Dynamic Time Warping Applied to Tamil Character Recognition

Ralph Niels and Louis Vuurpijl
Nijmegen Institute for Cognition and Information
{*r.niels,vuurpijl*}*@nici.ru.nl*

Abstract

This paper describes the use of Dynamic Time Warping (DTW) for classifying handwritten Tamil characters. Since DTW can match characters of arbitrary length, it is particularly suited for this domain. We built a prototype based classifier that uses DTW both for generating prototypes and for calculating a list of nearest prototypes. Prototypes were automatically generated and selected. Two tests were performed to measure the performance of our classifier in a writer dependent, and in a writer independent setting. Furthermore, several strategies were developed for rejecting uncertain cases. Two different rejection variables were implemented and using a Monte Carlo simulation, the performance of the system was tested in various configurations. The results are promising and show that the classifier can be of use in both writer dependent and writer independent automatic recognition of handwritten Tamil characters.

1. Introduction

As in many Asian alphabets, the Tamil alphabet contains so many characters that easy to use keyboard mapping systems do not exist. An accurate automatic handwriting recognition system could be a solution to this problem: when using it with a pen and writing tablet, an alternative input device could be created.

For the automatic recognition of online characters, a matching technique like Dynamic Time Warping (DTW) [4] can be used. One of the advantages of DTW is that it is able to match two curves of unequal length. This means that no resampling is needed for the technique to match them.

Resampling can be a problem in the case of matching characters of different length. It is known that in the Latin alphabet, a resampling to 30 points can do justice to short characters, like the '*c*', as well as long characters, like the '*m*' [7]. In the Tamil alphabet, which is a syllabic language, there is a large difference between the length of short characters, like the 'ட' and long characters, like the 'ணூ' (see Table 1). When these characters are resampled, undersampling can cause loss of vital information, whereas oversampling increases the computational complexity.

Our DTW-classifier has already proved to be useful for classifying Latin characters [5, 6, 9], but recent research [3] has raised the interest of using our DTW-classifier for the Tamil alphabet.

To test the performance of the classifier on unseen data, we have conducted three experiments: (i) a writer dependent test, to test the performance of the system on data that was produced by writers that also produced the train data, (ii) a writer independent test, to test the performance of the system on data that was produced by writers other than the writers that produced the train data; and (iii) a rejection test, to test the behavior of the system in a writer independent setting, where it only accepts classifications that pass a particular confidence threshold.

The rest of this paper is organized as follows. Section 2 describes the dataset that was used for creating and testing the classifier. Section 3 describes our Dynamic Time Warping implementation and how DTW can be equipped with rejection capabilities. Sections 4 and 5 describe the three tests that were conducted. Finally, in Section 6, our conclusions are described.

2. Dataset

The data [3] was recorded using two different devices. 15 writers wrote on an iPAQ pocket PC with a sampling rate of 90 points per second, and 25 writers wrote on a HP TabletPC with a sampling rate of 120 points per second (please note that recording devices with different temporal resolutions will yield characters with different numbers of points). Each of the 40 writers wrote 10 instances of 156 different Tamil characters. The characters were written in separate boxes, so that no segmentation was needed.

All data was recorded and stored *online*, which means that not only the x and y coordinates of the points were stored, but also the order in which the points were produced. Additionally, the z coordinate, which indicates whether the pen was on (pen-down) or off (pen-up) the tablet was recorded. Each sample character was labeled, to indicate the

Character	Device	Lowest	Highest	Average
ட	iPAQ	11	54	18.4
	HP Tablet	21	96	44.8
ப	iPAQ	13	80	22.9
	HP Tablet	30	98	44.8
ஞா	iPAQ	50	192	89.1
	HP Tablet	125	470	220.3
ணீ	iPAQ	49	166	79.9
	HP Tablet	117	318	180.6

Table 1. Short versus long characters. Character length in number of points of the data recorded with iPAQ pocket PC and HP TabletPC. The lowest and highest occurrence, as well as the average number of points in one sample are mentioned.

Tamil character it was representing. All characters were normalized by equally translating their center to (0,0) and scaling their RMS radius to one. Table 1 shows the variation of the number of points of different Tamil characters.

