Data Reduction Methods for Pattern Recognition

Fu Chang[†], Chin-Chin Lin[†‡] and Wen-Hsiung Lin[†]
[†]Institute of Information Science, Academia Sinica, Taipei, Taiwan
[‡]Dept. of Electrical Engineering, National Taipei University of Technology, Taipei, Taiwan

ABSTRACT

In this paper we present a series of data-reduction methods for classifying an unknown object as one member of a large set of possible patterns. The first reduction method is a learning algorithm that reduces a gigantic set of training samples into a condensed set of templates, each represented in vector form. When used in a testing process, these templates hold target patterns within the nearest *K* templates for almost all unknown objects, where *K* is a small number. The second reduction method exploits the nature of templates and classification trees to form a fast tree-retrieval mechanism. Experiment results show that this retrieval mechanism is a lot more effective than the K-means clustering method in meeting the same objective. The third method is a disambiguation method that supplies a condensed set of confusing pairs. This method exploits an effective binary classification technique to re-evaluate all the confusing pairs that appear in the nearest *K* templates for each unknown object, and thus improves the accuracy rate of the final classification decision.

I. Introduction

Nearest-neighbor methodology [4, 5, 7], that matches each unknown object with all training samples and picks out the nearest or K-nearest training samples as basis for classification, can be justified in both theory and applications. In theory, it has been proved that, as the number of training samples approaches infinity, the classification error rate based on the nearest sample is at most twice the Bayes error rate [4]. In applications, it has been shown in a now-well-known example that this technique outstrips the performance of support vector machines for classifying handwritten numerals [12, 16].

A drawback of nearest neighbor method is its computing cost. Matching an un-

known vector with all possible training vectors requires a tremendous amount of computation. For example, in an application we have dealt with, there are approximately 400,000 samples, each represented as a 256-dimensional vector. The sheer vector-distance computation renders this method practically infeasible in such a large-scale task.

In this paper, we propose a series of reduction methods that condenses computations from one process to another so that in the end we achieve a very high accuracy rate for pattern classification in a computationally effective fashion that allows us to make approximately 600 classification decisions per second when classifying a 256-dimensional vector as one of 5,973 possible classes.

We propose three data reduction methods that are sequentially applied for solving pattern recognition problems. The first method is a *template construction* method, whose purpose is to transform a training data set into a much more condensed set of standard models or templates. Templates do not have to be training samples but must reside in the same vector space as training samples. The number of templates can exceed, but never drop below, the number of possible class types. Keeping a small size of templates is an important objective in the template construction process. Another important objective is to hold, within K-nearest templates of each unknown object, a template that bears the same class type as the unknown object, whereas K is a small number (2 or 3 in our applications).

The number of templates is proportionally small compared to that of training samples (6~7% in our applications). However, they are still too many to match with each unknown object in a one-to-one fashion. The *fast template-retrieval* method, our second method, is thus required. This method combines templates with classification trees to form a tree-retrieval mechanism. This mechanism stores templates in certain binary trees. Since templates are “representatives” of certain training samples, the leaf

that stores a template T is deemed to be associated with the leaves that store the samples represented by T. To retrieve templates of an unknown object from a binary tree, we not only retrieve the templates stored in one leaf, but also the templates stored in its associated leaves. It turns out that we can obtain extremely small amount of templates (less than 0.5% of all templates in our applications) through this retrieval mechanism at near-zero loss.

Our third method is a *disambiguation* method used for resolving those pairs of class types that are easily confused with each other. This process works on a very small amount of confusing pairs (only 0.2% of all possible pairs in our applications) derived as byproducts of the template construction process. To differentiate between confusing pair requires an effective binary classification technique. For this purpose, we employ support vector machines to compute the optimal supporting hyperplane for the two groups of training samples that correspond to a given confusing pair.

This paper is organized as follows. Section 2 contains the formulation for template construction problem, proposed learning algorithms, application tasks, and training and testing results. In Section 3, we describe the storage device as well as training method for fast template retrieval, and also the testing and comparison results. Section 4 outlines the disambiguation process, its training method, and all related results. The summary of this article is given in Section 5.

