A Hybrid Method for Multiclass Classification and Its Application to Handwritten Character Recognition

Fu Chang[†], *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*

E-mail: fchang@iis.sinica.edu.tw, erikson@iis.sinica.edu.tw, dean@iis.sinica.edu.tw

ABSTRACT

The support vector machine (SVM) is an effective pattern classification method. However, solving $N(N\text{-}1)/2$ binary classifications in the training phase makes it too costly to use SVM in applications with a high number N of class types. In this paper, we propose a new prototype classification method that can be combined with SVM for pattern recognition. This hybrid method has the following merits. First, the learning algorithm for constructing prototypes determines both the number and the location of the prototypes. This algorithm not only terminates within a finite number of iterations, but also assures that each training sample matches in class type with the nearest prototype. Second, SVM can be used to process the top-rank candidates obtained by the prototype classification method, which saves time in both the training and testing processes. We apply our method to recognizing handwritten numerals and handwritten Chinese/Hiragana characters. Experiment results show that the hybrid method saves a great deal of training and testing time in large scale tasks and achieves comparable accuracy to that achieved by using SVM solely. Our results also show that the hybrid method performs better than the nearest neighbour method.

Keywords: handwritten character recognition, multiclass classification, nearest neighbor, pattern matching, prototype, support vector machine

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 for $N > 2$. The *support vector machines* (SVM) method (Cortes and Vapnik [5], Vapnik [22]) is very effective for binary classification and can also be used for multiclass classification by decomposing the problem into binary classification sub-problems. The three approaches for decomposing the problem are: *one-against-all* (Bottou et al. [2]), *one-against-one* (Knerr et al. [14]), and directed acyclic graph SVM (DAGSVM) (Platt et al. [19]). In the one-against-all approach, there are N sub-problems, each of which classifies an object as A or not A. In both one-against-one and DAGSVM approaches, there are $N(N\text{-}1)/2$ sub-problems, each of which classifies an object as A or B, where A and B are any two classes.

Comparison of the results shows that the one-against-all requires more training time and more support vectors than the other two approaches (Hsu and Lin [12]). One-against-one and DAGSVM need to solve $N(N\text{-}1)/2$ binary sub-problems in the training phase. In the testing phase, one-against-one conducts $N(N\text{-}1)/2$ binary classification, each of which casts a vote for the class type assigned to the object. The class type that gains the highest number of votes is taken as the classification outcome. DAGSVM, on the other hand, conducts N-1 binary classifications with the help of a directed acyclic graph that has $N(N\text{-}1)/2$ nodes and N leaves. Both approaches require the same amount of memory space to store the support vectors that are the solutions to the $N(N\text{-}1)/2$ sub-problems in the training phase. In Chinese/Hiragana character recognition, the number of classes is at least 900 and can exceed 5,000 in some database. For this scale of pattern recognition, training time and memory space is prohibitively large in all of the three approaches.

For large-scale pattern recognition, a long-employed approach is the *nearest-neighbor* (NN) classification method (Dasarathy [7], Levine et al. [16], O'Callaghan [18]). The NN

method matches each object against all training samples and finds the nearest sample, or *k*-nearest samples, as the basis for classification. In theory, as the number of training samples approaches infinity, the classification error rate is at most twice the Bayes error rate (Cover and Hart [6]). The NN method takes no time in training and is usually faster than SVM in testing, due to the simplicity of computations involved in NN. However, matching each object against all training samples still lengthens the computational time. Also, the accuracy of NN method is not as good as that of SVM.

We propose a method that exploits the advantages of both NN and SVM and avoids their deficiencies. In the matching process, the set of all training samples is replaced by a much smaller set of *prototype*s. The innovation of our proposed method is a learning process that constructs prototypes from training samples. The proposed algorithm determines *both* the number and location of prototypes from the same set of training samples. The use of prototypes reduces the computational burdens of SVM in the training and testing phases, since SVM only needs to work on the top-rank candidates that are obtained in each phase.

We apply this hybrid method to the recognition of handwritten numerals and Chinese/Hiragana characters. There are three advantages to this method. First, the accuracy of using the hybrid method relatively equals SVM accuracy, and is better than NN accuracy. Second, the matching speed of the hybrid method is always faster than that of SVM. Third, if SVM is used solely for classification, decision functions must be created for all possible pairs of class types. In the hybrid method, however, a much smaller set of pairs can be determined from the prototype-learning process. The subsequent SVM process only needs to set up decision functions for these pairs, thereby tremendously reducing the training time and the memory space for storing support vectors.

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 Sec-

tion 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. An appendix is given at the end of this paper, containing a proof for the convergence of the major prototype-learning algorithm.