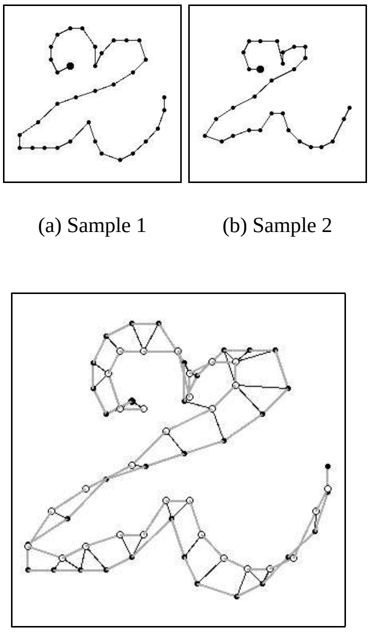
3. Dynamic Time Warping classifier

The classifier we built is prototype based, which means that an unknown sample is compared to a set of labeled prototypes, and is classified using the label of the nearest prototype. In our classifier, DTW is used to calculate the distance between the sample and a prototype. This section describes our DTW-implementation and a feature, called *rejection*, that was added to the system to give it the possibility to either accept or reject a classification, based on a certainty value.

3.1. Dynamic Time Warping

DTW is a technique that matches two trajectories (handwritten characters) and calculates a distance from this matching. In our DTW-implementation (that is based on the one described by Vuori [8]), given two trajectories $P = (p_1, p_2, ..., p_N)$ and $Q = (q_1, q_2, ..., q_M)$, two points p_i and q_j match if the following three conditions (with decreasing priority) are satisfied: (i) *Boundary condition*: p_i and q_j are both the first, or both the last points of the corresponding trajectories P and Q; (ii) *Penup/Pendown condition*: p_i and q_j can only match if either both are pen-down, or if both are pen-up (this condition is an addition to the implementation described by Vuori); and (iii) *Continuity condition*: p_i and q_j can only match if Equation 1 (where c is a constant between 0 and 1, indicating the strictness of the condition) is satisfied.

$$\frac{M}{N}i - cM \leq j \leq \frac{M}{N}i + cM \qquad (1)$$



(a) Sample 1

(b) Sample 2

(c) DTW-match

Figure 1. Example of a DTW-match of character ஐ.

The algorithm computes the distance between P and Q by finding a path that minimizes the average cumulative cost. In our implementation, the cost $\delta(P, Q)$ is defined by the average Euclidean distance between all matches (p_i, q_j). An illustration of a DTW-match between two Tamil characters can be seen in Figure 1.

Given an unknown sample, DTW calculates the distances to all prototypes, sorts them by distance, and returns the nearest prototype.

On a system with a Pentium 4 processor (3.60 GHz) and 1 GB RAM, the classification of an average sample, using a prototype set containing 1846 samples, takes about 1.9 seconds. This involves that for the envisaged interactive applications, DTW is a relatively time-consuming technique. In particular for samples with a large number of points, and in particular for word recognition. However, for 60% of the samples (with less than 50 coordinates), recognition is performed in less than a second.

3.2. Rejection

In addition to the DTW-algorithm, we provided our classifier with the possibility to reject a classification. We implemented two variables that were used by the system to judge the certainty of a classification. If this certainty is below the set threshold, the classification is rejected.

The two variables that were implemented are:

(i) *Agreement*: This certainty value is calculated using a list of the five prototypes that are nearest to the sample.

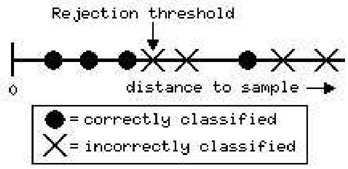


Figure 2. Rejection distance. The distance between a prototype and the nearest sample for which it caused an incorrect classification is used as rejection threshold.

First, the label of the nearest prototype is decided. The certainty value is the amount of the other four prototypes that agree with this label. If this certainty is high, it means that the chance that the label of the nearest prototype is correct is high, and that the chance that an incorrect prototype has ended up as being the nearest prototype is small. This certainty value always is an integer between 0 and 4 (inclusive), where a value of 0 represents the lowest certainty, and the value of 4 represents the highest certainty.