2. Template Construction

2.1 The Problem

Templates are derived from training samples. We assume that a set of samples is given and that their class types are also specified. Each sample is represented as a vector lying in the n-dimensional space. For any 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

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

For any collection T of templates and $\mathbf{t} \in T$, the *attraction domain* DOM($\mathbf{t}$) of $\mathbf{t}$ is the collection of all training samples for which $\mathbf{t}$ is the nearest template, namely,

$$\text{DOM}(\mathbf{t}) = \{\mathbf{s} : \ \mathbf{s} \text{ is a sample and } \text{dist}(\mathbf{s}, \mathbf{t}) < \text{dist}(\mathbf{s}, \mathbf{u}) \ \text{ for all } \mathbf{u} \in T \text{ and } \mathbf{t} \neq \mathbf{u}\}$$

For a set of templates to serve as a *solution* for the template construction problem, we require that each sample belongs to a template's attraction domain and, moreover, that each attraction domain is *homogeneous*, namely, that it contains samples of the same class type (Figure 1). This condition reflects the intuitive requirement that each template serves the representative of its neighboring samples.

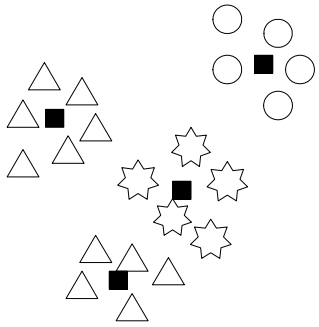

Figure 1. A set of templates. Each template is shown by a darkened square, while training samples of different types are shown as different shapes.

2.2 Template Construction Algorithms

We present two methods for template construction here. The first method is adopted from RCE algorithm [10, 11]. It is stated as follows.

(1) Initiation: we randomly pick a sample out of each class type as a template. The type of each template is set to be the same as the selected sample.

(2) Absorption: for each sample **s**, find the nearest template **t** to **s**. If **s** bears the same type as **t**, then **s** is said to be *absorbed*. Otherwise, it is *unabsorbed*.

(3) Building new templates: if there are still unabsorbed samples, we randomly pick an unabsorbed type-C sample as a new type-C template.

(4) Stopping criterion: if there are still unabsorbed samples, go to step 2. Otherwise, we stop the whole process.

When the process is terminated, each attraction domain becomes homogeneous. Moreover, each sample belongs to the attraction domain of a certain template. In this method, once templates are constructed, they are unchanged. For this reason, this method is a *static* construction method.

The second method dynamically alters the number of templates as well as their locations. For this reason, we call it a *dynamic* construction method.

(A) Initiation: for each class type C, the number of templates K(C) is set to be 1.

(B) Determining templates: if K(C) = 1, the only type-C template is set to be the statistical average of all type-C samples. If K(C) > 1, we apply the K-means clustering method to all type-C samples to form K(C) clusters, using randomly picked K(C) samples as seeds. It is possible that certain clusters become empty at the end of applying the K-means clustering method. In this case, we keep only non-empty clusters. The centers of these clusters are assigned as type-C templates.

(C) Determining the number of templates: for each sample **s**, find the nearest template **t** to **s**. If **s** bears the same type as **t**, then **s** is said to be *absorbed*. Otherwise, it is *unabsorbed*. For each class type C, if there are still unabsorbed samples, we increase K(C) by 1.

(D) Stopping criterion: same as in the static construction method.

The K-means clustering method, which is employed in step B of the dynamic construction method, groups samples according to the following procedure. To start with, it uses given seeds as initial cluster centers. It then assigns each sample to the cluster whose center is nearest. Once a new member joins a cluster, the cluster center is reset to be the statistical average of all the members in the cluster [5-7].

Both static and dynamic construction methods adopt the same stopping criterion to ensure that all samples are absorbed. In practice, we allow the construction process to stop as long as the number of absorbed samples decreases from one cycle to the next. When this occurs, we restore the set of templates obtained in the previous cycle. Prompt cessation of the template construction process avoids time wasted building ineffective templates.

