A Method to Accelerate Writer Adaptation for On-Line Handwriting Recognition of a Large Character Set

Akira Nakamura
Digital Systems Development Center, SANYO Electric Co., Ltd.
180 Ohmori, Anpachi-Cho, Gifu, 503-0195 Japan
naka@gf.hm.rd.sanyo.co.jp

Abstract

An approach to accelerate writer adaptation for on-line handwriting recognition is proposed. It is known that adapting to a writer by learning the writer's own style significantly improves recognition accuracy. However, adapting to a writer can take considerable time until the performance comes up to a satisfactory level, particularly for recognition of a large character set. This paper proposes an adaptation method which uses not only misclassified patterns but also correctly-classified patterns as learning samples. The strategy employed in the method selects acquired prototypes based on their contribution to classification, while treating the misclassified prototypes(i.e. the acquired prototypes that were misclassified before being added) with higher priority when updating the prototype set. The results demonstrate that the proposed method improves the performance and accelerates adaptation especially during the early phase of adaptation. It is also shown that the method yields stable improvement in accuracy over a long period of adaptation with the computational cost acceptable for most real applications.

1. Introduction

On-line handwriting character recognition has already been applied to various applications, such as PDA's and other pen-based computers. However, users' demand for recognition accuracy is still higher than the level achieved with the existing technology. Even if a classifier is trained with learning samples written by various writers, it is still almost impossible to cover all the presumable writing styles. Consequently, its accuracy could be poor for a writer whose writing style is not sufficiently covered by the learning samples.

It is known that adapting to a writer by learning the writer's own style improves recognition accuracy. This approach works well especially on the systems used by a single writer. Since it would be unacceptable for most users to be held up by an extra enrollment program before the normal use of the device, the dynamic writer adaptation method which improves recognition accuracy during normal use has been proposed in the works [1-6].

In these works, the input patterns mislabeled by a classifier are automatically learned based on the user's operation selecting the correct class, so that the similar input patterns will be correctly recognized. In recognition of handwritten alphanumeric characters, Vuori compared several adaptation strategies such as adding new prototypes, deactivating confusing prototypes, reshaping based on Learning Vector Quantization(LVQ), and their combinations[2]. Yokota's approach also employed the algorithm which reshapes the input pattern by averaging with the standard prototype before adding it as a new prototype[3]. Iwayama proposed a hybrid adaptation algorithm which combines learning misclassified patterns and learning input strings[4]. Kimura analyzed the effect of writer adaptation on pen-based computers[5]. Adaptation techniques in off-line word recognition for HMM-based classifier were studied as well[7]. These works demonstrate that writer adaptation significantly improves the recognition accuracy for the writer to whom the classifier is adapted. However, if adapting to a writer takes considerable time, the writer needs to be patient until the performance comes up to a satisfactory level. This is particularly an issue for recognition of a large character set, such as Japanese, Chinese or Korean.

In general, writer adaptation for on-line handwriting recognition poses the following two important issues. One is, of course,

- To what degree the performance is improved through adaptation?

and another is

- How much longer does it take to achieve sufficient accuracy?

Considering from the viewpoint of practical usage, the second one, which has not been well-discussed apart from a rare exception in alphabetical word recognition[6], is particularly a serious issue. Since most users expect to realize the improvement in accuracy as soon as possible, it is desirable that writer adaptation can improve the performance with minimal usage. In other words, the two issues described above can be expressed as:

- To what degree the performance is improved by feeding a certain (and possibly limited) amount of input patterns?

In most of the related works mentioned above, only the

Proceedings of the 9th Int'l Workshop on Frontiers in Handwriting Recognition (IWFHR-9 2004)
0-7695-2187-8/04 $20.00 © 2004 IEEE

misclassified patterns are used as learning samples during adaptation, except Vuori's method also uses correctly-classified patterns based on the k-NN rule[1][2]. Using only the misclassified patterns seems intuitively reasonable, however, assuming a correctly-classified pattern is nearby a decision boundary, in the future such a pattern might prevent a recognition error caused possibly by a pattern which is close to it but located on the opposite side of the boundary. In addition, it should be noted that learning only the misclassified patterns means the classifier can rarely acquire new patterns. Suppose an initial recognition rate is 80%, new learning patterns can be obtained only every 5 characters at the early stage of adaptation, and the frequency will go down as the classifier adapts to the writer.