(ii) *Rejection distance*: For each prototype, a rejection distance is calculated. If the distance between the unknown sample and the nearest prototype is higher than this distance (or a multiplication of the distance), the classification is rejected. The rejection distances for the prototypes are calculated by classifying a set of unseen samples, and recording the distances for incorrect classifications. The rejection distance for each prototype is set to the distance between the prototype and the nearest sample for which it caused an incorrect classification (see Figure 2).

When using rejection, the recognition performance of the classifier is not the only measure to judge its quality. We used three other performance measures: (i) % accepted (proportion of the samples of which the classification was below the rejection threshold); (ii) % false accepts (proportion of the samples that was accepted by the system, while the classification was incorrect); and (iii) % false rejects (proportion of the samples that was rejected by the system, while the classification was correct).

Note that the two variables that were implemented can be used together: a classification is only accepted when both thresholds are reached.

4. Writer dependent test

The writer dependent test was conducted to find out how well the system performs when the prototypes and the classifier options are based only on data that was produced by a writer that also produced the train data.

The following procedure was repeated for a random selection of 10 different writers. For each writer, the recognition performance was recorded. The ten numbers were then averaged, resulting in a general recognition performance of the system in a writer dependent setting.

Writer nr.	Recogn. perf. (%)
07	90.384615
09	95.192308
12	93.910256
15	92.307692
16	79.166667
21	89.102564
30	88.782051
34	76.923077
38	81.410256
40	90.384615
Average	87.756410

Table 2. Recognition performance of the system in the writer dependent test.

The data produced by the writer was divided into three sets: (i) *Trainset1*, containing 5 of the 10 instances of each character; (ii) *Trainset2*, containing 2 of the remaining instances; and (iii) *Testset*, containing the 3 remaining instances.

All samples in *Trainset1* were used as prototypes (no editing or averaging was performed). *Trainset2* was offered a number of times to the system using that prototype set, each time with a different c-value (see Equation1) to find the c-value that produces the best recognition performance.

Finally, *Testset* was offered to the classifier using the prototype set and the c-value found in the previous step to find the recognition performance of the system on unseen data.

The recognition performances of the system for each of the 10 tested writers can be found in Table 2. The average performance of the system on these 10 writers was 87.76%.

5. Writer independent tests

The writer independent tests were conducted to test how well the system performs when the prototypes and the classifier options are based on data that was produced by writers other than the ones that produced the train data. Also, the effects and performance of the *rejection* option in a writer independent setting were examined.

The two tests were preceded by the automatic creation of prototypes and optimization of the classifier settings. The same prototypes and settings were used for both tests.

The complete data set was divided into four sets: (i) *Trainset1*, containing all instances of all characters written by a random selection of 20 writers; (ii) *Trainset2*, containing 2 of the 10 instances of all characters written by a random selection of 10 of the remaining writers; (iii) *Testset*,containing all instances of all characters written by the 10 remaining writers; and (iv) *Rejectionset*, containing 5 of the remaining instances of all characters written by the writers from *Trainset2*.

5.1. Automatic prototype creation

Trainset1 was used for the creation of the prototypes. First, the set was divided into 156 subsets (one for each character) that were processed one by one. For each of the subsets, a Monte Carlo (MC) simulation was used to create 72 different prototype sets, by varying options of the algorithms that were used in the steps that were taken.

