as an effective solution for large-scale multiclass classification.*

Keywords: Fuzzy c-means clustering algorithm, handwritten character recognition, K-means clustering algorithm, support vector machine, prototype learning

1 Introduction

In pattern recognition, one deals with either binary classification in which each object is classified as one of two classes, or multiclass classification in which each object is classified as one of N classes, $N > 2$. SVM is very effective for binary classification and it can be used for multiclass classification by decomposing the problem into binary classification sub-problems. Two useful methods for decomposing the problem are one-against-one [3] and DAGSVM [4]. In the training phase, both methods require solving $N(N\text{-}1)/2$ binary classification problems. In the testing phase, the one-against-one technique conducts $N(N\text{-}1)/2$ classifications, while DAGSVM technique employs a directed acyclic graph that has $N(N\text{-}1)/2$ nodes and N leaves, reducing the number of classifications to N-1. Both methods are subject to the drawback that, when the number of classes N is large, they incurs exhaustive amount of training time and produce an extremely large set of support vectors.

From our experience, we estimate that the training of an SVM classifier for 3,036 character types out of 303,600 samples takes 32 days at a PC of Pentium IV 2.4G CPU. In the testing phase, DAGSVM requires 0.37 second to recognize a character and requires storing 1.5×10^8 support vectors on the RAM. On the other hand, a hybrid classifier proposed in this paper requires only 245,421 seconds or 68.2 hours to complete the training. In the testing phase, it takes merely 0.002 second to recognize a character and requires storing only 6.6% of support vectors that are required by one-against-one or DAGSVM.

This paper is organized as follows. Section 2 contains the formulation for the prototype construction problem, and proposed learning algorithms as solutions to the problem. In Section 3, we describe the disambiguation process using SVM for training and matching. Section 4 details the application of our hybrid method to handwritten character recognition. In Section 5, we provide training and testing results, and also make comparisons with all alternative methods. In Section 6, we present our conclusion.

2 Prototype-Construction Method

We propose two algorithms for determining the number and the location of prototypes. They employ a similar learning process but differ in the way of computing the location of prototypes and in the criterion for stopping the process. We first state the process that uses KM for adjusting the location of prototypes.

Step 1 Initiation: For each class type C, we use the statistical average of all C-samples to initiate a C-prototype.

Step 2 Absorption: For each sample s, find the prototype p that is nearest to s. If the class type of s matches with the class type of p, declare s as *absorbed.* Otherwise, s is unabsorbed.

* **0-7803-8566-7/04/$20.00 © 2004 IEEE.**

Step 3 Prototype Augmentation: For each class type C, if there exists any unabsorbed C-samples, we select one of these samples as a new C-prototype. Otherwise, no new prototype is added to class type C.

Step 4 Prototype Adjustment: For the class type C to which a new C-prototype is added in Step 3, we apply KM to adjust all C-prototypes, using the added and also existing C-prototypes as seeds.

Step 5 Stopping Criterion: If there are unabsorbed samples, go to step 2. Otherwise, we stop the process.

The above learning process assures that each sample is absorbed within a finite number of iterations. If FCM, instead of KM, is used in Step 4, then there is no assurance for the eventual absorption of all samples. To cope with this situation, we modify the process as follows. When an unabsorbed sample s is added as a C-prototype in Step 3, we check whether this addition helps to reduce any unabsorbed C-samples. If not, s changes its status from unabsorbed to *futile*, and we restore all old C-prototypes. The process stops when all samples are either absorbed or futile.