2.3 Testing Results

The above two template construction algorithms are employed while building recognizers for multi-font printed characters. In this task, we collected 390,823 traditional Chinese (TC) character images, and 360,443 simplified Chinese (SC) images. These images are from two major sources: characters generated by computers, and articles in newspapers, magazines, or books.

There are 5,973 class types in TC and 6,767 class types in SC. Each character image is normalized to a bitmap of 64×64 pixels and represented as a vector. Each component of the vector takes the number of black pixels found within a 4×4 cell as its value. Since there are 256 (=16×16) non-overlapping cells within a 64×64 bitmap, the dimension of each vector is 256 (Figure 2). For alternative character features and recognition methods, readers are referred to [7, 14, 15].

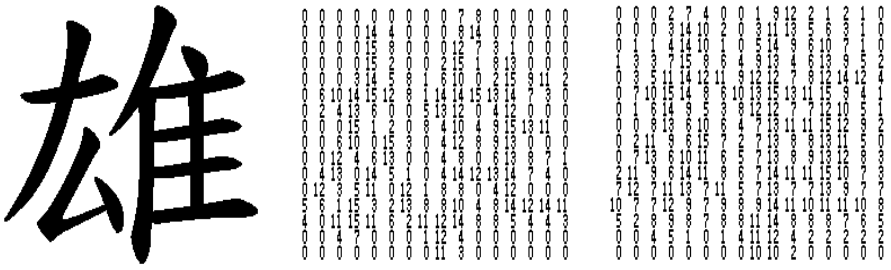

Figure 2. Left panel: the 64×64 bitmap of a character image. Middle panel: the 16×16 vector representation of the image. Right panel: the nearest template.

The training and testing results for the two template construction methods are

listed in Table 1. In both applications, the static construction method produces a lot more templates and yet maintains lower top-1 accuracy rates on testing data than the dynamic construction method, and demonstrates the advantage of the dynamic construction method.

Table 1. Training and testing results of static and dynamic construction methods for traditional Chinese (TC) and simplified Chinese (SC) applications.

Applications	TC		SC	
Template Construction Methods	Static	Dynamic	Static	Dynamic
Number of Templates	33,543	21,595	35,084	24,145
Accuracy Rate (Top-1)	98.12%	98.60%	98.53%	99.28%

Further performance results of the dynamic construction method in two applications are listed in Table 2. The results indicate that the large volume of training samples are condensed into a much smaller number of templates (at a reduction rate of 5.5% in the TC application and 6.7% in the SC application), while the rate of holding target patterns to the nearest three (top-3) templates is 99.80% for TC and 99.78% for SC. Note, however, that there is a non-negligible gap between the top-1 and top-3 accuracy rates in both applications. This is taken care in the disambiguation process, to be described in Section 4.

Table 2. Performance results of dynamic construction method in both applications.

Applications	TC	SC
Number of Training Samples (S)	390,823	360,443
Number of Templates (T)	21,595	24,145
Reduction Rate (T/S)	5.5%	6.7%
Number of Testing Samples	48,910	27,326
Accuracy Rate (top-1)	98.60%	99.28%
Accuracy Rate (top-2)	99.65%	99.72%
Accuracy Rate (top-3)	99.80%	99.78%

3. Fast Template Retrieval

3.1 Tree-Retrieval Technique

A number of hierarchical decision mechanisms have been proposed [1, 8, 9, 13, 17]. In this paper, we propose a fast template-retrieval method that employs *template trees* (Figure 3). Template trees bear the same structure as classification trees (Breiman, Friedman, Olshen, and Stone [1]). Each tree is associated with a type of vector $(x_1, x_2, \ldots, x_n)$. At the root level, the first component of each input vector is examined and assigned to one of two partitions. Extended from the root are two branches, each of which leads to a sub-tree. Each sub-tree has the same structure as a full tree. The leaves (i.e., nodes that have no branches) store templates to be retrieved for each input. We require a training process to determine the branch point at each node, the templates stored in the leaves, and a certain parameter that determines the templates to be retrieved.

