Proc. 9th Int'l Conf. on Document Analysis and Recognition (ICDAR 2007) pp.1083-1087

A Visualization Tool to Improve the Performance of a Classifier Based on Hidden Markov Models

Gleidson Pegoretti da Silva, Masaki Nakagawa
Department of Computer and Information Sciences
Tokyo University of Agriculture and Technology
Naka-cho 2-24-16, Koganei, Tokyo, 184-8588, Japan
pegoretti@hands.ei.tuat.ac.jp, nakagawa@cc.tuat.ac.jp

Abstract

This paper presents a visualization tool to improve the performance of a classifier based on the hidden Markov Model. A specific recognition system for which the visualization tool is designed is an on-line handwritten Japanese character recognition system. The recognition system was built from already estimated parameter values, which leads to some difficulties when trying to adjust the system. To tackle this problem we describe how visual information can be helpful to interpret the results and how it can be used to build a set of viewers for helping the tuning task. These viewers were used to examine the data structure and internal procedures of the recognition engine allowing to detect and correct errors in the first implementation. We conclude the paper comparing the two implemented versions of the classifier by showing the increase we achieved in recognition accuracy.

1. Introduction

At present, several types of recognition systems have been developed and are still growing in popularity. Facial, voice, gesture and handwriting recognition systems are good examples of how easy such a system can be found in daily life.

Before the use, a recognition system must be trained by extracting necessary information from a set of training patterns. After that, the system becomes able to recognize and classify a new input data.

Despite the nature of the pattern in real world, the information has to be discretized in order to be used by a machine, resulting in an amount of numbers. If a human has to deal with the internal data of such a system, the meaning of the numbers can be confusing and hard to understand.

In this case, a clean graph might be convenient to represent the information. Since graphs, when properly chosen and presented, express that same data in a way that is easier to comprehend, it gives the person a faster way to detect and correct possible problems.

In this work we present the design of a graphical tool for handwriting recognition based on hidden Markov models (HMM) which uses previously trained parameters. Since we need to make adjustments to fit the estimated data to the new developed system, an easy way to interpret the underlying recognition process is required.

In order to accomplish this task, we propose a implementation of a set of viewers that allow the visualization of some important internal structures as well as results from the procedures.

The rest of this paper is organized as follows. Section 2 gives an overview about HMM. Section 3 proposes the viewers. Section 4 describes the system, giving some details about the features used. Section 5 describes the viewers. Section 6 makes a comparison of the two implemented versions of the system and reports some experimental results. Finally, section 5 concludes this work.

2. Hidden Markov Model

A hidden Markov model (HMM) is a statistical model which can be used to model any time series. It consists of nodes representing hidden states, interconnected by links describing the conditional probabilities of a transition between the states. Each hidden state also has an associated set of probabilities of emitting a particular emission symbol, or also called visible state[2].

A Markov model can be described as a finite state machine which changes its state once every time unit and which generates a observation vector o_t, with a probability density $b_j(o_t)$, every time t that a state j is entered. Furthermore, the transition from state i to state j is also probabilistic and is governed by the discrete probability a_{ij}. These

two matrices A and B that completely describe the model are called model parameters and have the following properties.

$$\sum_j a_{ij} = 1 \quad , \quad \sum_k b_{jk} = 1 \tag{1}$$

Given a set of observation samples, the above parameters can be set so as to best describe training patterns for the known categories. All the transition and emission probabilities can be estimated iteratively from sample sequences using a variation of the Expectation-Maximization (EM) algorithm. After the training, the models can be used to classify a new input pattern. Classification proceeds by finding the single model among candidates that is most likely to have produced a given observed sequence. Or in other words, an input pattern is classified by the model that has the highest posterior probability.

The Bayes's theorem allows the posterior probability to be expressed in terms of a prior probability together with a class-conditional probability $p(O, X|M)$[1].

$$posterior = \frac{likelihood \times prior}{normalization factor} \tag{2}$$

$$P(M|O, X) = \frac{p(O, X|M)P(M)}{p(O)} \tag{3}$$

where O is the sequence of observations, X is the sequence of states and M is the model (actually the model parameters).

