Handwritten Devanagari Script Segmentation using Support Vector Machines

Gaurav Agrawal, Kshitij, Amitabha Mukerjee
Department of Computer Science & Engineering
Indian Institute of Technology
Kanpur, INDIA 208016
E-mail: gauagr, kshit, amit@iitk.ac.in

Nimit Kumar
Department of Electrical Engineering
Indian Institute of Technology
Kanpur, INDIA 208016
E-mail: nimitk@iitk.ac.in

***Abstract*— In this paper a novel method for Devanagari handwriting segmentation using Support Vector Machines is proposed. Given a handwritten Devanagari script, the purpose is to segment is and recognize the characters .This uses to large extent the knowledge of what the character is, the system can be easily extended into a handwriting recognizer. We use several preprocessing algorithms to refine the search method. All text lines are extracted with the *shirorekha* removed and then broken into constituting words using a simple connectivity test algorithm. The characters within a word are identified and segmented using a polynomial kernel based Support Vector Machines. Further heuristics are used to recheck the segments obtained.**

I. Introduction

Machine recognition of handwritten documents has a variety of commercial and practical applications in reading forms, manuscripts, their archival etc. A direct application is in making the railway reservation procedure completely automated by processing the forms filled by the intended passengers. It is a great help to visually handicapped and illiterate people when integrated with voice synthesizer. Although a number of commercial systems are available for reading English texts such systems for Devanagari script are still in research and development stage as they pose difficulties unaccounted in their roman counterpart.

Most of the earlier research in this problem focused only on either segmentation or recognition. They were considered to be two well separated problems with segmentation being the pre-requisite of the recognition phase. But we identified that these two problems were actually integrated & inter-dependent problems. To segment into constituting characters correctly, one has to have some knowledge of the class to which the input belongs & hence this requires recognition at a low level. Similarly, for recognition at a high level, segmentation at a lower level is essential followed by using semantics & low level recognition information. For recognition conventional techniques used are matching certain properties like curvature, template match etc. But all these have various limitations. Hence Support Vector Machines were identified as a potential solution for low level recognition.

First step towards recognition of a handwritten sample is the extraction of text zone. Further processing of the extracted text zone is then done. This processed output is sent for segmentation through which each line will be segmented into words and then each word to its constituting characters. At this level one is able to predict a class to which the character extracted belongs. This information can be used in final recognition of the sample.

In this paper we present some of the major aspects of the design of a Devanagari manuscript segmentor with a view to recognise the text after segmenting it into constituting characters. The objective of the design is to develop a uniform methodology by which any handwritten sample can be tackled. Further, after segmenting a sample we also wish to extract enough information about the class to which the character belongs so as to provide a firm base for our next level of recognition. Unlike simple juxtaposition of characters in roman script and idiographic scripts like Chinese, Japanese and Korean, Devanagari is an alphabetic script with a complex composition of constituent symbols. Treating the word or even a composed unit as an atom for recognition purpose will be an unnatural way of dealing with the script.

II. Support Vector Machines

A support vector machine (SVM) [1] maps the input vectors of a sample space into the higher dimensional feature space through a nonlinear mapping so that an optimal separating hyperplane can be constructed. In this sense the SVM works in the same spirit as the Rosenblatt's perceptron introduced by Frank Rosenblatt [2]. However, the success of the SVM lies in the selection of the aforementioned nonlinear mapping function, which must be selected prior to the optimization involved in the SVM method. To avoid this explicit dependence over the nonlinear mapping function, the idea of the kernel function was proposed inspired from the Hilbert-Schmidt theory. The kernel function here defines an inner product on a Hilbert space.

Given a set of N training samples,$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, the generalized SVM solves the dual quadratic optimization problem as in Equation 1 and determines an optimal separating hyperplane.Solution to Equation 1 based on the conditions in Equation 2 and 3 leads to the maximization of the margin of two classes and minimization of the training error simultaneously. This is made possible because of the Structural Risk Minimization Principle [3].

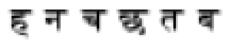

Fig. 1. Some typical Devanagari characters.