In this paper, we propose an adaptation method which uses not only misclassified patterns but also correctly-classified patterns as learning samples in order to accelerate the speed of writer adaptation. This method employs a strategy for selecting acquired prototypes on the basis of their contribution to correct/erroneous recognition. The method treats the misclassified and correctly-classified patterns equally when they are added to the prototype set, while the misclassified patterns are given higher priority to be kept than correctly-labeled ones after being added. Positive and negative effects of both the misclassified and correctly-classified learning samples are also evaluated. The results show that the proposed method improves the performance and accelerates adaptation particularly during the early phase of adaptation.

2. Writer adaptation method

2.1 Overview of recognition system

Figure 1 illustrates the overview of our recognition system. In the classifier, an input pattern written by a user is recognized by Okamoto's prototype-based classifier[8], and a list of ranked classes associated with their scores is returned. The list of classes is then verified in the subsequent post-processing step based on linguistic knowledge such as character n-grams and/or lexicon, and a final recognition result is shown on the screen. When the result is wrong, the user chooses the correct class from the list.

The standard and the acquired prototype sets contain the prototypes used by the classifier. The standard prototype set, which consists of the prototypes generated from the learning samples written by various writers, is not modified during the use of the system, while the acquired prototype set is continuously updated based on the input patterns and the operations. In order to control the growth of computational time, the classifier restricts the acquired prototypes compared with the input pattern to only the ones with the stroke count between n_s-1 and n_s+2, where n_s denotes the stroke count of the input pattern. This restriction, which has been ascertained to be effective in reducing computational time through a preliminary experiment, is based on the hypothesis that the variations in stroke counts with a single writer tend to be small enough.

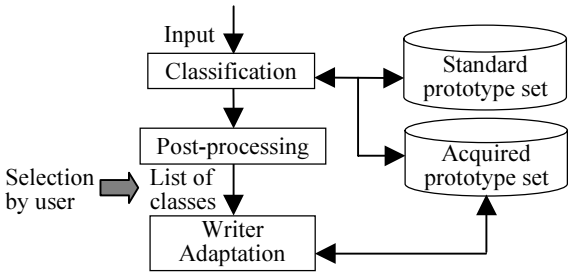

Fig. 1 System overview

Each prototype stored in the acquired prototype set is associated with its true class, the weight value which represents its contribution to the classifier, and the attribute which indicates if it was correctly classified when it was added to the set. The writer adaptation step updates the acquired prototype set by deleting an unnecessary prototype, adding a new prototype and modifying the weight value associated with each prototype already acquired.

2.2 Algorithm for acquiring and selecting prototypes

This subsection shows the algorithm for updating the acquired prototype set. In the following description of the algorithm, we denote the maximum number of the prototypes which can be contained in the acquired prototype set by N_P, the current number of the prototypes actually stored in the acquired prototype set by n, and the maximum number of the acquired prototypes which belong to a same class by N_C, respectively. The two parameters N_P and N_C are incorporated to control the growth of the acquired prototype set.

Let $\Omega=\{\omega_1, \omega_2,\ldots, \omega_M\}$ be the set of M classes and $\boldsymbol{X}$ be an input pattern which belongs to the class ω_k ($1\le k\le M$). Receiving L ranked classes $\boldsymbol{C}(\boldsymbol{X})=\{\omega_1^X, \omega_2^X,..., \omega_L^X\}$, the user selects the correct class from the list when the top candidate ω_1^X is not the true class ω_k. The acquired prototype set is then updated in the following way.