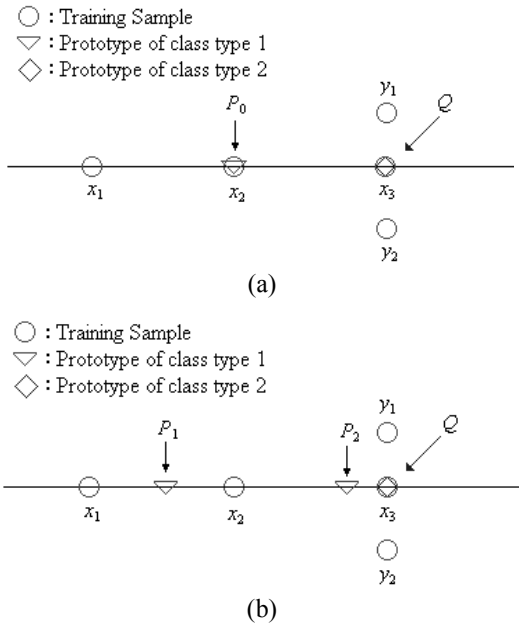


Figure 1. (a) Initially, a type-1 prototype P_0 and a type-2 prototype Q are created. (b) Two type-1 prototypes P_1 and P_2 are created.

We show with an example that the algorithm using FCM can fail to converge if the old stopping criterion is used. In Figure 1a, there are three type-1 samples x_1, x_2 and x_3 and two type-2 samples y_1 and y_2. Using Step 1, we obtain initially a type-1 prototype P_0 and a type-2 prototype Q. Samples x_1 and x_3 keep an equal horizontal distance to x_2, so P_0 locates exactly at the same position as x_2. Samples y_1 and y_2 keep an equal vertical distance to x_3, so Q locates at the same position as x_3. The type-1 sample x_3 is unabsorbed, since its distance to the type-2 prototype Q is shorter than its distance to the type-1 prototype P_0. All other samples are absorbed. Following Step 3 and 4 and using FCM rather than KM, we create two new type-1 prototypes P_1 and P_2 to replace P_0, as shown in Figure 1b. In this new setting, sample x_3 remains unabsorbed, while other samples are absorbed. If the old stopping criterion is used, we will continue to create more type-1 prototypes in later iterations. All these prototypes lie left to x_3 and none of them locate at the same position as x_3. So x_3 remains unabsorbed in all iterations. Our stopping criterion for FCM remedies this problem, since x_3 is marked as futile and no more prototypes will be created for it.

3 Disambiguation Using SVM

The prototype-matching method achieves very high accuracy rates for k nearest prototypes for fairly large k, but has a noticeable gap between top-k and top-1 accuracy rates. The disambiguation procedure bridges this gap. There are some requisites for the training and testing process. In the training process, we must determine which class types can be mistaken for another during the prototype-matching process. These types are always paired and are therefore referred to as *confusing pairs*. For these pairs, we have to specify *reassessing schemes* using an SVM method. We use these schemes in the testing process to reassess the top-rank candidates for each object.

Recall that, in the prototype construction process, we must determine the nearest prototype for each training sample s. At the end of the process, we find k nearest prototypes for each s. The *class types* of these k prototypes will be referred to as *candidates* of s. In the training process, k_0 is a small integer, but is not necessarily the same as k_1 in the testing process, in which k_1 candidates of test samples are reassessed. We collect the pairs (C_i, C_j), where C_i and C_j are i^{th} and j^{th} candidates of s for $1 \le i, j \le k_0$.

For each confusing pair (C, D) and its training samples, we use SVM to create a reassessing scheme. The purpose of the SVM is to provide *decision functions* for classifying objects into class C or D, where the parameters and support vectors that appear in the decision function are derived from an optimization problem using training samples of C and D as components. Details are given in *The Nature of Statistical Learning Theory* (Vapnik [5]). For handwritten character recognition, we adopt the dual formulation of the optimization problem using the polynomial kernel of degree 2 for the choice of kernel function. In *The Nature of Statistical Learning Theory*, comparisons of SVM and other methods for classifying UPS handwritten numerals are given. SVM is shown to perform competitively.

After completing the training process by determining the reassessing scheme for each confusing pair, we can address the testing process. Suppose that an object O is given and its first k_1 candidates have already been found. We apply reassessing schemes to all confusing pairs found within the top-k_1 candidates of O. When the confusing pair is (C, D) and the unknown object is classified as C, then C scores one unit. When all the confusing pairs in the candidate list are reassessed, we re-order the involved candidates. The candidate with the highest score is ranked first; the candidate with the second highest score is ranked second, and so on. If two candidates receive the same score, their relative positions remain the same as before. We then rearrange the involved candidates according to their assigned ranks.

