2010 International Conference on Pattern Recognition

Unsupervised Learning of Stroke Tagger for Online Kanji Handwriting Recognition

Mathieu Blondel
Graduate School of System Informatics
Kobe University
Kobe, Japan
mblondel@ai.cs.kobe-u.ac.jp

Kazuhiro Seki
Organization of Advanced Science and Technology
Kobe University
Kobe, Japan
seki@cs.kobe-u.ac.jp

Kuniaki Uehara
Graduate School of System Informatics
Kobe University
Kobe, Japan
uehara@kobe-u.ac.jp

***Abstract*—Traditionally, HMM-based approaches to online Kanji handwriting recognition have relied on a hand-made dictionary, mapping characters to primitives such as strokes or substrokes. We present an unsupervised way to learn a stroke tagger from data, which we eventually use to automatically generate such a dictionary. In addition to not requiring a prior hand-made dictionary, our approach can improve the recognition accuracy by exploiting unlabeled data when the amount of labeled data is limited.**

***Keywords*-kanji; handwriting recognition; HMM; clustering;**

I. Introduction

With the growing popularity of touchscreen-equipped devices such as smartphones and tablet-PC, online handwriting recognition of Kanji (characters of Chinese origin used in Japanese) is finding wide adoption as an input method and in applications including electronic dictionaries, games and educational software. Unlike characters of western writing systems, Kanji include thousands of classes and are written with a large number of strokes. For example, the Japanese Industrial Standard (JIS) X 0208 defines code points for 6879 characters and these characters may contain up to 34 strokes. Moreover, although Kanji have a standard, correct way of being written, in practice, users are likely to use different stroke orders or numbers. A good Kanji handwriting recognition system must therefore be specifically designed to have good runtime speed and be robust to stroke order and number variations.

Hidden Markov Models (HMMs) are popular stochastic models especially known for their application in temporal pattern recognition. Most recent applications of HMMs to online Kanji handwriting recognition have used HMMs to represent primitives such as strokes [1] or substrokes [2], rather than entire characters. This is because the number of such primitives is much reduced compared to entire characters and therefore, the recognition speed can be greatly improved by employing efficient network search. Furthermore, since primitives are shared among different characters, fewer character training samples are needed to converge to the optimal accuracy allowed by the system.

However, unlike entire characters, primitives are usually not labeled in handwriting databases, since labeling each single primitive would be too expensive. Their partitioning is thus necessary, one way or another. For that purpose, previous supervised approaches have usually relied on a hand-made dictionary, mapping characters to their primitives, sometimes with additional spatial structure information [3]. For characters written in free stroke order and number, however, matching primitives to their labels in the dictionary amounts to a pattern recognition problem, which is the problem that we are ultimately trying to solve by training HMMs. Thus, in order to match primitives unambiguously to their labels, only characters written in the correct stroke order and number can be used. A human must therefore carry out the time-consuming and error-prone task of preparing the training data by removing incorrectly written characters.

In [4], a pre-existing hand-made dictionary is used to learn substroke HMMs, which are subsequently used to automatically augment the dictionary with additional stroke order rules. The learned new dictionary can be used for stroke order free recognition. On the other hand, HMM approaches based on cube-search like [5] do not require a dictionary for stroke-order free character recognition, as the different stroke orders are handled efficiently directly in the search procedure. However, these methods have traditionally relied on a hand-made dictionary for the supervised training of stroke HMMs. Finally, in [6], a two-stage stroke clustering scheme is proposed, although not in a probabilistic framework. The purpose is to identify prototypes for template-matching based recognition, that is robust to allographs.

In this paper, we present an approach where no prior hand-made dictionary is needed; the dictionary is entirely learned from data. Stroke HMMs are trained in an unsupervised fashion and the training data need not be manually checked. Because character labels are only necessary for the dictionary generation, this approach can also naturally exploit unlabeled data when labeled data are limited.