(1)**Deleting an unnecessary prototype**

If the number of the already acquired prototypes which belong to ω_k has reached N_C, or the total number of the prototypes n has reached N_P, one prototype is selected and then deleted before adding $\boldsymbol{X}$ as a new prototype. In both cases, the misclassified prototypes are given higher priority than the correctly-classified ones. This is based on the hypothesis: "the misclassified prototypes probably contribute more than the correctly-classified ones,

although adding the correctly-classified ones is expected to be better than nothing is added." The algorithm first tries to select a candidate for the deletion from the prototypes that were correctly classified when they were added to the set, and if it fails then selects from the misclassified ones. In the following description, $n^{\omega k}$ stands for the number of the prototypes that belong to ω_k.

Case1 ($n^{\omega k}=N_C$): The least contributive prototype is selected to be deleted from the ones whose true class is ω_k. If any prototypes that belong to the class ω_k were correctly classified when they were added, the prototype whose weight value is minimum among them is selected. Otherwise, the prototype whose weight value is minimum among all the prototypes that belong to the class ω_k is selected. The selected prototype will be replaced by the new prototype in the next step.

Case2 ($n^{\omega k}<N_C$ *and* $n=N_P$): The least contributive prototype is selected to be deleted from all the prototypes in the set. If any prototypes were correctly classified when they were added, the prototype whose weight value is minimum among them is selected. Otherwise, the prototype whose weight value is minimum among all the prototypes is selected. The selected prototype will be replaced by the new prototype in the next step.

(2)**Adding a new prototype**

Next, the input pattern $\boldsymbol{X}$ is added to the prototype set as a new prototype regardless of whether the input pattern is correctly classified or not, associated with the true class ω_k, the weight value initialized to a certain value, and the attribute which indicates if it is correctly classified. In order to focus on the evaluation of the strategies for adding and selecting prototypes, in this method any algorithm for reshaping the pattern typified by LVQ is not employed and hence the input pattern is added as such.

(3)**Modifying the weight value**

Based on the influence upon classification, the weight value of each existing prototype is modified in the Following way.

When ith candidate ω_i^X ($1\leq i\leq L$) is the correct class and the candidate class is brought by one of the acquired prototypes, the weight value of the prototype that brought the correct candidate is increased by the value corresponding to the candidate's rank. Then, if any of the incorrect candidates ω_j^X ($1\leq j<i$) upper than the correct candidate is caused by one of the acquired prototypes, the weight value of the prototype that caused the wrong candidate is decreased by the value corresponding to the candidate's rank. The higher the rank, the larger value is increased or decreased in each case.

3. Experimental results

3.1 Adaptation strategies for comparison

In the experiments described below, we evaluated the three kinds of the adaptation strategies as follows:

(1)***Add All patterns and preferentially Keep Misclassified patterns*** (***AAKM***) : All the input patterns are added to the prototype set, while the misclassified prototypes are treated with higher priority than the correctly-classified ones when the algorithm selects the prototype to be deleted on the basis of the weight value of each prototype. This strategy corresponds to the algorithm described in subsection 2.2.

(2)***Add All patterns*** (***AA***) : All the input patterns are simply added to the prototype set and equally treated when the algorithm selects the prototype to be deleted on the basis of the weight value of each prototype.

(3)***Add Misclassified patterns***(***AM***) : Only the misclassified input patterns are added to the prototype set, and the prototype to be deleted is selected on the basis of the weight value of each prototype. This strategy corresponds to the common method used in most related works.

By comparing these strategies, we examined the behavior of the misclassified prototypes and the correctly-classified ones.

3.2 Experiment 1 -Effects of adaptation and the optimum number of prototypes-

Controlling the growth of the acquired prototype set is one of the important issues, especially for a large character set. In the first experiment, the three strategies described above were applied with various values of the upper limit of the total number of the prototypes. The upper limit of the number of prototypes per class was set at 10 through preliminary experiments.

The Japanese character set contains many pairs of similar classes, and some of them even have identical shapes. These classes should be classified through linguistic processing rather than writer adaptation. For this reason, the post-processing step based on bi-gram probabilities[9] is also incorporated in the recognition system, and the experiment was carried out for both cases where post-processing was enabled and disabled. In this experiment, the data sets No.1-80 in HANDS-kuchibue DB[10] were used to generate the standard prototypes, and the data sets No.101-120 were used as the test samples for writer adaptation. Each data set in this DB is written by a single writer. Since the post-processing step is employed, 10154 characters (1537 classes) out of 11962 characters in each data set, which were sampled in a sequence of sentences, were used as the test samples.