2. The Prototype-Construction Problem and Its Solutions

To set up the problem, we assume that a set of training samples is given and that the samples' class types are specified. Each sample is represented as a vector in n-dimensional Euclidean space. For two vectors $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ and $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ in this space, their distance is defined as

$$dist(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{n} (v_i - w_i)^2. \quad (1)$$

A prototype can be any n-dimensional vector whose class type is also specified. Let *type*(**x**) denote the class type of **x**, when **x** is either a sample or a prototype. A set Π of prototypes is said to be a *solution* to the problem of prototype construction if the following condition holds for every sample **s**. There exists a prototype **p** in Π such that *type*(**p**) = *type*(**s**) and *dist*(**s**, **p**) < *dist*(**s**, **q**) for all other **q** in Π.

If Π is a solution for prototype-construction, more than one prototype for the same class type may be found in Π (see Figure 1). If we examine the *attraction domain* of each prototype **p**, defined as the set of all samples for which **p** is the nearest prototype, we find that it contains samples of the same class types. For this reason, each **p** in Π can serve as the representative of its neighboring samples.

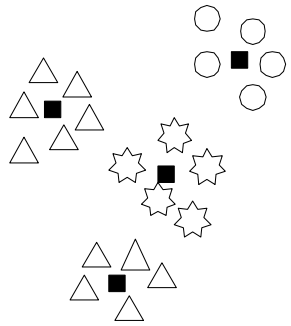


Figure 1. A solution to the prototype-construction problem. Each prototype is shown as a darkened square, and sample types are shown as different shapes.

We present two learning algorithms that solve the prototype-construction problem. One is called a *static algorithm* (SA) because prototypes will never be modified once they are constructed. The second is a *dynamic algorithm* (DA), which continues to modify the locations of existing prototypes, as well as constructing new ones.

The SA is given below.

(1) Initiation: We randomly pick a sample out of each class type as a prototype. The class type of this prototype is set the same as the selected sample.

(2) Absorption: For each sample **s**, we find the nearest prototype **p**. If *type*(**s**) = *type*(**p**) and *dist*(**s**, **p**) < *dist*(**s**, **q**) for all other prototypes **q**, then **s** is *absorbed*. Otherwise, it is *unabsorbed*.

(3) New prototype construction: If there are unabsorbed samples, we randomly pick an unabsorbed *C*-sample as a new *C*-prototype.

(4) Process cessation: If there are unabsorbed samples, go to step 2. Otherwise, we stop the whole process.

Note that the design of SA is very similar to the condensation nearest neighbor (CNN) rule proposed by Hart [9]. In CNN, each training sample is matched against all existing prototypes. When a training sample does not bear the same class type as the nearest prototype, it becomes a new prototype. After all samples have been examined, the process is repeated until no new prototypes are created. Our experiment results show that SA and CNN have

no new prototypes are created. Our experiment results show that SA and CNN have similar performance in terms of accuracy rates, as well as the number of prototypes constructed from training data.

The DA algorithm dynamically alters the number of prototypes, as well as their locations. DA constructs prototypes differently than SA. In SA, prototypes are selected out of samples, whereas in DA, they are statistical averages of the samples.

The DA is given below.

(1) Initiation: For each class type C, the initial C-prototype is the statistical average of all C-samples.

(2) Absorption: We determine each sample as absorbed or unabsorbed.

(3) New prototype construction: For each class type C, let the number of C-prototypes be $num(C)$. If there are unabsorbed C-samples, we construct $num(C)+1$ C-prototypes. Otherwise, C-prototypes remain the same as in the previous iteration.

(4) Process cessation: If there are unabsorbed samples, go to step 2. Otherwise, we stop the whole process.

In Step 3, the construction of new C-prototypes is as follows. First, we select a sample from the unabsorbed C-samples. We then use the selected sample and existing C-prototypes as initial *seeds* and employ the K-means clustering method to form new C-prototypes. To select an unabsorbed C-sample, we focus on a set Ψ_C consisting of unabsorbed C-samples that are *not* themselves C-prototypes. We let each sample in Ψ_C cast a vote to the nearest sample in Ψ_C. We then select the sample in Ψ_C that gains the highest number of votes. To construct new C-prototypes, we apply the K-means method to group all C-samples according to the following procedure. The K-means method assigns each sample to the cluster whose seed is nearest, and resets the seed as the geometric center of this cluster. This procedure continues

until all cluster seeds remain unchanged. The final cluster seeds are then assigned as new prototypes.

To address the convergence of the proposed algorithms, it is obvious that SA always converges. It is also true that DA terminates within a finite number of iterations, where the number of iterations is the number of times Step 3 is executed. This comes from the fact that the total sum of distances between samples and the nearest prototypes of the same class types decreases by at least a constant in each iteration. A sample thus remains unabsorbed for only a finite number of iterations. Details of the proof are given in the Appendix of this paper.