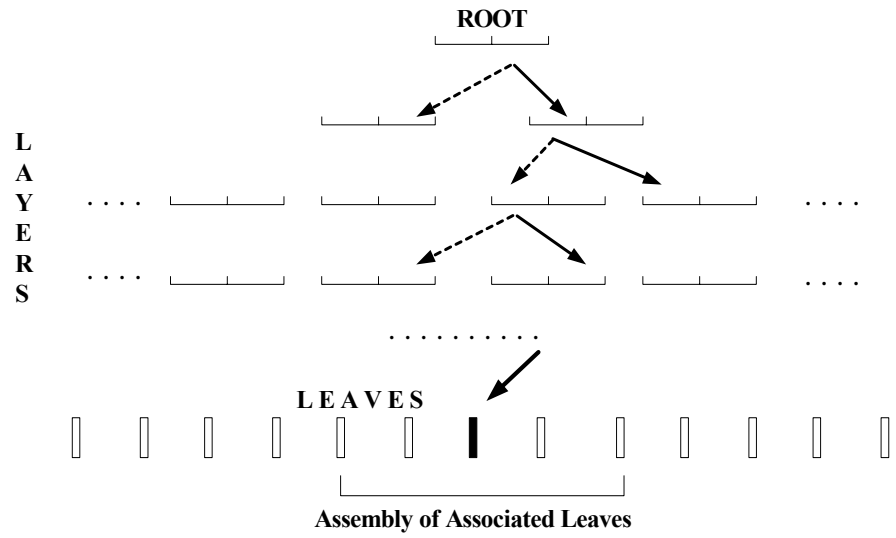

Figure 3. A template tree structure. Solid arrows indicate where the input actually migrates and dashed arrows where it can possibly migrate.

To retrieve templates from a single tree results in a large template set. *Multiple trees*, on the other hand, help to reduce the number of retrieved templates. So, instead of feeding a gigantic vector into a single tree, we subdivide the vector into several

sub-vectors and feed each into its corresponding tree. For convenience, we still refer to each sub-vector as a vector.

3.2 Growing Template Trees

Each template vector **v** enters a tree through its root. At the root, **v** is assigned to one of two partitions according to the first component v_1 of **v**. If v_1 falls below the *branch point*, this vector migrates to the left branch. Otherwise, it migrates to the right branch. The rest of the operations obey the same rule.

Following [1], we set the branch point for each node N to be the value s that maximizes $\Delta \mathrm{I}_N(t)$, namely,

$$s = \arg\max_t \Delta \mathrm{I}_N(t).$$

The value $\Delta \mathrm{I}_N(t)$ measures the *decrease in impurity* at node N when the branch point of N is set at t. Since templates passing N migrate to two braches, with a proportion p_{L} moving into the left branch $\mathrm{t_L}$ and a proportion p_{R} moving into the right branch $\mathrm{t_R}$, the decrease of impurity is defined to be

$$\Delta \mathrm{I}_N(t) = \mathrm{I}_N - p_{\mathrm{L}} \mathrm{I}_N(\mathrm{t_L}) - p_{\mathrm{R}} \mathrm{I}_N(\mathrm{t_R}),$$

where $\mathrm{I}_N = -\sum_i \mathrm{p}_N(i) \log \mathrm{p}_N(i)$ measures impurity at node N, $\mathrm{p}_N(i)$ is the proportion of type-i templates that pass N, $\mathrm{I}_N(\mathrm{t_L})$ measures impurity at branch $\mathrm{t_L}$, and $\mathrm{I}_N(\mathrm{t_R})$ impurity at branch $\mathrm{t_R}$.

The depth of a tree can be controlled in the following way. Whenever a template vector passes a node, we increase the node counter by 1. When the counter value exceeds a certain threshold, a sub-tree is allowed to grow from this node as long as the node does not sit on the deepest possible layer of the tree. The maximum depth of a tree is the same as the dimension of vectors that are input to the tree.

At each leaf, we store the templates that take the leaf as their destination.

3.3 Template Retrieval

Testing vectors are input to template trees and passed through trees in the same way as template vectors. When a testing vector reaches a leaf, we examine the templates stored in the *assembly* of that leaf (Figure 3). The assembly of each leaf consists of that leaf and all its associated leaves. The templates to be retrieved are those that appear in at least V of such assemblies. The assembly of each leaf and the value of V are determined in a training process.