Shown in Fig.2 is the evolution of the recognition rates with various values of N_p. Each recognition rate plotted is the total rate from the beginning until the point when the corresponding number of characters were input to the classifier, and averaged over the 20 writers. The result

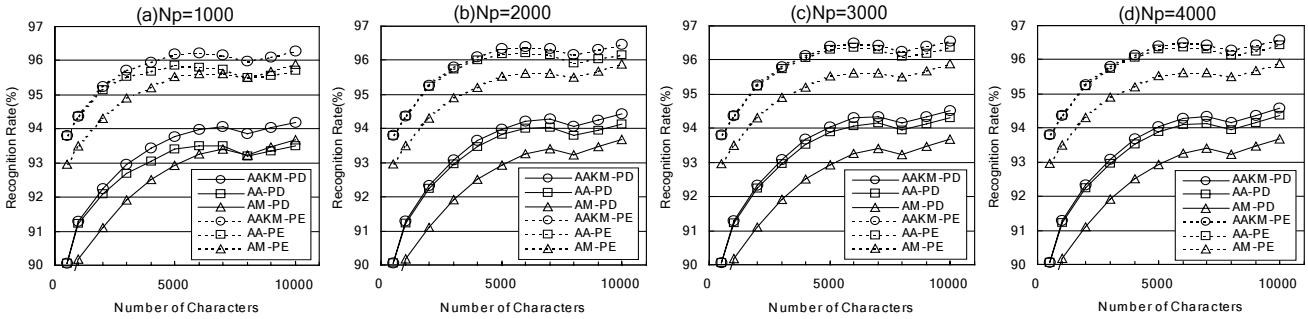

Fig. 2 Evolution of recognition rates
N_p denotes the upper limit of the number of prototypes. "-PD" and "-PE" stand for "Post-processing Disabled" and "Post-processing Enabled", respectively.

shows writer adaptation improves the recognition rates with increasing input characters, and even at the end of the experiment the effect of adaptation does not seem to be saturated yet. ***AAKM*** and ***AA*** yielded much better result than ***AM*** except during the latter stage when N_p=1000, and ***AAKM*** was slightly more accurate than ***AA***. This demonstrates the use of the correctly-classified patterns further improves the performance. Best results were obtained when N_p=4000 with ***AAKM***, however, approximately same accuracy was obtained when N_p=2000. This suggests that 2000 prototypes may be almost sufficient for writer adaptation with ***AAKM***. The recognition rates with ***AM*** did not changed for different values of N_p. This is because ***AM*** obtained much fewer prototypes than the other two. At the end of this experiment, the number of the prototypes averaged over the writers was slightly fewer than 1000 with ***AM***, whereas ***AAKM*** and ***AA*** had reached the limit halfway through the experiment.

Comparing the cases where post-processing was disabled and enabled, although the difference between ***AM*** and the other two strategies are smaller with post-processing, the use of the correctly-classified patterns still yielded better accuracy. The differences of the rates between the cases where post-processing was enabled and disabled were approximately 2% for all the three strategies. This percentage probably corresponds to the pairs of the similar classes which are difficult to classify through writer adaptation but can be solved with linguistic knowledge.