We now address the relationship between the K-means algorithm and the DA algorithm. The K-means algorithm has a useful property in that it continues to adjust cluster centers, so that the sum of the distances between samples and the nearest cluster centers decreases with each round of adjustment. DA takes advantage of this property to reduce the sum of distances between samples and the nearest prototypes of the same class types in each round of prototype construction. However, the K-means algorithm lacks guidelines for choosing the number of clusters, initial partitions, and the stopping criterion (Backer [1], Jain et al. [13]). The DA algorithm, on the other hand, supplies these missing components.

3. Disambiguation Using SVM

The prototype-matching method achieves very high accuracy rates for k nearest prototypes when $k > 1$, 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

using an SVM method. We use these schemes in the testing process to reassess the top-k 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 [22]). 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 for Handwritten Characters

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. According to Umeda [21], 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

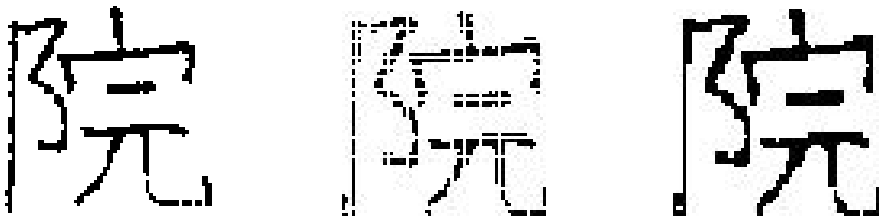
The shape normalization method (Lee and Park [15]) transforms a binary image of size $I \times J$ into a normalized image of size $M \times N$. The transformation is expressed as

$$m = \sum_{k=1}^{i} H(k) \times \frac{M}{\sum_{k=1}^{I} H(k)}, \qquad n = \sum_{k=1}^{j} V(k) \times \frac{N}{\sum_{k=1}^{J} V(k)}, \tag{2}$$

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. [23]) 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. [17], 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 (2).

A problem we have encountered with normalization is that the set $\{g(i, j)$: $i = 1, 2, \ldots, I$; $j = 1, 2, \ldots, J\}$ may be only a subset of the normalized image $\{(m, n)$: $m = 1, 2, \ldots, M$; $n = 1, 2, \ldots, N\}$, where $g(i, j)$ maps point (i, j) in the original image to a point in the normalized image. As shown in Figure 2b, certain points in the normalized image do not correspond to points in the original image (Figure 2a) and are set by default as white points.



(a) (b) (c)

Figure 2. (a) The original image. (b) The normalized image *before* interpolation is applied. (c) The normalized image *after* interpolation is applied.

To solve this problem, we use the following *interpolation* technique. When two neighboring points **P** and **Q** in the original image are mapped into two non-neighboring points **R** and **S**, we fill in the rectangle determined by **R** and **S** with black points (Figure 3). Figure 2c shows the normalized image after interpolation.

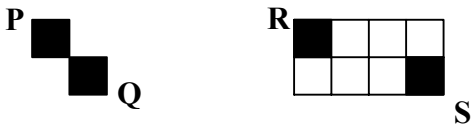


Figure 3. Left: Two neighboring points in the original image. Right: The corresponding points in the normalized image. The points in the rectangle are to be filled with black.

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 [8]) as shown below.

(1) Extract the character contour using a contour-tracing algorithm proposed by Haig and Attikiouzel [10] and modified by Chang and Chen [3].

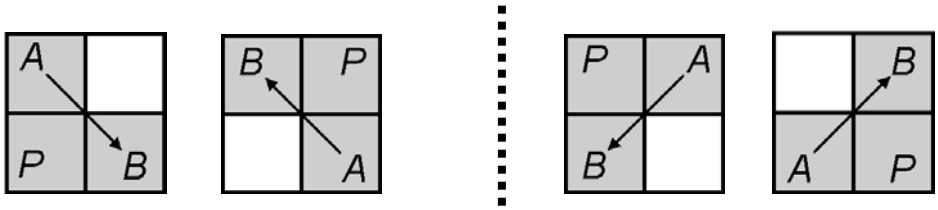
(2) To each contour point **A** and a neighboring contour point **B**, assign a directional attribute to **A** and **B** according to their relative positions. The assignment is shown in Figure 4. For example, if **B** lies to the right or left of **A**, both **A** and **B** are assigned attribute 1. 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 4. Directional attribute assignment. If **B** lies at the position labeled 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 5). **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**.

(4) Assign all remaining points directional attribute –1.