1051-4651/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPR.2010.486

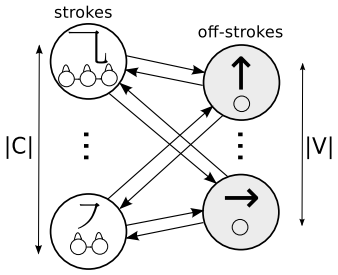

Figure 1. Stroke tagger λ

II. STROKE TAGGER

The approach presented in this paper revolves around what we name a *stroke tagger*, by analogy with part-of-speech (POS) taggers. As its name indicates, the stroke tagger can be used to annotate characters with their corresponding stroke tag sequence. More precisely, we make the distinction between two kinds of pen movements. A *stroke* is defined as the pen movements from a pen-down to the next pen-up. An *off-stroke* is defined as the movement in the air between a pen-up and a pen-down. Formally, a character X_k composed of M strokes S also includes $M-1$ off-strokes $\mathbf{v}$, i.e., $\mathsf{X}_k = S_1, \mathbf{v}_1, S_2, \mathbf{v}_2, \ldots, \mathbf{v}_{M-1}, S_M$. The tagger can be used to find the most likely sequence $W_k = w_1, \ldots, w_N$ of stroke and off-stroke tags corresponding to X_k. If the tagger does not make any mistake, N typically equals $2M-1$. For example, the character "子" may be mapped to the tags "¬ ∅ | ↖ −" (three strokes and two off-strokes).

The stroke tagger, that we name λ and is illustrated in Figure 1, is a composite HMM, constructed by linking stroke HMMs to off-stroke HMMs, and vice versa. As HMMs are first-order Markov processes, stroke states depend only on the previous off-stroke states, and vice-versa. We therefore more specifically call λ a *bigram stroke tagger*. A higher-level, generative way of seeing this single, big HMM is as the model by which all Kanji are generated, and whose states are in turn HMMs. These lower-level HMMs have a tag and emit a stroke or off-stroke. It should be noted that, for clarity, we added actual names to tags. However, in reality, tags are identified by their cluster identifier, which is an integer.

The specificity of the proposed approach is that the tagger is learned in an unsupervised fashion. In particular, we require neither stroke labels in the database nor prior hand-made dictionary. Formally, let $D_l = \{(\mathsf{X}_i, \mathsf{y}_i)\}$ and $D_u = \{\mathsf{X}_j\}$ be sets of labeled and unlabeled characters, respectively. λ is trained using $D_l \cup D_u$, as described in the following sections. In order to initialize the HMMs that compose λ, we first tackle stroke and off-stroke clustering separately.

A. Stroke clustering

In this research, strokes are variable-length sequences of observations (feature vectors). Their clustering into clusters $C = \{C_i\}$ is thus a multivariate time-series clustering problem. We define the distance measure between two strokes $S = \mathbf{s}_1, \ldots, \mathbf{s}_{|S|}$ and $R = \mathbf{r}_1, \ldots, \mathbf{r}_{|R|}$ as $dist(S,R) = D(|S|,|R|)$.

$$D(i,j) = d(\mathbf{s}_i, \mathbf{r}_j) + min\{D(i-1,j), D(i-1,j-1), D(i,j-1)\}$$

D, which is symmetric, is the cumulated distance along the best alignment path up to $\mathbf{s}_i$ and $\mathbf{r}_j$, computed by Dynamic Time Warping (DTW), a dynamic programming (DP) algorithm. $d(\mathbf{a},\mathbf{b})$ is the local distance between the vectors $\mathbf{a}$ and $\mathbf{b}$. For d, we prefer the Mahalanobis distance to the Euclidean distance because the components of the feature vectors have different scales.

We first compute the pairwise distance matrix between pairs of stroke samples. From the distance matrix, we then perform agglomerative, bottom-up clustering. Pairs of clusters are merged together using the group-average criterion until $|C|$ clusters are formed.