Positive and negative effects of the acquired prototypes are shown in Table.1. In this table, DoE denotes the decrease of classification error due to the acquired prototypes, and IoE denotes the increase of classification error due to the acquired prototypes. DoE and IoE therefore mean the positive and negative effects of adaptation, respectively, and (DoE – IoE) corresponds to the improvement in the recognition rate. The result indicates that both misclassified prototypes and correctly-classified prototypes contribute to the improvement,

Table.1 Positive and negative effects of the acquired prototypes (Np=2000, post-processing is disabled)

		DoE (%)	IoE (%)	DoE – IoE (%)
AAKM	Effects of misclassified prototypes	3.794	0.839	2.955
	Effects of correctly classified prototypes	4.800	1.031	3.769
	Total	8.594	1.870	6.724
AA	Effects of misclassified prototypes	1.985	0.470	1.515
	Effects of correctly classified prototypes	6.160	1.254	4.906
	Total	8.145	1.724	6.421
AM	Effects of misclassified prototypes	7.622	1.648	5.974
	Effects of correctly classified prototypes	0	0	0
	Total	7.622	1.648	5.974

however, the improvement with ***AA*** is mainly due to the correctly-classified prototypes and the improvement with ***AM*** is, of course, caused by only the misclassified ones. The strategy ***AA*** seems to assign too much importance to the correctly-classified prototypes, thereby losing much improvement which could be obtained with the misclassified ones. This suggests the strategy ***AAKM***, which uses both misclassified and correctly-classified input patterns as prototypes while the misclassified ones are treated with higher priority, has the best balance.

3.3 Experiment 2 –Simulation of adaptation over a long period-

Having found the effect of writer adaptation not reaching saturation in the first experiment, we then carried out a simulation over a longer period. The result of the first experiment suggests that the data sets which consist of 10154 characters each are not necessarily sufficient for exploring the process of adaptation, and hence we increased the patterns by reshaping the characters while preserving each writer's style. A character in the original data sets is reshaped by linear transformation as follows:

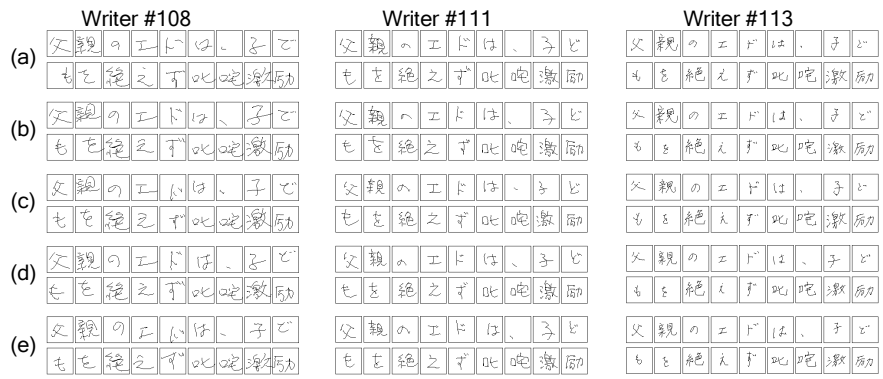

Fig. 3 Examples of generated characters
(a)original data; (b)-(e)generated data

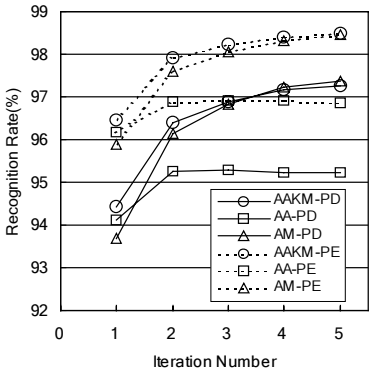

Fig. 4 Result of adaptation during a long period

$$(x', y')^t = \begin{pmatrix} 1+\delta_1 & \delta_2 \\ \delta_3 & 1+\delta_4 \end{pmatrix} (x, y)^t$$

where (x, y) and (x', y') denote the coordinates of a point in a original pattern and a reshaped pattern, respectively, and δ_1, δ_2, δ_3 and δ_4 are the random numbers separately generated within a certain range. A set of four random numbers $\{\delta_1, \delta_2, \delta_3, \delta_4\}$, which specifies the degree of deformation, is generated for every original pattern and commonly used for all the points in the pattern.

By iteratively applying the transformation described above, four synthetic data sets were generated from each of the original data sets. Since most classes contain two or more patterns in a original set, the pattern before the transformation was randomly selected from the ones belonging to the same class. The four random numbers were generated in the –0.1 to 0.1 range. Figure 3 illustrates some examples of the generated patterns, as well as the original ones. In this figure, No.108, 111 and 113 specify the number of the data sets. As shown, the generated patterns appear to be preserving the writing styles of the original ones.