To determine the assembly of each leaf, we consider each leaf L_1 and each template $\boldsymbol{\sigma}$ stored in L_1. We feed in all the training samples that belong to the attraction domain of $\boldsymbol{\sigma}$. If one such training sample reaches leaf L_2, then L_1 is an associated leaf of L_2 and L_1 is thus in the assembly of L_2 (Figure 4).

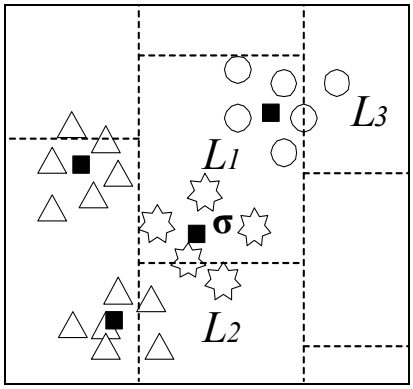

Figure 4. Leaves are separated by dotted lines. L_1 is an associated leaf of L_2 and L_3, since the attraction domains of the two templates stored in L_1 overlap with L_2 and L_3.

The next step is to determine the value V, based on a *second* set of training samples. For this purpose, we feed all these samples into template trees and calculate, for each possible value of V, the proportion P(V) of samples whose nearest templates fall in at least V assemblies. The optimal value of V is then set to be the smallest V such that P(V) exceeds 99.9%.

Having determined the value of V, we can further feed the second set of training

samples into template trees (we cannot do so before the optimal V is determined). It has the effect of enhancing the performance of the retrieval scheme on testing data.

3.4 Order Rearrangement

Thus far, all vectors pass through a template tree according to the original order of their components. As an alternative, they can pass through the tree according to a rearranged order. That is, when a vector **v** enters at a node N, the c^{th} component of **v** is examined at N, if c is the index registered at N.

Order rearrangement helps to reduce the number of retrieved templates from template trees. The idea of order rearrangement is to re-order vector components according a metric that evaluates the *maximal decrease of impurity*

$$\mathrm{m}_N(c) = \max_t \Delta \mathrm{I}_N(t \mid c),$$

where $\Delta \mathrm{I}_N(t \mid c)$ is defined similarly as $\Delta \mathrm{I}_N(t)$ except that the c^{th} component of each vector is being examined at node N.

The index registered at each node is determined layer by layer. The index registered at the root is $i = \operatorname{augmax}_k \mathrm{m}_{\text{root}}(k)$. We then go on to determine the index registered at each node N of the first layer. All indices except i are evaluated at node N. The index j attaining the highest value of $\mathrm{m}_N(\cdot)$ is registered at N, that is, $j = \operatorname{augmax}_{k \neq i} \mathrm{m}_N(k)$. We then proceed to the second layer of the tree. The order rearrangement naturally affects the size of assemblies. It also helps to reduce the average number of retrieved templates, as shown in Section 3.5.

3.5 Testing and Comparison Results

In character recognition tasks, we form 16 template trees. Recall that we obtain 21,595 templates in TC application and 24,145 templates in SC application, each of which is a 256-dimensional vector. We evenly divide each vector into 16 sub-vectors, each of which consists of 16 components. These 16 sub-vectors are then used to grow

the 16 template trees.

A comparison can be made between two different approaches. In the one approach, vectors pass template trees according to their original order. In the other, their orders are dynamically rearranged (cf. Subsection 3.4). Table 3 displays the results of applying these two approaches to the aforementioned testing samples, where the *hit rate* is defined as the proportion of samples whose retrieved templates contains their nearest templates. The results show that the rearranged order has certain advantage over the original order. The average number of retrieved templates, in both approaches, is a very small proportion of total number of templates.

Table 3. Average number of templates retrieved from template trees. There are 21,595 templates stored in the 16 trees.