Given that X is unknown, the required likelihood is computed by summing up all possible state sequences $X = x(1), x(2), x(3), ..., x(T)$, that is:

$$P(O|M) = \sum_X a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t) a_{x(t)x(t+1)} \tag{4}$$

Alternatively, the likelihood can be approximated by only considering the most likely state sequence, that is:

$$P(O|M) = \max_X \left\{ a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t) a_{x(t)x(t+1)} \right\} \tag{5}$$

As for continuous HMM, the output probabilities $b_{x(t)}(o_t)$ are usually modeled using the normal (Gaussian) distribution. Equation 6 describes the output probability for one feature component using a normal distribution:

$$b(o_t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(o_t - \mu)^2}{2\sigma^2} \right\} \tag{6}$$

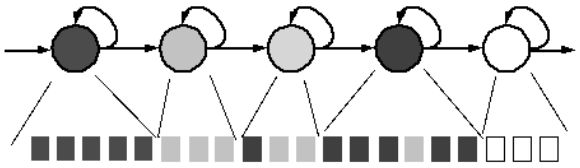

Figure 1. An HMM diagram using color to represent the mapping of an observation sequence

3. Proposition of Visualization

When designing classifiers based on HMM, the Viterbi algorithm[3, 4] is usually chosen to calculate an approximation of the posterior probability.

The result of Viterbi algorithm is the optimum path through the trellis $state \times time$ which segments the pattern by associating each segment to a particular state. It can be said that each state *maps* a particular part of an input pattern.

By looking a HMM diagram, we can say that the states represent a cluster of observation data with mean μ. Although it is complicated to visualize this by using the entire feature vector, it becomes easy if we use just one component of the feature vector at a time. However, we need to create a scheme to represent both pieces of information: the mean of that cluster and the observation vectors (figure 1).

Since it would be helpful to have a better comprehension of this mapping, we propose to build three different viewers. The first one that allows the analysis of the trellis, where the users can view the entire path and examine the associations between the segments and the states.

A second viewer in which the input pattern is plotted after the normalization process and the segmented part of this pattern is highlighted.

Another good way to detect bad segmentation is to see where each feature component will be plotted on a Gaussian graph correspondent to the probability density function of the associated state. This is the third viewer we propose.

4. System overview

This system was totally written in Java language and it consists of a graphical interface which has a writing surface and a set of viewer panels. These panels allow the user to choose a particular model and state, edit some parameters, view some of the internal data structures as well as some procedures results, e.g., the Viterbi path (segmentation).

The recognition engine was not built from scratch. Instead, we used an already estimated parameter file with just

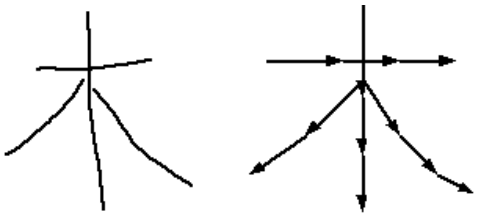

Figure 2. An example of a handwritten pattern with the direction information

a simple description of the features.

The classifier is based on hidden Markov model (HMM) with left-to-right topology. It comprises a total amount of 5600 models.

4.1. Features

An input pattern is the handwritten character that we intend to recognize. In the case of this system all the input patterns are Chinese characters (Kanji). Each Kanji is composed of a set of pen traces called strokes.

Like shown in figure 2, a stroke is just a sequence of points (X-Y-coordinates) recorded between a pen-down event (when pen was put into contact with the writing surface) and a pen-up event (when the pen was lifted from the surface). The line between two consecutive points in a stroke is called a sub-stroke. All the features used by the classifier are taken from each sub-stroke. The segment formed by the pair pen-up and pen-down events is typically called off-stroke.

In order to explain how the features are extracted and used in this system, consider the following pair of consecutive point information $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ from an arbitrary sub-stroke. With the segment $\overline{P_1P_2}$ in mind, the features extracted from this sub-stroke are as follow:

- *X* - The x_2 coordinate from the point P_2.
- *Y* - The y_2 coordinate from the point P_2.
- size - the Euclidean distance of the segment (sub-stroke) $\overline{P_1P_2}$.

$$size = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (7)$$

- angle - the inclination angle of the segment $\overline{P_1P_2}$.

$$angle = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \qquad (8)$$

Figure 3. Three color schemes used to represent feature components

Figure 4. A feature vector representation by using color scheme.

All the n sub-strokes of a given input pattern are processed by the feature extractor, which extracts the four features described above and returns a vector of size n. The off-strokes are also used like a normal sub-stroke by the feature extractor.