Using the original data sets and the generated data sets, we conducted the second experiment to simulate adaptation over a long period. In this experiment, five data sets from a single writer, namely, the original data set and four generated sets, were treated as the writer's test data. The upper limit of the number of the prototypes was fixed at 2000, which was found to be suitable in the first experiment. Shown in Fig.4 are the evolution of the recognition rates. The recognition rates plotted at the iteration number 1 are the rates for the original data sets, plotted at the iteration numbers 2-5 are the rates for the first to fourth generated data sets, respectively. All the rates plotted are averaged over the 20 writers. As illustrated in this figure, the performance was improved at the second iteration and later. In the case that post-processing was disabled, ***AAKM*** yielded the best performance until the third iteration, and ***AM*** was slightly more accurate than that afterward. On the other hand, in the case that post-processing was enabled, ***AAKM*** outperformed throughout the iteration, although the differences from ***AM*** decreased as adaptation proceeded.

It should be noted that ***AA***, whose performance was better than ***AM*** and close to ***AAKM*** in the first experiment, yielded lower performance than ***AM*** except at the first iteration. Regardless of the presence of post-processing, the accuracy with ***AA*** seems to be saturated around the third iteration. This result demonstrates that using all the input patterns without any consideration could lead to poor result than using only the misclassified patterns in the long run. However, the result also suggests that appropriate use of the correctly-classified patterns as ***AAKM*** may overcome this problem.

4. Discussion

Through the experiments, appropriate use of the correctly-classified patterns improves the performance of writer adaptation particularly during the early phases. Writer adaptation can be regarded as a kind of on-line learning, and the degree of improvement at the early stage is also an important issue. Compared with ***AM***, which uses only the misclassified patterns, ***AAKM*** yields approximately equivalent improvement with half as many input characters as ***AM*** when the number of input characters is fewer than 10000, indicating the advantage of the proposed method.

The performance with ***AA*** was close to that of ***AAKM*** at the early stage, but it saturated at the latter stage of the second experiment. This is probably due to a potentially useful prototype becoming easily replaced by some other new prototype as adaptation with ***AA*** proceeds, whereas such a problem are mostly avoided on ***AAKM***. This problem is basically caused by the fact that the upper limit of the prototype count is too small for the scale of the character set. Suppose we could store an infinite number of prototypes, they would never be replaced at all and hence ***AA*** would possibly yield higher performance. However,

the data sets, which contain about 10000 characters each, appear to be insufficient to examine with larger values of N_p.

Computational speed is also a serious issue for practical use. When N_p=2000 the computational time with ***AAKM*** was 1.21 times as large as when adaptation is deactivated. Focusing on evaluating the effects of misclassified and correctly-classified patterns, we currently employ no particular measures to reduce the number of prototypes, such as reshaping existing prototypes instead of adding new ones. Nonetheless, the current computational time may be acceptable enough for most real applications. Obtaining the equivalent performance with fewer prototypes, say around 1000 or less, would further enhance the advantage of this method.

The generation of test samples in the second experiment should be also discussed. The test samples for the simulation of adaptation over a long period were increased by simple linear transformation, whose validity has not been fully verified yet. Motivated by the insufficiency of learning samples, generation of synthetic patterns with geometrical transformations has been studied[11]-[13]. Although the purpose of these works is to generate unbiased patterns to enlarge the data set written by various writers rather than to preserve the style of a single writer, knowledge of these works may be applied also to writer adaptation.

5. Conclusion

In this paper, we have proposed an adaptation method for handwriting recognition of a large character set. The proposed method treats the misclassified and correctly-classified patterns equally when they are added to the prototype set, while the misclassified patterns are given higher priority to be kept than correctly-classified ones after being added to the set. The experimental results have shown that the proposed method improves recognition accuracy and accelerates adaptation particularly during the early phase of adaptation, suggesting that the method may alleviate the cost of inputted characters for adaptation to the writer's style on a practical system. Until the number of input characters reaches 10000, adaptation with the proposed method proceeds approximately twice as fast as the common method.