4 Feature Extraction Methods

To test the effectiveness of our method, we apply it to the recognition of handwritten characters. We aim to test the hybrid method and compare it with some alternatives. For this purpose, we employ two types of feature extraction method. The first consists of three feature-matching techniques: non-linear normalization, directional feature extraction and feature blurring. It is referred to as 'direction' feature. According to Umeda [6], these techniques are major breakthroughs in handwritten Chinese character recognition. The second method is the 'density' feature extraction technique. Details about each technique are given in the following sub-sections.

4.1 Nonlinear Normalization

The shape normalization method (Lee and Park [7]) transforms a binary image of size $I \times J$ into a normalized image of size $M \times N$. The transformation is expressed as

$$m = \sum_{k=1}^{i} H(k) \times \frac{M}{\sum_{k=1}^{I} H(k)}, \quad n = \sum_{k=1}^{j} V(k) \times \frac{N}{\sum_{k=1}^{J} V(k)} \quad (1)$$

where $i = 1, 2, \ldots, I$; $j = 1, 2, \ldots, J$; $m = 1, 2, \ldots, M$; $n = 1, 2, \ldots, N$. If $H(i) = V(j) = 1$, we obtain linear normalization.

We adopt Yamada's method (Yamada et al. [8]) for nonlinear normalization. In this method, $H(i)$ accumulates line density values along a vertical line whose x-coordinate is i. The line density at a point derives its value from an inscribed circle centered at that point; the wider the circle, the lower the line density. $V(j)$ is similarly defined. We also adopt a modification of Yamada's method, proposed by Liu et al. [9], which adjusts undue distortion of the peripheral shape in four boundaries of the character image. In our applications, we normalize each original character image into a 64×64 image. Thus, we have $M = N = 64$ in (1).

4.2 Directional Feature Extraction

This operation assigns directional attributes to all points in the normalized character image. To assign directional attributes, we use the chain code (Freeman [10]) as shown below.

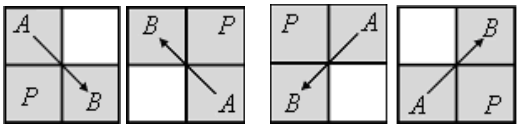
(1) Extract the character contour using a contour-tracing algorithm proposed by Haig and Attikiouzel [11] and modified by Chang and Chen [12].

(2) To each contour point A and a neighboring contour point B, assign a directional attribute to A and B according to their relative positions. The assignment is shown in Figure 2. For example, if B lies to the right or left of A, both A and B are assigned attribute 1. Since a contour point has two neighbors, each contour point can have more than one directional attribute.

2	0	3
1	*A*	1
3	0	2

Figure 2. Directional attribute assignment. If B lies at the position labelled 3, the common directional attribute of A and B is 3.

(3) Assign a non-negative directional attribute to each non-contour point P whose neighbors A and B are contour points with directional attribute 2 or 3 (Figure 3). P is assigned three directional attributes: 0, 1, and 2, if A and B have directional attribute 2; or 0, 1, and 3, if A and B have directional attribute 3. The purpose of these assignments is to include all the attribute ambiguities associated with P. Assign all remaining points directional attribute –1.



(a) (b)

Figure 3. P is not a contour point, but its two neighbors A and B are. The arrow indicates the order in which contour points are traced. (a) The common directional attribute of A and B is 2. (b) The common directional attribute of A and B is 3.

In our applications, we produce four *feature images* F_i, $i = 0$, 1, 2, and 3 out of a given normalized character image E, where each F_i has the same size as E. The feature image F_i collects points in E that have directional attribute i. These points are assigned 1 (black) in F_i. The remaining points in F_i are assigned 0 (white).

4.3 Blurring

The four feature images F_i, i = 0, 1, 2, and 3, are further processed by a blurring technique, which absorb possible local displacements within characters. We adopt the blurring mask proposed by Liu et al. [9]. As shown in Figure 4, this 4×4 mask approximates a Gaussian mask that serves as an optimal low-pass filter.