In this research, connected strokes are handled by the character decoder (see below). To this respect, although the stroke clustering can cope with characters written in free stroke order, characters written in free stroke number, i.e. characters containing connected strokes, are likely to perturb the clustering process, either by warranting distinct clusters or by becoming outliers of existing clusters. To work around this problem, we temporarily automatically remove characters for which the stroke number is not standard, from the training data. Obviously, this can only be done for labeled characters, therefore the stroke clustering is effectively performed on a subset $D'_l \subset D_l$.

For each cluster, we can now train a corresponding HMM by using the strokes assigned to that cluster. We initialize left-right HMMs by aligning observations to states uniformly and retrain them with the Baum-Welch (BW) algorithm. Rather than using a fixed number of states for each HMM, we found it beneficial to use the following simple technique to estimate the HMM state numbers from data.

$$statenum(C_i) = [A \frac{avgvar(C_i)}{\sum_{C_j \in C} avgvar(C_j)}]$$

$$avgvar(C_i) = \frac{\sum_{S \in C_i} \sum_{t=1}^{|S|-1} d(\mathbf{s}_t, \mathbf{s}_{t+1})}{\sum_{S \in C_i} (|S|-1)}$$

A is an empirically set value which can be interpreted as the number of states that we want for a stroke of average variation over time. The number of stroke clusters $|C|$ has a direct impact on the optimal number of states: a small value of $|C|$ will typically require a large number of states per

HMM and vice-versa. Our simple technique can therefore accommodate the number of states of each HMM, for any value of $|C|$.

Each HMM also includes an additional non-emitting exit state, which when reached, notifies the Viterbi decoder to jump to an off-stroke. This allows the decoder to handle connected strokes.

B. Off-stroke clustering

Off-strokes are single vectors that denote the displacement between two strokes. Their clustering into clusters $V = \{V_i\}$ is thus a much simpler task. Off-strokes are modeled with $|V|$ one-state HMMs. These single states have 1.0 next-transition probability to ensure that no more than one time unit is spent in them. To learn their emission probability density function, we first initialize $|V|$ single-mixture multivariate gaussians randomly, then we let the Expectation-Maximization (EM) algorithm iteratively update them.

C. Iterative retraining

Our stroke tagger λ can now be constructed by linking stroke HMMs to off-stroke HMMs, and vice versa. Similarly to [7], where it has been proposed in the discrete domain, we regard the DTW-based hierarchical clustering as an imperfect yet better than random initial guess, with which we bootstrap our HMMs. In this section, we describe an EM-like method to iteratively refine the quality of the HMMs. Unlike the initialization, it can handle connected strokes.

We set initial stroke to off-stroke and off-stroke to stroke transition probabilities to $1/|V|$ and $1/|C|$, respectively. Strokes (respectively off-strokes) can not reach other strokes (respectively off-strokes) directly. Let $O = \mathbf{o}_1, \ldots, \mathbf{o}_T$ be the observation sequence of a character X_k. Stroke and off-stroke boundaries in O are discarded, therefore O can be seen as a single long stroke. An iteration of our retraining procedure has two steps. First, in the E-like step, for all $\mathsf{X}_k \in D_l \cup D_u$, we find the most likely corresponding tag sequence $Q* = \operatorname{argmax}_W P(O, W|\lambda)$. $Q*$ can be efficiently computed by an alternative formulation of the Viterbi algorithm called the *token passing* or *Viterbi search* algorithm, in which tokens retain the optimal tag sequence while traversing λ. This step provides us with an alignment between observations and tags. Second, in the M-like step, we use the previously computed observation-tag alignments to update tag transition probabilities and to retrain stroke and off-stroke HMMs with BW. The retraining procedure stops when no more progress is made.