5. Viewers

In handwriting recognition, a typical feature used to model an on-line character pattern is a sequence of point coordinates (X,Y) as well as direction information (angle θ formed by two consecutive points). A good way to represent spatial data can be achieved by using some sort of visual information. Because we make intensive use of colors in one of our tools (Viterbi Path Viewer), we need to explain how the colors are used to represent (encode) the information.

We use three color schemes to encode the feature vectors in the system. Figure 3 shows these schemes in a normalized box 128×128 pixels. From left to right we have the color scheme used for the components *X*, *Y* and angle, respectively.

In order to understand how we use the color schemes, consider a pattern drawn inside a normalized box. When extracting the information for *X* coordinate, instead of using the value itself we take the color at the same position according to our color scheme.

As for the angles, we use a unitary vector with direction θ in the middle of the color scheme and take that color to encode this information.

By using these schemes, we can represent a given feature vector by a sequence of boxes colored according to a specific color scheme (figure 4).

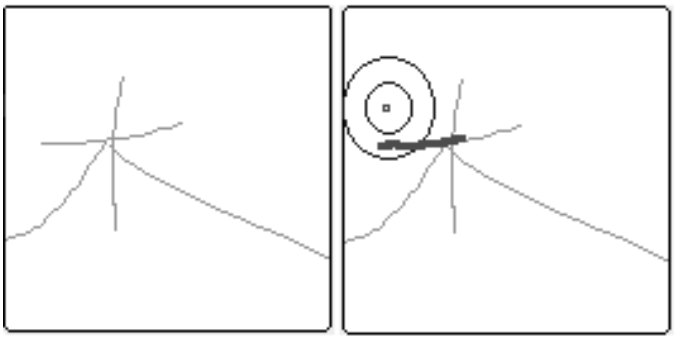

Figure 5. The normalized pattern viewer with a mapped region delimited by circles

Each column of figure 4 represents a feature vector with the components $< \theta, X, Y >$ in a unit of time. And each line comprises a feature vector component along all the observation sequence.

5.1. Normalized Pattern Viewer

We implemented a viewer of the written pattern after the normalization process has been done. The normalization employed is a common linear normalization.

The user can select the model as well as the state he wants to examine. Once the state has been selected, the normalized viewer shows the region mapped by that state based on the parameters for X and Y coordinates.

At the left side of figure 5, a input pattern is shown after the normalization process. At the right, the same pattern is shown just after the user has selected one state of a HMM. At this time, the system draws two concentric circles indicating the region that should be mapped by the selected state. In addition, the system highlights a pattern segment that was really mapped. That way, the user can compare if the mapping result was satisfactory or not.

By looking at the normalized viewer, we realized that some definitions in the original feature extractor differed from ours as described below:

- the normalized pattern had to be centralized in the enclosure box
- the orientation of Y-axis should be set UP to DOWN
- the size and angle should be related to the point P_2 of the segment $\overline{P_1 P_2}$.

5.2. Viterbi Path viewer

The Viterbi Path viewer is a panel that shows a matrix of boxes that represents a trellis $state \times time$. Given an input pattern and a model, this panel shows the optimum path in the trellis determined by the Viterbi algorithm[6].

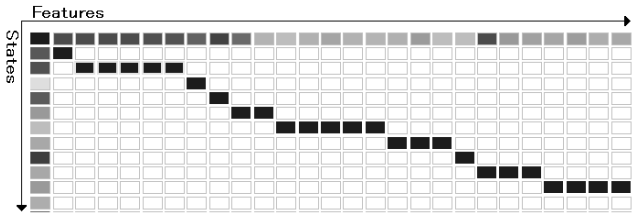

Figure 6. Example of Viterbi Path viewer

In order to help the task of interpreting the segmentation, all the boxes in the first row of the trellis is filled with a color correspondent to the feature vector in that time slot. While the direction information of an input pattern is plotted using the HSV color scheme, as for *size*, X and Y features we use the gray-scale gradient from white to black according to the value.

Regardless which feature component is used, the feature sequence (time) is always plotted along X axis, in the first row of the trellis, like shown in figure 6.

The cells in first column are colored to represent the mean μ_i of the feature component i of each state.

Therefore, by looking at this panel, the user can detect which segments of the sequence are mapped by each state and verify if the segmentation was correctly done.

5.3. Gaussian Viewers

The system uses the probability density function of each state to draw a Gaussian curve for each component of a feature vector. We call this tool a Gaussian Viewer.

Every time the user selects a state, the feature vectors that composes the segment mapped by that state are plotted is the Gaussian Viewer.