(a) (b)

Figure 5. 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. [17]. As shown in Figure 6, 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 6. 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

There are seven applications in our experiments (Table 1). 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 [22] and CENPARMI [20] handwritten numerals. The second group consists of two *middle-scale* classification tasks. In each task, there are 350 class types of handwritten Chinese/Hiragana characters taken from ETL8B and ETL9B databases [11]. Each class type contains the same number of samples as the original database. These two tasks are referred to as ETL8B (subset) and ETL9B (subset). The third group consists of two *large-scale* classification tasks, in which we use full ETL8B and ETL9B sets. They are referred to as ETL8B (full set) and ETL9B (full set).

Table 1. The seven applications.

	# Class Types	# Training Samples	# Test Samples
UPS	10	7,091	2,007
CENPARMI (Density)	10	4,000	2,000
CENPARMI (Direction)	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

For all databases, except UPS and CENPARMI, we use the feature extraction method consisting of nonlinear normalization, directional feature extraction and blurring described in Section 4.1–4.3. For UPS, because all images are 16×16 in scale, we simply take each number as a component to form a 256-dimensional vector. For CENPARMI, two feature extraction methods are used. One is the same as we use for all other databases. It is referred to as

'Direction' in Table 1. The other method is the density feature extraction technique, as described in Section 4.4. This technique is referred to as 'Density' in Table 1. We apply two feature extraction methods to the same data set to show that the relative standings of classification methods are little affected by different feature extraction methods. In Table 1, we list the number of class types, the number of training samples and the number of test samples in each application.

In all applications, we compare our hybrid method to 1-NN (which uses all training samples as prototypes and finds the nearest one to each test sample), CNN, SA and DA. Since DA outperforms CNN and SA in both the number of prototypes and accuracy rates, we take the combination of the DA and SVM to form our hybrid solution. The confusing pairs in the hybrid method are formed and used in the following way. In the training phase, we obtain confusing pairs (C_i, C_j), where C_i and C_j are rank-i and rank-j candidates of a sample for $1 \leq i$, $j \leq 5$. In the testing phase, we only deal with confusing pairs that appear in the top-3 candidates of each sample. We use the polynomial kernel of degree two for the choice of kernel function. The software package, Torch [4], is used to conduct all SVM experiments.

In small-scale and middle-scale tasks, using SVM solely for classification is feasible. We apply both the one-against-one and DAGSVM approaches to these data sets. For large-scale classification, SVM 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 Tables 2 through 8, the first column lists all of the classification methods to be compared. The second column lists the accuracy rates of these methods that are applied to test data, and the third column lists the number of prototypes used in prototype-matching methods, or the number of confusing pairs used in SVM and hybrid methods. When one-against-one and DAGSVM approaches are used, the term 'confusing pairs' refers to the number of binary classification problems that must be solved in the training phase. The results in Tables 2 through 8 show

that the hybrid method achieves comparable accuracy rates to the two SVM approaches and achieves better accuracy rates than 1-NN, CNN and SA.

Table 2. UPS data.

Classification Methods	Accuracy Rates	Number	
1-NN	94.47%	Prototypes:	7,291
CNN	91.53%	Prototypes:	836
SA	92.13%	Prototypes:	854
DA	92.37%	Prototypes:	393
One-against-one SVM	95.12%	Confusing Pairs:	45
DAGSVM	95.47%	Confusing Pairs:	45
Hybrid (DA + SVM)	**95.22**%	Confusing Pairs:	45

Table 3. CENPARMI data, using density feature extraction technique.

Classification Methods	Accuracy Rates	Number	
1-NN	92.90%	Prototypes:	4,000
CNN	88.65%	Prototypes:	781
SA	89.30%	Prototypes:	794
DA	92.45%	Prototypes:	335
One-against-one SVM	95.60%	Confusing Pairs:	45
DAGSVM	95.80%	Confusing Pairs:	45
Hybrid (DA + SVM)	**95.95**%	Confusing pairs:	45

Table 4. CENPARMI data, using directional feature extraction technique.

Classification Methods	Accuracy Rates	Number	
1-NN	96.45%	Prototypes:	4,000
CNN	92.55%	Prototypes:	513
SA	92.00%	Prototypes:	486
DA	95.15%	Prototypes:	214
One-against-one SVM	97.10%	Confusing Pairs:	45
DAGSVM	97.35%	Confusing Pairs:	45
Hybrid (DA + SVM)	**97.60**%	Confusing Pairs:	45

Table 5. ETL8B (Subset) data.