D. Dictionary generation and decoding

Once it is trained, λ can be used to annotate new, previously unseen characters with their corresponding most likely tag sequence. For the purpose of automatically generating the dictionary that we need for recognizing new characters, however, we can use the stroke tagger to annotate characters in D_l. For each $(\mathsf{X}_i, \mathsf{y}_i) \in D_l$, a dictionary entry is defined as the mapping between the character label y_i and the tag sequence $Q*$ corresponding to X_i. Importantly, since $Q*$ is also needed for each $\mathsf{X}_i \in D_l$ during the retraining procedure, we need not recompute it and can simply use the one computed in the last iteration of the retraining.

The dictionary may contain several entries per character to support various stroke orders. However, unlikely stroke-orders are pruned out so the dictionary size does not exceed an empirically set size. Contrary to a hand-made dictionary, the automatically generated dictionary is not human-readable since stroke tags correspond to cluster identifiers. This can optionally be addressed by finding one representative per cluster and asking a human to give a name to it.

The system decodes a new feature vector sequence O to a character $W*$ which gives maximum likelihood among all dictionary entries W.

$$\begin{aligned} W* &= \operatorname{argmax}_W P(W|O) \\ &= \operatorname{argmax}_W P(O|W)P(W)/P(O) \\ &\approx \operatorname{argmax}_W P(O, Q|W) \end{aligned}$$

The second line uses the well-known Bayes' rule. $P(O)$ can be safely removed since the maximization is done with respect to W. $P(W)$ is also removed because we consider all W to be equiprobable. Replacing $P(O|W)$ by $P(O, Q|W)$ comes from the fact that we restrict to the likelihood along the best path $Q = q_1, \ldots, q_T$, which allows to use *Viterbi search*. For fast and efficient search during character recognition, the flat dictionary should be converted to a tree or Directed Acyclic Graph (DAG). Pruning can optionally be used during the search by applying a beam width.

III. Experiments

To assert the performance of the proposed approach, we performed open evaluation on the public database "HANDS-kuchibue_d-97-06-10" [8] (hereafter Kuchibue). We used 70% as training data and 30% as test data, for a total of respectively 39,424 and 16,896 characters, from 2965 classes (JIS 1), written in free stroke order. In addition, we used KanjiVG [9], a project under *creative commons* license including 6400 Kanji templates and a description of their component and stroke structure. We used the Tegaki [10] framework as a basis for our implementation.

Characters were linearly rescaled, resampled and converted to feature vectors. Let (x_t, y_t) be the pen-coordinates at time t, $(\Delta x_t, \Delta y_t) = (x_t - x_{t-1}, y_t - y_{t-1})$, $r_t = \sqrt{(\Delta x_t)^2 + (\Delta y_t)^2}$ and $\theta_t = \arctan(\Delta y_t / \Delta x_t)$. Thanks to the off-stroke modeling, $\mathbf{o}_t = (r_t, \theta_t)$ can be used for feature vectors. In our experiments, however, we used $\mathbf{o}_t = (r_t, \theta_t, x_t, y_t)$, as we obtained better results, albeit at the cost of a slower recognition speed. A and $|V|$ were empirically set to respectively 3 and 8. The dictionary size was limited to 20,000 entries.

Table I
ACCURACY COMPARISON

Dictionary	1-best	5-best	10-best
Hand-made	82.3	89.3	91.9
$\|C\| = 30$	84.1	91.6	92.4
$\|C\| = 40$	85.4	91.9	92.1
$\|C\| = 50$	88.7	92.8	93.7

A. Hand-made vs. Generated

In this experiment, we investigated the benefits of a generated dictionary over a hand-made dictionary in terms of accuracy. For this purpose, we replaced the clustering of Sections 2.A, 2.B and 2.C by a supervised learning based on the stroke information from KanjiVG but we also carried out the procedure described in Section 2.D so as to learn additional stroke orders from data, very much like [4].