Order Assumptions of Template Trees	Original Order		Rearranged Order	
Applications	TC	SC	TC	SC
Num. Stored Templates (T)	21,595	24,145	21,595	24,145
Hit Rate on Testing Samples	99.84%	99.92%	99.90%	99.94%
Avg. Num. Retrieved Templates (R)	124	44	95	37
Reduction Rate (R/T)	0.57%	0.18%	0.43%	0.15%

It is also interesting to compare the performance of template trees with that of the K-means clustering method. To make a fair comparison, we apply the K-means clustering method to the 21,595 templates, obtained in the TC application, to form a series of collections, each collection corresponding to a specified value of K (the number of clusters). To determine a retrieval scheme for each collection, we have to use the 390,823 training samples to determine the smallest V such that the probability of finding the nearest template from the first V neighboring clusters exceeds 99.9%. The results are listed in Table 4. Thus, when $K = 50$, the optimal V is found to be 21 and the average number of templates is 9,854.

As indicated by the last row of Tables 3 and 4, the template-trees method has a

clear advantage in the average number of retrieved templates. In addition, the template-trees method takes much shorter time to compute than the K-means clustering method, since the retrieval operation of the K-means method consists of 256 assignments of numerical values, which takes about the same time as computing the distance of an unknown object to one cluster center in the template-trees method. Thus, the ratio of one retrieval-cycle time is about 1:*K* between the two methods, where *K* is the number of clusters obtained by the K-means clustering method.

Table 4. The results obtained by the K-means clustering method.

Number of clusters (*K*)	50	100	150	200	250
Vote (*V*)	21	28	39	43	43
Hit Rate	99.9%	99.89%	99.9%	99.89%	99.9%
Average Number of Templates	9,854	6,425	6,245	5,186	4,309

4. Disambiguation

Our experiment results show very high rates for maintaining target patterns within the first three nearest templates in both TC and SC applications. . There is, however, a non-negligible gap between top-3 and top-1 accuracy rates (1.2% for TC and 0.5% for SC, cf. table 2). Bridging this gap is the purpose of our second-stage procedure, the disambiguation procedure.

A disambiguation procedure consists of working units to be used in both the off-line and online process. The offline process determines which class types are easily mistaken for each other. These confusing types are always taken in pairs and are thus referred to as *confusing pairs*. For each pair, the offline process specifies a single *re-assessing scheme*. The online process relies upon these schemes to reassess the candidates for unknown objects.

4.1 Confusing Pairs

Let us first talk about the offline process. Recall that, in the template construction process, we determine the domain of attraction for each template and also the nearest template for each training sample **x**. At the end of the process, we also sort the templates in ascending order, according to their distance to **x**. In fact, we only require the first K of these templates be sorted, where K is a certain small integer (for example, 2 or 3). These templates will be referred to as *candidates* of **x**. We then collect the pairs (C_1, C_j), where C_j is the rank-j candidate of **x** for $1 < j \leq K$.

When these pairs have been formed for all samples, we make a slight generalization: When a pair consists of a type-A template and a type-B template, both pairs (A, B) and (B, A) are stored in a list. On the other hand, if the target pattern is outside the K-nearest templates, we do not include any of these pairs in the list. Next, for each confusing pair thus determined, we provide a *reassessing scheme*, based on the training samples of the relevant class types.

4.2 Support Vector Machines

We employ support vector machines (SVM) to set up reassessing schemes as follows. For each confusing pair and the training samples corresponding to the two class types associated with this pair, we use SVM to obtain key parameters (**w**, b) for each confusing pair.

SVM is a powerful method for binary classification. The goal of SVM is to derive a separating hyperplane with the largest margin out of labeled training samples $\{\mathbf{x}_i, y_i\}$, $y_i \in \{-1, 1\}$, $i = 1, \ldots, I$, where $\mathbf{x}_i$ is the vector, y_i is the label (1 for one class type and –1 for another) of i^{th} sample, and I is the total number of training samples. In the SVM problem, the labeled training samples satisfy the following constraints:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0. \quad (1)$$

where ξ_i, $i = 1, \ldots, I$, are positive slack variables, $\mathbf{w}$ is normal to the hyperplane, and $|b|/\|\mathbf{w}\|$ is the perpendicular distance from the hyperplane to the origin. Note that $2/\|\mathbf{w}\|$ is the margin between two groups of data.