Before applying this mask, we first reduce each 64×64 feature image F_i into a 16×16 image G_i, where i = 0, 1, 2, 3. The reduction is done in such a way that each 4×4 block reduces to a point whose value is the number of 1's in this block. We add a row of zeros above and below G_i, and a column of zeros to its left and right. We call this augmented image H_i. The mask is then applied to every 4×4 block in H_i whose upper left point is (x, y), where both x and y are even numbers. We thus obtain 64 values from each H_i, and 256 values in total. These 256 values form the feature vector that we extract from each character image.

γ	β	β	γ
β	α	α	β
β	α	α	β
γ	β	β	γ

Figure 4. The blurring mask used to extract feature vectors, where α = 0.1444, β = 0.0456 and γ = 0.0144.

4.4 Density Feature Extraction

This method reduces a 64×64 original image to a 16×16 image. Each number in the latter image derives its value from the sum of 1's in a 4×4 block of the original image. The density feature vector then consists of 16 components, whose values range from 0 to 255.

5 Experiment Results

To test the effectiveness of the proposed method, we apply it to the recognition of handwritten characters. There are six databases we use in our experiments. They can be divided into three groups in terms of the number of class types involved. The first group contains two *small-scale* classification tasks. The databases we employ are UPS [13] and CENPARMI [14] handwritten numerals. The second group consists of two *middle-scale* classification tasks, in each of which there are 350 class types of handwritten Chinese characters that are taken from ETL8B and ETL9B databases. [15] Each class type contains the same number of samples as the original database. The third group consists of two *large-scale* classification tasks, in which we use full sets of ETL8B (comprised of 956 character types) and ETL9B (comprised of 3,036 character types).

In Table 1, we list the number of class types (# CS), the number of training samples (# TrS) and the number of test samples (# TeS) in each application. In our experiments, only UPS database uses the 'Density' feature, as described in Section 4.4. For the other databases except UPS, we use the feature extraction method consisting of nonlinear normalization, directional feature extraction and blurring described in Section 4.1–4.3. According to Umeda [6], these techniques are major breakthroughs in handwritten Chinese character recognition.

Table 1. Number of class types, training samples and test samples in the six applications.

Database	# CS	# TrS	# TeS
UPS	10	7,291	2,007
CENPARMI	10	4,000	2,000
ETL8B (Subset)	350	28,000	28,000
ETL9B (Subset)	350	35,000	35,000
ETL8B (Full Set)	956	76,480	76,480
ETL9B (Full Set)	3,036	303,600	303,600

In Table 2 to 7, the training and testing results of the two hybrid classifiers, KM+SVM and FCM+SVM, are displayed. For comparison, we employ NN (which uses all training samples as prototypes and finds the nearest one to each test sample) and DAGSVM as classifiers to produce similar outputs. These results show that the two hybrid classifiers achieve comparable accuracy rates to the DAGSVM classifier, while consuming much less training and testing time, and producing much less number of support vectors. Furthermore, three classifiers (KM+SVM, FCM+SVM, DAGSVM) achieve better accuracy rates than the NN classifier. In small-scale and middle-scale tasks, using DAGSVM solely for classification is feasible. However, for large-scale classification, DAGSVM takes an extremely long time for training (for example, it takes 32 days to complete the training for the ETL9B full set) and is, therefore, not used.

In Table 2 to 7, we also display the results of prototype classifiers (i.e., classifiers that match objects against prototypes). There are two such classifiers, the KM classifier and the FCM classifier, corresponding to the prototype learning algorithms using KM and FCM for adjusting the location of prototypes. The results show that the two prototype classifiers perform less well when they are used solely than when they are combined with SVM to form hybrid classifiers. The results also show that the FCM classier performs better than the KM classifier. The FCM classifier even outperforms the NN classifier for ETL8B and ETL9B databases (subset and full set). The last finding is interesting, for it suggests that if SVM is not used as a post-process, the FCM classifier should be employed, rather than the KM classifier. Note that, for fuzzy c-means method, we apply an efficient implementation proposed by Kolen and Hutcheson [16].