As shown in Table I, in our experiments, the proposed approach performed favourably compared to the hand-made dictionary approach. This may be explained by several possible reasons. First, the stroke labels in KanjiVG are not necessarily designed to optimize recognition accuracy. In contrast, our clusters may be regarded as optimal, since they are generated from data. Another possible reason is the inevitable errors made during training data preparation (incorrect characters must be removed). Our experiments also show that larger values for $|C|$ lead to better performance without much degradation in the recognition speed, thanks to the efficient search. However, we did not find the accuracy to improve when $|C| > 50$. In comparison, KanjiVG includes 26 stroke classes.

Although the recognition speed of the proposed approach and the hand-made dictionary approach are similar, the training time of the proposed approach was found to require roughly 5 times more time. This is due to the pairwise distance matrix computation and the iterative retraining, which are computationally expensive.

B. With vs. Without unlabeled data

In the previous experiment, we used exclusively labeled data (i.e., $D_u = \emptyset$). In this experiment, we reserved 10,000 characters that we treated as unlabeled for D_u and compared the accuracy with and without unlabeled data for various amounts of labeled characters D_l. This time, in addition to Kuchibue, we also used character templates from KanjiVG to ensure that each character has at least one dictionary entry.

Figure 2 shows that the addition of unlabeled data tends to help improve the system accuracy, especially when labeled data are scarce. This is because D_l is used to generate the dictionary, while both D_l and D_u are used to learn the stroke tagger. This can be useful when labeled data are scarce while unlabeled data (e.g., logs of past input characters) are also available.

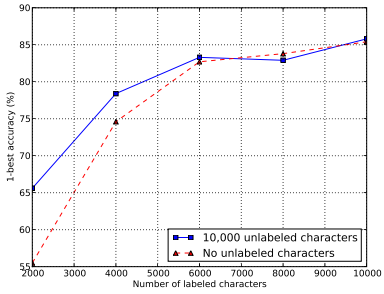

Figure 2. Effect of unlabeled data

IV. CONCLUSION

We presented a novel, unified framework, called *stroke tagger*, for Kanji handwriting recognition, where no prior hand-made dictionary was necessary. In addition to performing favourably compared to a supervised method based on a hand-made dictionary, our approach allowed us to take advantage of unlabeled data in the training. In the future, we would like to extend our work to learn the spatial structure of characters, in addition to stroke tag sequences.

REFERENCES

[1] Y. Katayama, S. Uchida, and H. Sakoe, "A new hmm for on-line character recognition using pen-direction and pen-coordinate features," *ICPR*, pp. 1–4, 2008.

[2] M. Nakai, N. Akira, H. Shimodaira, and S. Sagayama, "Substroke approach to hmm-based on-line kanji handwriting recognition," *ICDAR*, p. 491, 2001.

[3] J. Tokuno, Y. Yang, G. P. da Silva, A. Kitadai, and M. Nakagawa, "Pen-coordinate information modeling by scpr-based hmm for on-line japanese handwriting recognition," *ICPR*, pp. 348–351, 2006.

[4] M. Nakai, H. Shimodaira, and S. Sagayama, "Generation of hierarchical dictionary for stroke-order free kanji handwriting recognition based on substroke hmm," *ICDAR*, p. 514, 2003.

[5] Y. Katayama, S. Uchida, and H. Sakoe, "Stochastic model of stroke order variation," *ICDAR*, pp. 803–807, 2009.

[6] K. Yamasaki, "Automatic prototype stroke generation based on stroke clustering for on-line handwritten japanese character recognition," *ICDAR*, p. 673, 1999.

[7] T. Oates, L. Firoiu, and P. R. Cohen, "Clustering time series with hidden markov models and dynamic time warping," *IJCAI-99 Workshop on Neural, Symbolic and Reinforcement Learning Methods for Sequence Learning*, pp. 17–21, 1999.

[8] M. Nakagawa and K. Matsumoto, "Collection of on-line handwritten japanese character pattern databases and their analyses," *IJDAR*, pp. 69–81, 2004.

[9] "http://kanjivg.tagaini.net."

[10] "http://www.tegaki.org."