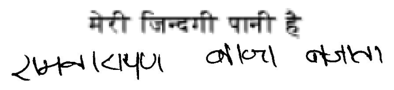

Fig. 2. OCR Devanagari Script and Handwritten Devanagari Script (with *shirorekha* removed). Note the variabilities in the Handwritten samples.

$$\mathbf{MAX}_{\alpha} W(\alpha) = \sum_{j=1}^{N} \alpha_j - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_{ij} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \quad (1)$$

$$\texttt{subject to } 0 \leq \alpha_i \leq C, i = 1, \ldots, N \quad (2)$$

$$\texttt{and } \sum_{i=1}^{N} y_i \alpha_i \quad (3)$$

The SVM constructs hyperplanes determined by Equation 4, where $\mathbf{z}_j$ is the Support Vector belonging to class y_j with the Lagrange multiplier being equal to α_j. K($\mathbf{x}, \mathbf{z}_j$) is the kernel function which introduces the implicit mapping between the input space and feature space.

$$f(\mathbf{x}) = \sum_{j=1}^{s} y_j \alpha_j K(\mathbf{x}, \mathbf{z}_j) + b \quad (4)$$

The choice of the Kernel Function is based on cross-validation methods and in this work a polynomial kernel of degree two is chosen.

In this work, we use SVM as a multiclass-classifier system which is trained on the different possible characters in Devanagari script. The Segmentation thus is a supervised classifier which classifies the input into one of the many classes. A scanning window is passed over the handwritten script and the resulting SVM output is obtained. The system, thus not only segments the handwritten script, it also can be used as a text recognition system.

III. Issues in Devanagari Script Segmentation

Devanagari script is an alphabetic script and has 11 vowels and 33 consonants along with 12 modifier symbols. It has its own specified composition rules for combining vowels, consonants and modifiers. A new modifier can be composed with the help of existing modifiers using the specified rules. An individual modifier and a composed modifier can be attached to a vowel or a consonant. The consonants and vowels can be written as an individual symbol in the word whereas a modifier has to be attached to a vowel or a consonant. The modifier symbols, known as *matras* are placed either on the top, at the bottom, to the left, to the right or a combination of these. The consonants may also have a half form. Devanagari script also has some characters which take the next character in their shadow because of their shape. Segmentation and Recognition in OCR Systems [4], though involved, is a much simpler job as compared to that in handwritten samples. Handwritten samples are much more complicated due to different personal styles of writing, which causes occluded characters and distorted representations. Figure 2 shows an OCR script and a handwritten script.

It is observed that a word can be broken into 3 stripes- a core strip, a bottom & a top strip. The top and bottom strips have only the modifiers whereas the core strip has composite characters comprising of half characters, characters and modifiers. A composite character may just be a character as well. The top and bottom strips may be empty for a word; just the top strip may be present or just the bottom. The top strip is separated from the core usually by the *shirorekha* - a horizontal line while the lower strip is below the core, though no regular feature separates the two. Thus it can be easily seen that a character can take the form of a composite character in many ways. These lead to a large number of character fusions (*conjuncts*) and character overlaps (either due to shape or due to modifiers). Even line segmentation is not an easy task as the lower strip of a line may overlap with the top strip of the other line. The characters are of varying heights and widths which make it inadequate to identify the composite characters based on their heights and widths alone. These issues are handled by a soft segmentation method using the standard BFS algorithm.

IV. Segmenting Handwritten Devanagari Script using SVM

In this work, segmentation & recognition are considered as inter-dependent problems, which can be efficiently handled using a robust classifier such as SVM. Handwriting recognition is conventionally handled using techniques of matching certain properties like curvature and template match. Support Vector Machines is an efficient classifier which transforms the problem into a higher-dimensional, so that the a linearly separable problem may be constructed. A multi-class SVM is used as implemented in Libsvm [6].

A. Preprocessing

Before the actual segmentation and recognition step, a number of preprocessing operations are carried out. Devanagari words usually have a *shirorekha* - a header line on the top of the core strip which separates it from the top strip. To segment a word into its constituting characters, the *shirorekha* has to be removed first. This is done by the following approaches:

1) The Image is filtered using various filtering algorithms.
2) The Text area is extracted from the document.
3) The general tilt in the text area is determined & taken care of by re-orienting the script.
4) Each individual line is separated. This is done by taking a side histogram of the area. The minima in the histogram marks the separation of lines as this region is devoid of any characters.
5) A side histogram of the line is then taken. The global peak in the top one-third portion is then taken to be the general position of the header line, and the portion above it is then cut off.

After the pre-processing steps, the input is free of the *shirorekha*. This was essential for the next step which identifies the broad clusters by doing a simple connectivity search.