Classification Methods	Accuracy Rates	Numbers	
1-NN	98.16%	Prototypes:	28,000
CNN	94.59%	Prototypes:	2,504
SA	94.87%	Prototypes:	2,638
DA	97.81%	Prototypes:	767
One-against-one SVM	**99.47**%	Confusing Pairs:	61,075
DAGSVM	**99.47**%	Confusing Pairs:	61,075
Hybrid (DA + SVM)	99.45%	Confusing Pairs:	17,286

Table 6. ETL9B (Subset) data.

Classification Methods	Accuracy Rates	Number	
1-NN	98.16%	Prototypes:	35,000
CNN	94.59%	Prototypes:	4,119
SA	94.87%	Prototypes:	4,120
DA	97.81%	Prototypes:	977
One-against-one SVM	**99.05**%	Confusing Pair:	61,075
DAGSVM	**99.05**%	Confusing Pairs:	61,075
Hybrid (DA + SVM)	98.94%	Confusing Pairs:	20,228

Table 7. ETL8B (Full Set) data.

Classification Methods	Accuracy Rates	Number	
1-NN	97.00%	Prototypes:	76,480
CNN	93.30%	Prototypes:	11,536
SA	93.24%	Prototypes:	11,397
DA	96.67%	Prototypes:	3,558
Hybrid (DA + SVM)	**98.45**%	Confusing pairs:	64,880

Table 8. ETL9B (Full Set) data.

Classification Methods	Accurate Rates	Number	
1-NN	91.90%	Prototypes:	30,3600
CNN	85.79%	Prototypes:	80,569
SA	86.02%	Prototypes:	78,398
DA	92.71%	Prototypes:	22,308
Hybrid (DA + SVM)	**96.07**%	Confusing pairs:	347,702

In Table 9, we list the total training and testing time (in seconds) of the two SVM approaches and the hybrid method. The number of support vectors (# SVs) employed in each method is also listed. Since the two SVM approaches employ the same training process, the training results are only listed once. For the full ETL8B and ETL9B sets, SVM training times would be too long. Their times are extrapolated from those obtained from the subset data. The hybrid training consists of two parts: prototype and SVM training. In prototype training, the following speed-up technique is found useful. To determine whether each sample **s** is absorbed or not, we only match **s** against the prototypes whose class types fall in $\Omega_k(\mathbf{s})$. We predetermine $\Omega_k(\mathbf{s})$ for each **s** as follows. For any class type D, let

$$d_{\mathbf{s}}(D) = \min_{\mathbf{x}} dist(\mathbf{s}, \mathbf{x}),$$

where **x** ranges over all D-samples. We then rank all class types according to the values of $d_{\mathbf{s}}(\cdot)$. $\Omega_k(\mathbf{s})$ is then the set of all class types whose ranks fall below k. In all Chinese/Hiragana applications, we set k at 50. For a large value of k, the probability that the nearest prototype to **s** assumes a class type in $\Omega_k(\mathbf{s})$ is extremely high. This justifies the use of the speed-up method.

Table 9. Training time, testing time, and number of support vectors.

	One-against-one SVM			**DAGSVM**	**Hybrid**		
	Training		Test	Test	Training		Test
	Time	# SVs	Time	Time	Time (DA+SVM)	# SVs	Time (DA+SVM)
UPS	**139**	7,233	51	9.96	261 (122+139)	7,233	**3.92** (3.12+0.797)
CENPARMI (Density)	**46.9**	4,950	27.46	8.65	96.7 (49.8+46.9)	4,950	**4.13** (0.984+3.15)
CENPARMI (Direction)	**50.4**	2,753	19.56	5.96	72.6 (22.2+50.4)	2,753	**3.08** (0.687+2.39)
ETL8B (Subset)	32,046	1,666,126	470,277	1,111	**11,211** (2,184+9,027)	**565,874**	**13.1** (3+10.1)
ETL9B (Subset)	37,160	1,887,218	500,815	1,504	**16,029** (3,488+12,541)	**689,776**	**16.8** (3.4+13.3)
ETL8B (Full Set)	239,520	1.2×10^7	2.6×10^8	8,281	**44,155** (10,112+34,043)	**1,946,922**	**70** (42+28)
ETL9B (Full Set)	2,802,977	1.5×10^8	2.8×10^{10}	112,901	**245,421** (33,880+211,541)	**11,317,700**	**650** (526+124)

Table 9 shows that, compared to the two SVM approaches, the hybrid method requires a shorter testing time and a smaller, or equal, number of support vectors. The difference greatly increases when N is large, as shown in all Chinese/Hiragana results. The hybrid method has a much shorter training time than SVM when N is large. For the UPS and CENPARMI data, the hybrid method requires the same number of confusing pairs as SVM, thus incurring a slightly longer training time than SVM. Figure 7 shows the respective training times (in seconds) required by SVM and the hybrid method for data sets of various sizes. The data sets are derived from ETL9B and the number of character types in each set ranges from 50, 100, 150, …, up to 1000. Each character type, in any set, consists of 100 samples.