To introduce positive Lagrange multipliers α_i, one for each inequality constraint (2), and parameters μ_i, one for each ξ_i, we are able to transform the SVM problem into the equivalent optimization constraint problem:

$$\min L = \min\{\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i - \sum_i \alpha_i [y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i] - \sum_i \mu_i \xi_i\} \tag{2}$$

subject to

$$\begin{aligned} &0 \le \alpha_i \le C, \\ &\sum_i \alpha_i y_i = 0 \end{aligned} \tag{3}$$

where C is the penalty parameter and $C\sum_i \xi_i$ is a penalty term, whose duty is to reduce the number of training errors. The solution to the SVM problem is

$$\mathbf{w} = \sum_{i=1}^{N_s} \alpha_i y_i \mathbf{x}_i \tag{4}$$

where N_s is the number of support vectors. SVM is detailed in [2, 16]. A set of useful tools, called LIBSVM, that provides SVM solutions for binary classification problems, can be found on a web site [3]. Our SVM-related experiments are based on this tool kit.

4.3 Online Process

Having determined the reassessing scheme for each confusing pair, we have completed the offline process. Let us now address the online process. Suppose that an unknown object U is given and its first K candidates are already found. We first apply the reassessing schemes to all the confusing pairs found within the K candidates of U. We then rearrange the order of all the candidates as follows.

If the reassessing scheme involves SVM and a confusing pair (A, B), we compute

sgn(**x**·**w**+*b*) for each given testing sample **x**, where **w** is the optimal separating hyperplane associated with (*A*, *B*). If sgn(**x**·**w**+*b*) takes the same sign as the label of **x**, then the class type of **x** scores one unit; otherwise, the opposite class type scores one unit.

When all the confusing pairs contained in the same candidate list are reassessed, we proceed to re-order the involved candidates: The candidate gaining the highest score is ranked first, the candidate gaining the second highest score is ranked second, etc. 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 rank.

4.4 Testing Results

Out of the training results, we take all the pairs (*A*, *B*) as confusing pairs, where *A* and *B* are picked from the top-2 candidates for certain training samples. There are 33,445 pairs obtained by applying the training process to TC, and 40,665 pairs obtained by applying the training process to SC (Table 5). The confusing pairs take only 0.2% of all *N*(*N*-1)/2 possible pairs in both applications.

Table 5. The proportion of confusing pairs out of all possible N(N-1)/2 pairs.

Applications	TC	SC
Number of Classes (N)	5,973	6,767
Number of Confusing Pairs (C)	33,445	40,665
Proportion: 2C/N(N-1)	0.2%	0.2%

Having identified confusing pairs, we apply the SVM method to all relevant top-3 candidates for all unknown objects. The accuracy rates before and after the application of disambiguation methods are listed in Table 6. The testing data are the same as previously described.

The ‘Ideal Disambiguation’ in Table 6 refers to the final outcome under the hypothesis that all confusing pairs are correctly discriminated. It is thus the least upper bound for any binary classifier employed as a reassessing method. The last item

'Top-3 Accuracy', on the other hand, constitutes the least upper bound for the current setting of our disambiguation procedure.

Table 6. Accuracy rates of the disambiguation method in two applications.

Applications	TC	SC
Before Disambiguation	98.60%	99.28%
After Disambiguation	99.67%	99.70%
Ideal Disambiguation	99.71%	99.74%
Top-3 Accuracy	99.80%	99.78%

5. Summary

We have described a series of data reduction methods for pattern recognition tasks. The core of these methods is the template construction algorithm that is able to condense a large set of training samples into a small set of templates. This reduction rate, of course, varies from case to case. In our character recognition applications, we obtained 5.5% and 6.7% for the two testing cases, respectively (cf. Table 2).

The template construction process lays down an important foundation for all the remaining work. In terms of the need of online process, the number of templates remains significantly large for one-to-one matching with unknown objects. Further reduction work needs to be set in action.

The process leading towards this reduction is to retrieve templates that are stored in template trees. When a vector **v** is input into a template tree, **v** will be assigned to one of the partitions according to the value of the designated component of **v**. When all sub-vectors of an unknown object are fed into their corresponding trees, we are able to retrieve templates from all leaves associated with the leaves reached by them.