B. Simple Connectivity Search

The input obtained for this step is a line free from *shirorekha*. A simple BFS is run on the input to find 8-connectivity in it. This utilizes the natural tendency of people to write two words separated by some gap. Further, if within a word two characters are separated by some space they are segmented in this step itself. It is to be noted that a vertical scan could not be made to obtain horizontal separation as it would lead to incorrect results in the case when the lower part of the left character is extended to the right so much that a part or whole of the right character lies over this extended portion of the left character.

For example, in the character *ra* followed by a vertical bar, the lower curve of *ra* may extend all the way below the vertical bar. Hence on a simple vertical scanning it would be identified as a single token. But through the 8-connectivity search, they are obtained as two separate tokens. Each token thus obtained is passed to the SVM-based classifier.

The BFS algorithm contains a parameter ignore-gap which specifies the gap to be ignored while doing the connectivity search. This eliminates the errors that may creep in if the writer has accidentally put some gap in a single character. Note that this gap will be small compared to the gap between two words in general. The tokens thus obtained are passed on to the second part where they are further segmented into individual characters using SVM.

C. Machine Training on the Dataset and Segmentation using SVM

A large dataset of Hindi characters (nearly 3000 at present) is used as training set for the SVM. Though there are forty-four characters in Hindi but based on frequency of occurrence & resemblance to some other more commonly used characters, 6 of them were discarded. Thus a total of 38 characters were there from which an input had to be identified. These different characters can be viewed as different classes. Thus 38 different classes were identified. Each image in dataset was mapped as a vector to a 1024 (32x32) dimensional space. The actual character was centralized in the image and proper padding was provided to standardize comparison. A vector-space was created comprising of these standard vectors. Each such vectors were obtained and a dataset of possible characters was obtained. This was then used to train the SVM with second-degree polynomial kernel.

D. Classification of Inputs into Classes

Each input token was scanned from right to left. The input tokens have been taken from right to left since in devanagri scipt, as there are some characters whose left parts match, for instance, in *ka* and *wa*, but there are no two characters whose rightmost parts match closely except from the vertical line. But even if the vertical line is common in the two characters, it's width is fairly small and if the character resembles the other for a fairly large number of scans then we can say with a high probability that the two characters are the same. A window scanner was created which took the snapshot of the area of the image it was placed upon. The snapshot obtained is processed to centralise the image in it. Each snapshot was then scaled and mapped as a vector to the 1024 dimension vector-space. Each vector is submitted to the *Support Vector Machine* which classifies it into a class in the vector-space. The class thus identified is then used to match standard data with the input. Thus a *minimum distance* is calculated in the vector-space. It is to be noted that the width of this window scanner has to be varied to identify a single character i.e. multiple snapshots of even a single character have to be taken.

This is done for the following reasons:

- The characters in hindi do not have a standard width, i.e., even in standard fonts the characters have different dimensions.
- At this moment(while taking snapshots), it is not known which class the current snapshot contains.
- If the width of the scan is fixed then it could result into following situations:
  - The width is more & the scan contains more than one (and may be a part of second) character.
  - The width is less & the scan contains only a part of a character.

  Both these situations lead to problems.

Hence a fixed width scanner would not work. So certain minimum and maximum character width - to - height ratios were identified based on a study of characters on the dataset (which is quite large and collected randomly & hence quite reliable). Typical ratios reached upon were 0.4 as minimum & 1.75 as maximum. It was observed that these ratios cover all the characters written normally by any normal person. Each of these varying width snapshots was taken keeping the right end of the window at same position and varying the width in the left direction. Next text area is again scanned with the same scanner but this time the scan is shifted a small distance to the left that is, compared to the previous scan the starting point of the scan this time is at a small distance to the left. This can be attributed to the fact that there is a notion of *matra* in Hindi. A character can have a vertical line at its right end signifying a *matra*. As we do not know beforehand whether the rightmost character is a pure Devanagari character or one with a *matra* on its right. So to account for the vertical line on the right of the character, we shift the scan a small distance to the left. We get snapshots in the same manner this time again.

For each snapshot of a particular width, a class was obtained from SVM and nearest distance by matching it against the standard images of that class. Thus there is an array of snapshots which contains at least one character. And for each snapshot, information obtained so far is its class number & minimum distance from the class.