Table 2. Experiment results for UPS database.

	UPS (10 Numeral Types)					
	Training (7,291 Samples)				Testing (2,007 Samples)	
	Time (s)	# PRs	# SVs	# CPs	Time (s)	Acc
NN		7,291			14.84	94.47%
DAGSVM	139		7,233	45	9.96	**95.47%**
KM	122	**393**			**0.8**	92.37%
FCM	1620	430			0.9	93.82%
KM+SVM	261 (122+139)	**393**	7,233	45	3.9 (0.8+3.1)	95.22%
FCM+SVM	1,759 (1620+139)	430	7,233	45	4.0 (0.9+3.1)	95.22%

PRs: Number of prototypes; # SVs: Number of Support Vectors; # CPs: Number of Confusing Pairs; Acc: Accuracy Rates.

Table 3. Experiment results for a subset of CENPARMI database.

	CENPARMI (10 Numeral Types)					
	Training (4,000 Samples)				Testing (2,000 Samples)	
	Time (s)	# PRs	# SVs	# CPs	Time (s)	Acc
NN		4,000			12.89	96.45%
DAGSVM	50.4		2,753	45	5.96	97.35%
KM	22.2	**214**			**0.69**	95.15%
FCM	300	243			0.8	95.75%
KM+SVM	72.6 (22.2+50.4)	**214**	2,753	45	3.08 (0.69+2.39)	**97.60%**
FCM+SVM	346.9 (300+46.9)	243	2,753	45	3.2 (0.8+3.2)	97.35%

Table 4. Experiment results for a subset of ETL8B database.

	A Subset of ETL8B (350 Character Types)					
	Training (28,000 Samples)				Testing (28,000 Samples)	
	Time (s)	# PRs	# SVs	# CPs	Time (s)	Acc
NN		28,000			109.5	98.16%
DAGSVM	32,046		1,666,126	61,075	1,111	**99.47%**
KM	2,184	**767**			**3.0**	97.81%
FCM	2,248	801			3.2	98.97%
KM+SVM	11,211 (2,184+9,027)	**767**	565,874	17,286	13.1 (3.0+10.1)	99.45%
FCM+SVM	11,087 (2,248+8,839)	801	**560,060**	**17,125**	13.3 (3.2+10.1)	99.46%

Table 5. Experiment results for a subset of ETL9B database.

	A Subset of ETL9B (350 Character Types)					
	Training (35,000 Samples)				Testing (35,000 Samples)	
	Time (s)	# PRs	# SVs	# CPs	Time (s)	Acc
NN		35,000			121.8	98.16%
DAGSVM	37,160		1,887,218	61,075	1,504	**99.05%**
KM	3,488	**977**			**3.4**	97.81%
FCM	5,328	1,042			3.6	98.36%
KM+SVM	16,029 (3,488+12,541)	**977**	689,776	20,228	16.7 (3.4+13.3)	98.94%
FCM+SVM	15,768 (5,328+10,440)	1,042	**677,090**	**19,856**	16.9 (3.6+13.3)	99.00%

Table 6. Experiment results for ETL8B (Full Set) database.

	ETL8B (956 Character Types)					
	Training (76,480 Samples)				Testing (76,480 Samples)	
	Time (s)	# PRs	# SVs	# CPs	Time (s)	Acc
NN		76,480			903	97.00%
DAGSVM	239,520		1.2×10^7	456,490	8,281	
KM	10,112	3,558			42	96.67%
FCM	10,963	**3,441**			**40.6**	97.03%
KM+SVM	44,155 (10,112+34,043)	3,558	1,946,922	64,880	70 (42+28)	98.45%
FCM+SVM	43,518 (10,963+32,555)	**3,441**	**1,914,215**	**63,794**	68.6 (40.6+28)	**98.51%**

Table 7. Experiment results for ETL9B (Full Set) database.