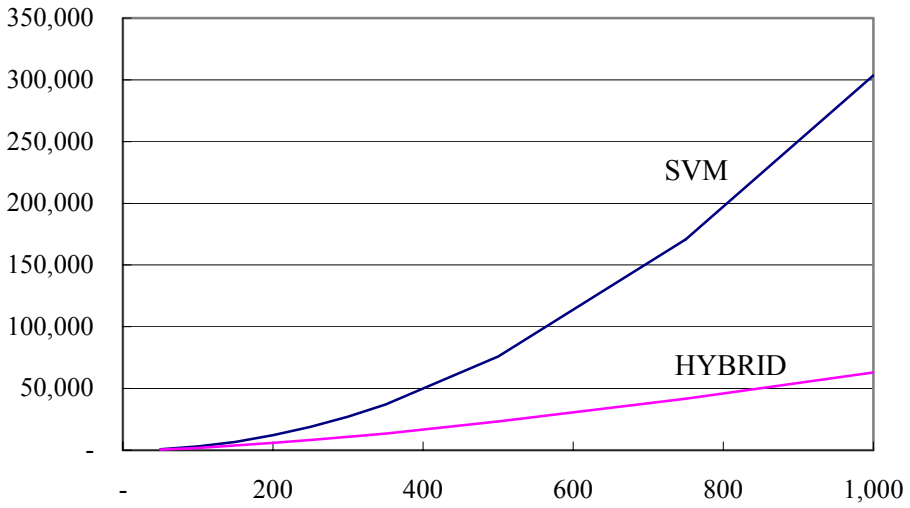


Figure 7. Training times of SVM and the hybrid method. The numbers of character types are plotted on the horizontal axis and the training times are plotted on the vertical axis.

6. Conclusion

To remedy the costly computation of SVM when N is large, we propose a hybrid solution that combines SVM with a prototype learning/matching method. The prototype learning method determines both the number and the location of prototypes. This is done in a learning process that constructs prototypes within a finite number of iterations by way of training samples, and assures that each sample matches in class type with the nearest prototype. This learning process determines confusing pairs that are processed by the SVM method. Applying the hybrid method to handwritten characters dramatically reduces training time, testing time, and the number of support vectors, as N increases. The reduction in training time and support vectors is due to the fact that the hybrid method deals only with a small proportion of binary classification problems. The reduction in testing time is due to the fact that prototype matching is faster than SVM classification. Our experiment results also show that the hybrid method maintains relatively the same accuracy rates as SVM. These results naturally prove that the hybrid method is a successful character recognition solution for all N scales.

References

[1] E. Backer, *Computer-Assisted Reasoning in Cluster Analysis*, New Jersey: Prentice Hall, 1995.

[2] L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, U. Muller, E. Sackinger, P. Simard, and V. Vapnik, Comparison of classifier methods: A case study in handwriting digit recognition, *Int. Conf. Pattern Recognition*, pp. 77–87, 1994.

[3] 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.

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

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

[6] T. Cover and P. Hart, Nearest neighbor pattern classification, *Proc. IEEE Trans. Information Theory*, IT-11, pp. 21-27, 1967.

[7] B. V. Dasarathy, NN concepts and techniques, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, B.V. Dasarathy (Ed.), IEEE Computer Society Press, pp. 1-30, 1991.

[8] H. Freeman, Techniques for the Digital Computer Analysis of Chain-Encoded Arbitrary Plane Curves, *Proc. Nat. Electronics Conf.*, 1961, pp. 421-432.

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

[10] T. D. Haig and Y. Attikiouzel, An Improved Algorithm for Border Following of Binary Images, *IEE European Conference on Circuit Theory and Design*, 1989, 118-122.

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

[12] C.-W. Hsu and C.-J. Lin, A comparison of methods for multiclass support vector machines, *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415-425, 2002.

[13] Anil K. Jain and R. P. W. Duin, and J Mao, Statistical pattern recognition: a review, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4-37, 2000.

[14] 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.

[15] 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.

[16] A. Levine, L. Lustick, and B. Saltzberg, The nearest neighbor rule for small samples drawn from uniform distributions, *IEEE Trans. Information Theory*, vol. IT-19, no. 5., pp. 697-699.

[17] 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.

[18] J. F. O'Callaghan, An alternative definition for neighborhood of a point, *IEEE Trans. Computers*, vol. C-24, no. 11, pp. 1121-1125.

[19] 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, 2000, vol. 12, pp. 547–553.