For each of the 156 subsets of *Trainset1*, the next steps were taken:

(i) *Distance calculation*: A matrix of the DTW-distances between all samples in the subset was calculated. We used the DTW-distance in the training process because the same metric would be used in the testing process. This made sure that the positioning of the prototypes in the feature space would be optimal for the testing. In the MC simulation, two different c-values (see Equation 1) were used;

(ii) *Clustering*: Using the distance matrix produced in the previous step, all samples from *Trainset1* were clustered through agglomerative hierarchical clustering [10]. As described in [10], various parameters rule the clustering process (and resulting clusters). In the MC simulation, these parameters were varied;

(iii) *Cluster selection*: In this step, a number of clusters from the complete cluster structure created in the previous step, was selected. This was done by deciding for each cluster whether it passed a number of thresholds. In the MC simulation, the thresholds for the number of members in one cluster, and the maximum average distance between members and centroid were used;

(iv) *Merging*: The members of each cluster that was selected in the previous step were merged into one prototype. An algorithm based on Learning Vector Quantization [8] and DTW [5] was used for this. One sample was selected from the cluster, and another sample was merged with it. The resulting trajectory was merged with the next sample, and this continued until all samples from the cluster were processed. In the MC simulation, two different c-values (see Equation 1) were used.

Using the Monte Carlo simulation, 72 different prototype sets (containing between 179 and 1911 prototypes) were created. Every prototype set was processed by our classifier, using two different c-values. This resulted in 128 different recognition performance percentages, of which 79.03% was the highest. The combination of the prototype set and the c-value that generated this performance was indicated as optimal. This set (containing 1846 prototypes) and c-value were used for the tests.

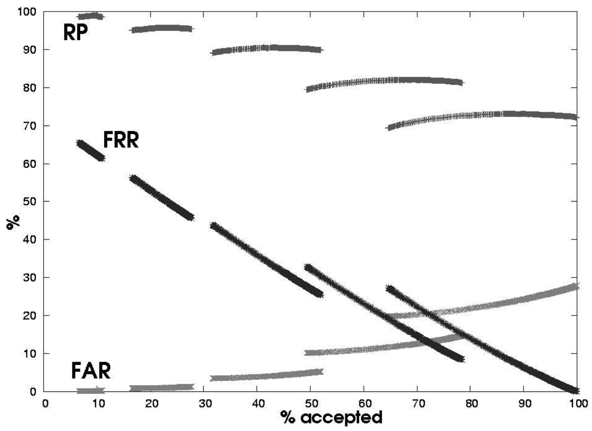


Figure 3. Rejection behavior. Visualization of the proportion of samples that are both accepted and classified correctly (recognition performance on accepted samples, RP), the false acceptance rate (FAR), and the false rejection rate (FRR) for each acceptance percentage.

5.2. Writer independent test

To test the performance of the system using the optimal prototype set and c-value, as determined in the previous steps, we offered *Testset*, containing 15600 allographs, to the system. The system correctly classified 72.11% of the samples.

5.3. Rejection test

To test the behavior of the system using the optimal prototype set and c-value, as determined in the previous step, a *rejection list* was created, and different rejection thresholds were tried to classify *Testset*. Because the train data was produced by other writers than the ones that created the test data, the results found in this test are writer independent.

A list of rejection distances was created using the method described in Section 3.2. The samples in *Rejectionset* were offered to the classifier using the optimal prototype set and optimal c-value. For each prototype that was at least once responsible for an incorrect classification, a rejection threshold was set. The created list was used in the next step.

We varied the strictness of the rejection by changing the *Agreement* threshold and the multiplication factor of the *rejection distance* (see Section 3.2). In a Monte Carlo simulation, 50000 different combinations of thresholds were tried (of which 1494 combinations actually generated different results), and Figure 3 shows the results.

On the horizontal axis, the proportion of samples that are accepted by the system is represented. This proportion is inversely correlated to the strictness of the *rejection* settings. The vertical axis shows (i) the proportion of samples that are both classified correctly and accepted by the system (the recognition performance on the accepted samples); (ii) the false acceptance rate: the proportion of samples that are classified incorrectly but were accepted by the system, as a percentage of the total amount of samples; and (iii) the false rejection rate: the proportion of samples that are classified correctly, but rejected by the system, as a percentage of the total amount of samples.

In the ideal situation, the number of false accepts and false rejects is minimized, while the amount of accepts is maximized. As can be seen in Figure 3, it is not possible to satisfy these constraints. Some settlement, depending on the application of the system, is needed.

6. Conclusion

DTW is able to compare trajectories of arbitrary length, which makes it suitable for comparison of Tamil characters. Also, as shown in previous research [6], it produces matches that are more visually perceptive and intuitive than that of other systems, which makes DTW a suitable technique for improving the user acceptance of a handwriting recognition system.