Through these graphs, it is possible to see if the components of a given segment fall in the region where the probability is high or low (in case of bad segmentation). Figure 7 shows both cases.

We could verify that in many cases the feature size were plotted far from the expected region in the Gaussian, suggesting something wrong either in the implementation or in the estimated data. As for the other features, the segmentation result was satisfactory.

6. Comparing the implementations

We conducted two experiments in order to compare the accuracy rate of the two versions of the system - the one prior the adjustments and the other after the adjustment.

The database used in these tests was the Kuchibue database[5] which consists of 120 files written by differ-

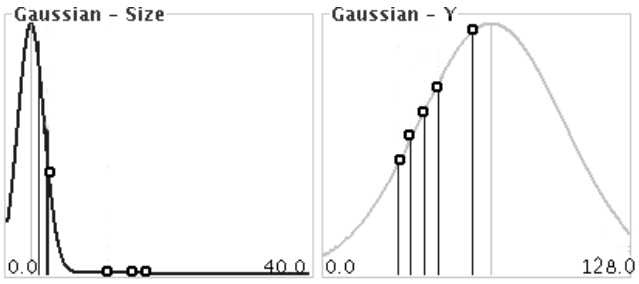

Figure 7. Two Gaussians showing feature vector components of the same segment

Table 1. Best accumulated recognition rates before the adjustments

File	Best1	Best3	Best5	Best10
MDB0116	80.52	90.52	93.45	95.69
MDB0117	88.21	94.84	96.43	98.13
MDB0118	80.14	91.63	94.89	97.04
MDB0119	85.25	92.56	94.37	96.12
MDB0120	86.80	94.55	96.68	98.30
Total	84.18	92.82	95.16	97.06

ent people. The results reported here for both the experiments show the recognition accuracy for the last 5 files of the database. The total amount of tested patterns for these experiments are over 20000 and the system recognizes 2965 classes.

The first implementation uses the sub-stroke size as a feature and the very first implementation of the feature extractor and normalization procedure. Table 1 shows the accumulated recognition rate for the Best 1, Best 3, Best 5, and Best 10 candidates, and the overall performance.

By using the Gaussian viewers and the Viterbi Pattern Viewer, we realized that the component $size$ of the observation sequence was not according to the values expected by the estimated parameters. Most of times the estimated $mean$ for the component $size$ was around one third of the extracted value, regardless the model and input pattern tested. Because the error occurred in a systematic way, we decided no longer use the feature $size$.

Moreover, by looking at the Normalized Pattern Viewer we could detect and correct the differences between the feature extractor used by the systems that performed the estimation and our system.

The second experiment was conducted by suppressing the feature size and applying adjustment on the feature extractor. Table 2 shows the results reaching 96.93 percent for the Best 1 candidate.

Table 2. Best accumulated recognition after the adjustments

File	Best1	Best3	Best5	Best10
MDB0116	96.10	98.99	99.20	99.46
MDB0117	97.69	99.68	99.82	99.95
MDB0118	95.39	98.79	99.41	99.82
MDB0119	97.47	99.48	99.75	99.88
MDB0120	98.00	96.64	99.79	99.95
Total	96.93	99.30	99.59	99.81

7. Conclusions

In this paper, we have presented a tool for helping in the tuning of a handwriting recognition system based on HMM built from an already estimated set of parameters.

By using a set of graphical helper tools it was possible to detected and correct errors in the feature extractor implementation. Moreover, systematic cases of bad segmentation could be found, suggesting the elimination of one feature that was not contributing to the classification.

After the adjustment was made, our classifier showed an increase in performance of more than 10% over the original system. Such an improvement justifies the modifications made, and demonstrates the usefulness of graphical helper tools in the tuning of handwriting recognition systems.

Acknowledgement

This research is being supported by the MEXT Research and Education Fund for Promoting Research on Symbiotic Information Technology.

References

[1] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[2] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.

[3] J. Forney, G.D. The Viterbi algorithm. *Proceedings of the IEEE*, 61:268–278, 1973.

[4] L. Lou, H. Implementing the Viterbi algorithm. *Signal Processing Magazine, IEEE*, 12:42–52, 1995.

[5] M. Nakagawa and K. Matsumoto. Collection of on-line handwritten Japanese character pattern databases and their analysis. *IJDAR*, 7(1):68–81, 2004.

[6] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition*, pages 267–296. Kaufmann, San Mateo, CA, 1990.