[20] C. Y. Suen, C. Nadal, R. Legault, T. A. Mai, L. Lam, Computer recognition of unconstrained handwritten numerals, *Proceedings of the IEEE*, vol. 80, no. 7, pp.1162-1180, 1992.

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

[22] V. Vapnik, *The Nature of Statistical Learning Theory*, New York: Springer Verlag, 1995.

[23] 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.

Appendix: The Convergence of The DA Algorithm

We want to prove that the DA algorithm terminates within a finite number of iterations, where the number iterations is the number of times Step 3 of DA is executed. The process in Step 3 is referred to here as the *prototype-construction* process, or in abbreviation, PCP. We want to show that, as prototypes are updated in PCP, all samples are eventually absorbed by prototypes of the same class types. This results from the fact that if we measure the distances between the samples and the nearest prototypes of the same class types, we find that the sum of these distances decreases by at least a constant in each iteration.

Recall that PCP performs two steps for each class type C. In the first step, a sample $\mathbf{s}_C$ is selected out of the set of unabsorbed C-samples that are *not* themselves C-prototypes. In the second step, if $\mathbf{s}_C$ cannot be found, no further work is done for class type C. Otherwise, the K-means clustering method is used to form new C-prototypes using $\mathbf{s}_C$ and all old C-prototypes as initial seeds. We define $SD(C)$ to be the sum of distances between the C-samples and the nearest C-prototypes, namely,

$$SD(C) = \sum_{\mathbf{x}} dist(\mathbf{x}, \mathbf{p}),$$

where **x** ranges over all *C*-samples and **p** is a C-prototype for which $dist(\mathbf{x}, \mathbf{p}) \leq dist(\mathbf{x}, \mathbf{q})$ for all other *C*-prototypes **q**. Let $SD_{\text{before}}(C)$ be the sum of distances *before* PCP is applied and $SD_{\text{after}}(C)$ be the sum of distances *after* it is applied.

Lemma 1. If a new seed $\mathbf{s}_C$ is found for class type *C*, then PCP decreases the value of $SD(C)$, namely, $SD_{\text{after}}(C) < SD_{\text{before}}(C)$.

Proof. In PCP, prototypes are constructed by means of the K-means clustering method. The K-means method produces a sequence of sets $\Pi^{(0)}$, $\Pi^{(1)}$, etc. $\Pi^{(0)}$ is the set of prototypes obtained in the previous iteration of DA; $\Pi^{(1)} = \Pi^{(0)} \cup \{\mathbf{s}_C\}$; $\Pi^{(n+1)}$ is the set of cluster centers obtained using members of $\Pi^{(n)}$ as seeds for $n = 1, 2, \ldots$. Let

$$SD^{(i)}(C) = \sum_{\mathbf{x}} dist(\mathbf{x}, \mathbf{p}^{(i)}(\mathbf{x})),$$

where **x** ranges over all *C*-samples, $\mathbf{p}^{(i)}(\mathbf{x})$ is the nearest member of $\Pi^{(i)}$ to **x** for $i = 0, 1, 2, \ldots$. We want to prove that $SD^{(i)}(C)$ is a strictly decreasing sequence.

$SD^{(1)}(C) \leq SD^{(0)}(C)$, since the nearest member of $\Pi^{(1)}$ to each *C*-sample is either a member of $\Pi^{(0)}$ or $\mathbf{s}_C$. One of the *C*-samples is $\mathbf{s}_C$ whose distance to the nearest member of $\Pi^{(0)}$ is non-zero, since $\mathbf{s}_C$ is not itself a *C*-prototype. The distance of $\mathbf{s}_C$ to the nearest member of $\Pi^{(1)}$ is zero, since $\mathbf{s}_C$ is a member of $\Pi^{(1)}$. Thus, the decrement of $SD^{(1)}(C)$ from $SD^{(0)}(C)$ is at least $dist(\mathbf{s}_C, \mathbf{p}^{(0)}(\mathbf{s}_C))$.

When the process proceeds from $\Pi^{(i)}$ to $\Pi^{(i+1)}$, new clusters are formed using members of $\Pi^{(i)}$ as seeds. The members of $\Pi^{(i+1)}$ are the centers of these clusters. Thus,

$$\sum_{\mathbf{x}} dist(\mathbf{x}, \mathbf{p}^{(i)}(\mathbf{x})) = \sum_{\mathbf{x}} dist(\mathbf{x}, \mathbf{p}^{(i+1)}(\mathbf{x})) + \sum_{\mathbf{x}} dist(\mathbf{p}^{(i)}(\mathbf{x}), \mathbf{p}^{(i+1)}(\mathbf{x})).$$

It follows that

$$\sum_{\mathbf{x}} dist(\mathbf{x}, \mathbf{p}^{(i+1)}(\mathbf{x})) < \sum_{\mathbf{x}} dist(\mathbf{x}, \mathbf{p}^{(i)}(\mathbf{x})).$$

We conclude that $SD^{(i+1)}(C) < SD^{(i)}(C)$. ∎

Recall that in PCP, the seeds are chosen from unabsorbed samples that are not identical to any prototypes of the same class types. We refer to these samples as *unabsorbed non-prototype samples*.