Two recognition performance studies and one assessment of different rejection strategies have been performed to determine the suitability of our DTW-classifier on handwritten Tamil characters. A performance of 87.76% in a writer dependent setting, and a performance of 72.11% in a writer independent setting were achieved.

These results cannot easily be compared to other studies, because of differences in the employed data and since the number of used Tamil characters differs among studies. In literature, performances of 88.22 to 96.30% were yielded with 156 characters, in a writer dependent setting [3]. Furthermore, results between 71.32 and 91.5% in a writer independent setting can be found when classifying 96 characters [1] and 79.9% when classifying 26 characters [2]. In this perspective, our results of more than 72% for 156 classes seem relatively good.

The outcomes from the experiments described here show that our DTW-implementation is suitable for the automatic recognition of Tamil handwriting and that, when using the rejection strategies, the reliability of the classifier can be improved. Although the recognition time using DTW is relatively high, for 60% of the characters, response times of less than a second are achieved.

As discussed in [9], the DTW-classifier is somewhat orthogonal to other classifiers, which makes it a proper candidate for multiple classifier systems (MCS). Our research has shown that the usage of DTW in an MCS improves the recognition performance.

Our current efforts are targeted on gaining more insight in the strong and weak points of our classifier. A detailed analysis of the performance of the system per character, and the confusion between characters, could show which characters are the problem cases, and what the properties of these characters are (e.g. are long characters better classified than short characters, are curly characters easier mixed up than non-curly characters?). This information can be used to further improve the performance of the system, not only for the recognition of Tamil, but also of Latin characters.

7. Acknowledgments

HP Labs, Bangalore, India are acknowledged for making their data available to us.

This research is sponsored by the Dutch NWO TRIGRAPH project.

References

[1] H. Aparna, V. Subramanian, M. Kasirajan, V. Prakash, V. Chakravarthy, and S. Madhvanath. Online handwriting recognition for Tamil. In F. Kimura and H. Fujisawa, editors, *Proc. IWFHR9*, pages 438–442, Tokyo, October 2004.

[2] S. Hewavitharana and H. Fernando. A two stage classification approach to Tamil handwriting recognition. In *Proc. TI2002*, Foster City, California, USA, September 2002.

[3] N. Joshi, G. Sita, A. G. Ramakrishnan, and S. Madhvanath. Comparison of elastic matching algorithms for online Tamil handwritten character recognition. In F. Kimura and H. Fujisawa, editors, *Proc. IWFHR9*, pages 444–449, Tokyo, October 2004.

[4] J. Kruskal and M. Liberman. The symmetric time-warping problem: from continuous to discrete. In D. Sankoff and J. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparisons*. Addison-Wesley, Reading, Massachusetts, 1983.

[5] R. Niels. Dynamic Time Warping: An intuitive way of handwriting recognition? Master's thesis, Radboud University Nijmegen, November-December 2004. `dtw.noviomagum.com`.

[6] R. Niels and L. Vuurpijl. Using Dynamic Time Warping for intuitive handwriting recognition. In *Proc. IGS2005*, 2005. *In press*.

[7] L. Schomaker. Using stroke- or character-based self-organizing maps in the recognition of on-line, connected cursive script. *Pattern Recognition*, 26(3):443–450, 1993.

[8] V. Vuori. *Adaptive Methods for On-Line Recognition of Isolated Handwritten Characters*. PhD thesis, Finnish Academies of Technology, 2002.

[9] L. Vuurpijl, R. Niels, M. van Erp, L. Schomaker, and E. Ratzlaff. Verifying the UNIPEN devset. In F. Kimura and H. Fujisawa, editors, *Proc. IWFHR9*, pages 586–591, Tokyo, Japan, October 2004.

[10] L. Vuurpijl and L. Schomaker. Finding structure in diversity: A hierarchical clustering method for the categorization of allographs in handwriting. In *Proc. ICDAR '97*, pages 387–393, Piscataway, NJ, USA, August 1997. IEEE Computer Society.