	ETL9B (3,036 Character Types)					
	Training (303,600 Samples)				Testing (303,600 Samples)	
	Time (s)	# PRs	# SVs	# CPs	Time (s)	Acc
NN		303,600			7,158	91.90%
DAGSVM	2,802,977		1.5×10^8	4,607,130	112,901	
KM	33,880	22,308			526	92.71%
FCM	19,685	**19,237**			**454**	94.08%
KM+SVM	245,421 (33,880+211,541)	22,308	11,317,700	347,702	650 (526+124)	96.07%
FCM+SVM	223,759 (19,685+204,074)	**19,237**	**11,104,041**	**341,138**	578 (454+124)	**96.59%**

6 Conclusion

To alleviate the computing costs of SVM in large-scale applications, we propose a hybrid method that combines SVM with a prototype classification method. Applying this method to handwritten characters has the effect of dramatically cutting down the training time, testing time, and the number of support vectors, as N increases. The results also show that the two hybrid classifiers maintain relatively the same accuracy rates as the DAGSVM classifier and achieves better performance than the NN classifier. Furthermore, the FCM classifier performs better than the KM classifier and the NN classifier.

References

[1] C. Cortes, and V. Vapnik, "Support-vector network," Machine Learning, vol. 20, pp. 273–297, 1995.

[2] J. C. Bezdek, Pattern recognition with fuzzy objective Function Algorithms. New York: Plenum, 1981.

[3] S. Knerr, L. Personnaz, and G. Dreyfus, "Single-layer learning revisited: A stepwise procedure for building and training a neural network," Neurocomputing: Algorithms, Architectures and Applications, J. Fogelman, Ed. New York: Springer-Verlag, 1990.

[4] J. C. Platt, N. Cristianini, and J. Shawe-Taylor, "Large margin DAG's for multiclass classification," Advances in Neural Information Processing Systems, Cambridge, MA: MIT Press, vol. 12, pp. 547–553, 2000.

[5] V. Vapnik, The nature of statistical learning theory, New York: Springer Verlag, 1995.

[6] M. Umeda, "Advances in recognition methods for handwritten Kanji characters," IEICE Trans. Information and Systems, vol. E79-D, no. 5, 1996.

[7] S.-W. Lee and J.-S. Park, "Nonlinear shape normalization methods for the recognition of large-set handwritten characters," Pattern Recognition, vol. 27, no. 7, pp. 895-902, 1994.

[8] H. Yamada, K. Yamamoto, and T. Saito, "A nonlinear normalization method for handprinted Kanji character recognition – line density equalization," Pattern Recognition, vol. 23, no. 9, pp. 1023-1029, 1990.

[9] C.-L. Liu, I.-J. Kim, and J.H. Kim, "High accuracy handwritten Chinese character recognition by improved feature matching method," 4th Intern. Conf. Document Analysis and Recognition, pp. 1033-1037, Ulm, Germany, 1997.

[10] H. Freeman, "Techniques for the digital computer analysis of chain-encoded arbitrary plane curves," Proc. Nat. Electronics Conf., pp. 421-432, 1961.

[11] T. D. Haig, and Y. Attikiouzel, "An improved algorithm for border following of binary images," IEE European Conference on Circuit Theory and Design, 118-122, 1989.

[12] F. Chang, C.-J. Chen, and C.-J. Lu, "A linear-time component labeling algorithm using contour tracing technique," Computer Vision and Image Understanding, vol. 93, no. 2, pp. 206-220, 2004.

[13] T. H. Hilderbrand, and W. Liu, "Optical recognition of Chinese characters: advance since 1980," Pattern Recognition, vol. 26, no. 2, pp. 205-225, 1993.

[14] P. Hart, "The condensed nearest neighbor rule," IEEE Trans. Information Theory, pp. 515-516, 1968.

[15] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," Technical Report IDIAP-RR 02-46, IDIAP, 2002.

[16] J. F. Kolen, and T. Hutcheson, "Reducing the time complexity of the fuzzy c-means algorithm," IEEE Trans. on Fuzzy Systems, vol. 10, no. 2, pp. 263-267, 2002.