Lemma 2. When there is an unabsorbed sample, there is an unabsorbed non-prototype sample, which is not necessarily of the same class type.

Proof. We assume that **s** is an unabsorbed *C*-sample. This means that **s** falls within the attraction domain of a prototype, say **p**, whose class type is *not C*. Let the class type of **p** be *D*.

Case1. Sample **s** is not a *C*-prototype.

Then, **s** is an unabsorbed non-prototype sample.

Case 2. Sample **s** is a *C*-prototype.

Since **s** is an unabsorbed sample, the distance from **s** to **p** cannot exceed the distance from **s** to any *C*-prototype, including **s** itself. So, $dist(\mathbf{s}, \mathbf{p}) \leq dist(\mathbf{s}, \mathbf{s}) = 0$, implying that $\mathbf{s} = \mathbf{p}$. Since each prototype is the center of a cluster, let the *C*-cluster centered at **s** be $\Lambda_{\mathbf{s}}$ and the *D*-cluster centered at **p** be $\Lambda_{\mathbf{p}}$. We first prove that $\Lambda_{\mathbf{p}}$ contains a *D*-sample other than **p**. Assume, on the contrary, that $\Lambda_{\mathbf{p}}$ has only one member, namely, **p**. Then **p** is a *D*-sample. Since $\mathbf{s} = \mathbf{p}$, **s** inherits the class type of **p** which is not *C*. This is an absurd result. So, let **t** be a member of $\Lambda_{\mathbf{p}}$ other than **p** (Figure 8). Since $\mathbf{s} = \mathbf{p}$, $dist(\mathbf{t}, \mathbf{p}) = dist(\mathbf{t}, \mathbf{s})$, implying that **t** is an unabsorbed *D*-sample. Moreover, since **t** is a member of $\Lambda_{\mathbf{p}}$ and $\mathbf{t} \neq \mathbf{p}$, **t** is an unabsorbed non-prototype sample. ∎

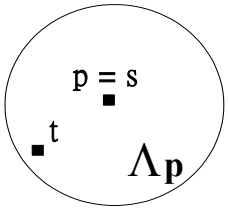


Figure 8. $\Lambda_{\mathbf{p}}$ has a member **t** and $\mathbf{t} \neq \mathbf{p}$.

We now define *TSD* to be the sum of the distances between the samples and the nearest prototypes of the same class types, namely,

$$TSD = \sum_C SD(C) .$$

Lemma 3. In each iteration of DA, PCP decreases *TSD* by at least a positive constant.

Proof. The onset of a new iteration in DA implies that there is at least one unabsorbed sample. Lemma 2 assures that there is also at least one unabsorbed non-prototype sample. PCP takes some unabsorbed non-prototype samples as seeds. Let **s** be such a sample whose class type is *C*. According the argument developed in the proof of Lemma 1, *SD*(*C*) decreases by at least *dist*(**s**, **q**), where **q** is the nearest *C*-prototype to **s** and $\mathbf{q} \neq \mathbf{s}$.

Each *C*-prototype is the center of a *C*-cluster. Let $\Sigma(\mathbf{s})$ be the collection of all subsets of *C*-samples whose center is *not* identical to **s**. $\Sigma(\mathbf{s})$ is non-empty, since the cluster containing **q** is a member of $\Sigma(\mathbf{s})$. Moreover, $dist(\mathbf{s}, \mathbf{q}) \geq \min_{\sigma \in \Sigma(\mathbf{s})} dist(\mathbf{s}, center(\sigma))$, where $center(\sigma)$ is the center of σ. From this, it follows that *TSD* decreases by at least $\delta = \min_{\mathbf{s}} \min_{\sigma \in \Sigma(\mathbf{s})} dist(\mathbf{s}, center(\sigma))$, where **s** ranges over all possible samples. Obviously, δ is a positive constant. ∎

THEOREM. The DA algorithm terminates within a finite number of iterations.

Proof. Assume, on the contrary, that DA never terminates. Consequently, there would always be unabsorbed samples, no matter how many iterations were performed. In each iteration, *TSD* decreases by at least a positive constant, so *TSD* decreases to zero within a finite number of iterations. This means that all samples become prototypes and are thus absorbed, contradicting the assumption we made. ∎