The two important criteria for deciding the final character are:

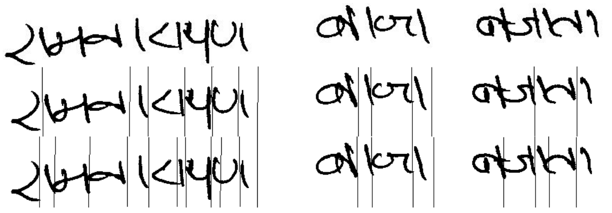
Fig. 3. Segmentation Results: The first image is the input to the BFS algorithm and the second image which is partially segmented is input to the SVM-based classifier which results in the third image.

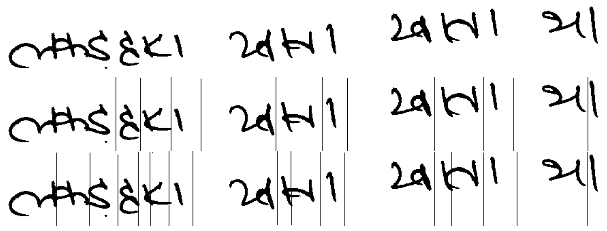
Fig. 4. Segmentation Results on another handwritten script.

- The number of points belonging to a particular class. More the number of points in a class more the probability of it being the correct character.
- The minimum distance from a class. It represents the degree to which a particular snapshot resembles a character. Less is the distance more the probability of it containing the correct character.

Proper weights are assigned to the number of points as well as to the minimum distance. But in order to assign the weights, the number of points in a class as well as the minimum distance has to be reduced to order 1. For the number of points belonging to a class this is done by finding the global maximum number & then dividing each number by this global maximum. To reduce the minimum distance to order 1 divide by the maximum distance of the minimums.

To each factor certain weights are assigned and the final weight of the particular snapshot is calculated. The snapshot which has the maximum weight is the correct character. The segmentation point is put just to the left of this image. The left end of the scanner window is placed on this newly segmented edge & the process is repeated. There are a number of times when a segmented token is obtained whose height or even width is far too less in comparison to the other characters of the handwritten text. Thus, if a segmented token is obtained whose width or height is less than the permissible threshold, it is merged with the token immediately to its right or left depending on which resulting token gives the best match with the standard dataset.

V. Results and Discussions

The handwriting segmentation system was run on many samples all gathered from different people selected randomly. Two typical sample inputs are shown in Figure 3 and 4. The Figures show input obtained after the pre-processing step & hence is free of *shirorekha*. The input for testing as it contains the cases where two different characters within a word are linked together, a single character is split in two, two different characters written so closely together that with a little less distance they might actually be read as a single character & so on.

VI. Conclusions

Handwriting recognition is a challenging field in the machine learning and this work identifies Support Vector Machines as a potential solution. In languages that do not have a very well defined writing pattern, as in the case of devanagri script, one cannot make use of the standard topology of the text. Results obtained using SVMs for recognition were more than 80% accurate and the inaccuracy that was there can be rectified by improving the database on which the machine was trained, by incorporating some more characters and by using some better heuristics for removing the headerline. The information from the segmentation part can be used to obtain better algorithms which can even serve to recognize the handwritten sample. As mentioned earlier, the segmentation method tells the class of each segment. We use a new concept *confusion distance* in further recognition. The *confusion distance* gives a measure of the physical similarity between two standard characters. For example, the *ka* & *pha* in devanagri script look pretty similar & hence the probability of a segment classified as *ka* actually being a *pha* will be more than the probability of its being a *ma*. Further, incorporating dictionary knowledge would further improve the recognition.

Acknowledgment

The authors would like to thank Mr. Abhaya Agarwal for his help in providing access to part of the Devanagari script database.

References

[1] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge: Cambridge University Press, 2000.

[2] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain”, *Psychological Review* vol. 65, pp. 386–408, 1958.

[3] V.N. Vapnik, *Statistical Learning Theory*, New York: John Wiley & Sons, 1998.

[4] V. Bansal and R. M. K. Sinha, “Integrating Knowledge Sources in Devanagri Text Recognition System,” *IEEE Trans. Systems, Man*, and *Cybernetics*- Part A, vol. 30, pp. 500–505,2000.

[5] R.M.K. Sinha and V. Bansal, “On Devanagari Document Processing”, *IEEE International Conference on Systems, Man and Cybernetics* , vol. 2, pp. 1621–1626, 1995

[6] C.C. Chang and C.J. Lin , “LIBSVM: a library for support vector machines”, 2001