It has been also found that appropriate use of the correctly-classified patterns yields stable improvement in recognition accuracy over a long period of adaptation. Although the improvement saturates as adaptation proceeds with the simple approach that equally uses the misclassified and correctly-classified patterns, the proposed method, which treats the misclassified patterns with higher priority when updating the acquired prototype set, achieves better performance with fewer number of the prototypes. The current method employs no particular measures to reduce the number of the prototypes, nevertheless, its computational cost may be acceptable enough for most real applications.

Future works involves applying the method for reducing prototypes while preserving both accuracy and the speed of adaptation. Obtaining the equivalent performance with fewer prototypes would further enhance the advantage of this method. The generation of synthetic sample patterns for the simulation of writer adaptation should be studied as well.

6. References

[1]J.Laaksonen, J.Hurri, E.Oja and J.Kangas, "Comparison of Adaptive Strategies for On-Line Character Recognition", Proc. ICANN'98, pp.245-250, 1998.

[2]V.Vuori, J.Laaksonen, E.Oja and J.Kangas, "On-line Adaptation in Recognition of Handwritten Alphanumeric Characters", Proc. 5th ICDAR, pp.792-795, 1999.

[3]T.Yokota, S.Kuzunuki, K.Gunji and N.Hamada, "User Adaptation in Handwriting Recognition by an Automatic Learning Algorithm", Proc. HCII 2001, pp.455-459, 2001.

[4]N.Iwayama, K.Akiyama and K.Ishigaki, "Hybrid Adaptation: Integration of Adaptive Classification with Adaptive Context Processing", Proc. 8th IWFHR, pp.169-174, 2002.

[5]Y.Kimura, K.Odaka, A.Suzuki and M.Sano, "Analysis and Evaluation of Dictionary Learning on Handy Type Pen-Input Interface for Personal Use", Trans. IEICE Japan, Vol.J84-D-II, No.3, pp.509-518, 2001(in Japanese).

[6]A.Brakensiek, A.Kosmala and G.Rigoll, "Comparing Adaptation Techniques for On-Line Handwriting Recognition", Proc. 6th ICDAR, pp.486-490, 2001.

[7]A.Vinciarelli and S.Bengio, "Writer Adaptation Techniques in Off-Line Cursive Word Recognition", Proc. 8th IWFHR, pp.287-291, 2002.

[8]M.Okamoto and K.Yamamoto, "On-line Handwritten Character Recognition Method Using Directional Features and Clockwise/Counterclockwise Direction Change Features", Proc. 5th ICDAR, pp.491-494, 1999.

[9]M.Nakagawa, K.Akiyama, L.V.Tu, A.Homma and T.Higashiyama, "Robust and Highly Customizable Recognition of On-line Handwritten Japanese Characters", Proc. 13th ICPR, vol.3, pp.269–273, 1996.

[10]M.Nakagawa, T.Higashiyama, Y.Yamanaka, S.Sawada, L.Higashigawa and K.Akiyama, "On-line Handwrittern Character Pattern Database Sampled in a Sequence of Sentences without Any Writing Instructions", Proc. 4th ICDAR, pp.376-381, 1997.

[11]M.Mori, A.Suzuki, A.Shio and S.Ohtsuka, "Generating New Samples from Handwritten Numerals Based on Point Correspondence", Proc. 7th IWFHR, pp.281-290, 2000.

[12]T.Varga and H.Bunke, "Generation of Synthetic Training Data for an HMM-based Handwriting Recognition System", Proc. 7th ICDAR, pp.618-622, 2003.

[13]Y.Waizumi, N.Ebisawa, N.Kato and Y.Nemoto, "Handwritten Character Recognition Using Learning Pattern Generation by Nonlinear Normalization", Trans. IEICE Japan, Vol.J86-D-II, No.10, pp.1391-1399, 2003(in Japanese).