To determine the associated leaves for each given leaf requires a training process. The training process first builds trees by feeding all templates into the trees and storing them in the destined leaves. It then uses the training samples, from which tem-

plates were actually derived, to determine the associated leaves as well as the value of a parameter that controls the range of templates to be retrieved from template trees. Test results show that the number of retrieved templates on average is 0.15~0.43% of the total number of templates stored in template trees, if rearranged orders of variables are used (cf. table 3).

Testing results show that the template-trees method outperforms the K-means clustering method both in retrieval speed and in reducing retrieved templates. The ratio of one retrieval-cycle time is approximately 1:K between the two methods, while the ratio of retrieved templates is 1:103 when K = 50, and 1:45 when K = 250 (cf. Table 3 and 4).

While template trees are effective in retrieving an extremely small amount of templates with near-zero loss, the disambiguation process is a way for improving recognition accuracy. In the disambiguation process, we re-evaluate top-K candidates to identify the best possible candidate. This requires $N(N\text{-}1)/2$ possible binary classifications, but only a very small proportion of these pairs (0.2% in our applications, cf. table 5) needs to be actually re-evaluated, based on the knowledge that acquired during the learning process. This means that we need to only solve an extremely small set of SVM problems.

In the online process, since only the top-K candidates are involved in the second-stage procedure, there are at most $K(K\text{-}1)/2$ binary decisions to be made. This is a small price to pay in terms of the substantial increase in the accuracy rate at a scale of 0.4~1% in the applications with which we are dealing. (cf. table 6).

Reference:

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, New York: Chapman and Hall, 1984.

[2] C. J. C. Burges, “A tutorial on support vector machines for pattern recognition,” *Knowledge Discovery and Data Mining*, 2(2), 1998.

[3] C-C. Chang and C-J. Lin, “LIBSVM -- A Library for Support Vector Machines”, http://www.csie.ntu.edu.tw/~cjlin/libsvm/

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

[5] R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification,* 2nd Edition, New York: John Wiley, 2001.

[6] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, New Jersey: Prentice Hall, 1988.

[7] A. K. Jain, P. W. Duin, and J. Mao, “Statistical Pattern Recognition: A Review,” *IEEE Transactions on Pattern Analysis and Intelligence*, vol. 22, no. 1, pp. 4-37, 2000.

[8] G. Karypis, E-H. Han, and V. Kumar, “Chameleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling,” *IEEE Computer*, vol. 32, pp. 68-75, 1999.

[9] J. R. Quinlan, “Induction of Decision Tree,” *Machine Learning*, vol. 1, no. 1, pp. 81-106, 1986.

[10] D. L. Reilly and L. N. Cooper, “An Overview of Neural Networks: Early Models to Real World Systems,” *An Introduction of Neural and Electronic Networks*, New York, Academic Press, 1990.

[11] D. L. Reilly, L. N. Cooper, and C. Elbaum, “A Neural Model for Category Learning,” *Biological Cybernetics*, vol. 45, 35-41, 1982.

[12] P. Simard, Y. LeCun, and J. Denker, “Efficient pattern recognition using a new transformation distance,” *Advances in Neural Information Processing Systems*, Morgan Kaufman, San Masteo, Ca, pp. 50-58, 1993.

[13] D. L. Swets and J. Weng, “Hierarchical Discriminant Analysis for Image Retrieval,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 386-401, 1999.

[14] Y. Y. Tang, L.-T. Tu, and Jiming Liu, “Offline Recognition of Chinese Handwriting by Multifeature and Multilevel Classification,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 5, pp. 556-561, 1998.

[15] Ø. D. Trier, A. K. Jain, and T. Taxt, “Feature Extraction Methods for Character

Recognition – A survey," *Pattern Recognition*, vol. 29, no. 4, pp. 641-662, 1996.

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

[17] Q. R. Wang and C. Y. Suen, "Analysis and Design of a Decision Tree Based on Entropy Reduction and Its Application to Large Character Set Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, No. 4, pp. 406-417, 1984.