*Sādhanā* Vol. 27, Part 1, February 2002, pp. 35–58. © Printed in India

# A font and size-independent OCR system for printed Kannada documents using support vector machines

T V ASHWIN* and P S SASTRY

Department of Electrical Engineering, Indian Institute of Science,
Bangalore 560 012, India
*Present address: Research Staff Member, IBM India Research Laboratories, I I T Campus, New Delhi 110 016, India
email: sastry@ee.iisc.ernet.in

**Abstract.** This paper describes an OCR system for printed text documents in Kannada, a South Indian language. The input to the system would be the scanned image of a page of text and the output is a machine editable file compatible with most typesetting software. The system first extracts words from the document image and then segments the words into sub-character level pieces. The segmentation algorithm is motivated by the structure of the script. We propose a novel set of features for the recognition problem which are computationally simple to extract. The final recognition is achieved by employing a number of 2-class classifiers based on the Support Vector Machine (SVM) method. The recognition is independent of the font and size of the printed text and the system is seen to deliver reasonable performance.

**Keywords.** OCR; pattern recognition; support vector machines; Kannada script.

## 1. Introduction

The objective of *Document Image Analysis* is to process the image of a printed page and render the information contained there into a form suitable for easy modification and manipulation on a computer. In general, document image analysis consists of two parts: *textual processing* and *graphical processing* (Gorman & Kasturi 1995).

Graphical processing is intended to handle the graphical parts such as figures, photographs etc. in the printed document while textual processing is for handling the text contained in the document.

In this paper we describe a document image analysis system that can handle printed text documents in Kannada, which is the official language of the south Indian state of Karnataka. The input to the system is the scanned image of a page of printed Kannada text. The output is an editable computer file containing the information in the printed page. The system is designed to be independent of the font and size of characters in the printed document and hence can be used with any kind of printed document in Kannada. The task of separating lines and words in the document is fairly independent of the script and hence can be achieved with standard techniques (O'Gorman & Kasturi 1995). However, due to the

peculiarities of the Kannada script, we make use of a novel segmentation scheme whereby words are first segmented to a sub-character level, the individual pieces are recognized and these are then put together to effect recognition of individual *aksharas* or characters. We use a novel low dimensional feature vector to characterize each segment and employ a classifier based on the recently developed concept of Support Vector Machines (SVM) (Burges 1998).

Currently there are many OCR systems available for handling printed English documents with reasonable levels of accuracy. (Such systems are also available for many European languages as well as some of the Asian languages such as Japanese, Chinese etc.) However, there are not many reported efforts at developing OCR systems for Indian languages. (See, e.g., Sinha and Mahabala 1979, Choudhury and Pal 1997, Bansal and Sinha 1999, Antani and Agnihotri 1999, for OCR systems for some of the Indian languages.) The work reported in this paper is motivated by the fact that there are no reported efforts at developing document analysis systems for the south Indian language of Kannada.

The rest of the paper is organized as follows. In § 2 we describe the Kannada script and bring out some of the special characteristics of the script from the point of view of developing an OCR system. We also motivate our method of segmenting words into pieces each of which represents only a part of a character or *akshara*. In § 3, we first provide an overview of our system and then describe some of the preprocessing steps needed to effect segmentation up to word level. Section 4 discusses the problem of segmenting words into recognizable units. Section 5 describes our feature detection and classification algorithms. In § 6 we present some results to illustrate the current capabilities of the system and conclude the paper in § 7.

## 2. The Kannada script

The Kannada alphabet is classified into two main categories: vowels and consonants. There are 16 vowels and 35 consonants as shown in figures 1 and 2.[1] Words in Kannada are composed of *aksharas* which are analogous to characters in an English word. While vowels and consonants are *aksharas*, the vast majority of *aksharas* are composed of combinations of these in a manner similar to most other Indian scripts.

An *akshara* can be one of the following,

(1) A stand alone vowel or a consonant (i.e. symbols appearing in figures 1 and 2).
(2) A consonant modified by a vowel.
(3) A consonant modified by one or more consonants and a vowel.

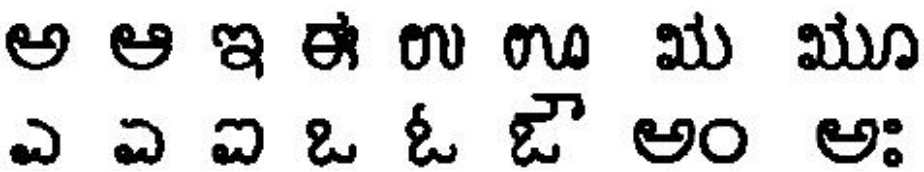

**Figure 1.** Vowels in Kannada.

[1]This list of Kannada alphabet includes a couple of characters (such as the eighth vowel and last consonant) which may not be included in the standard alphabet now-a-days

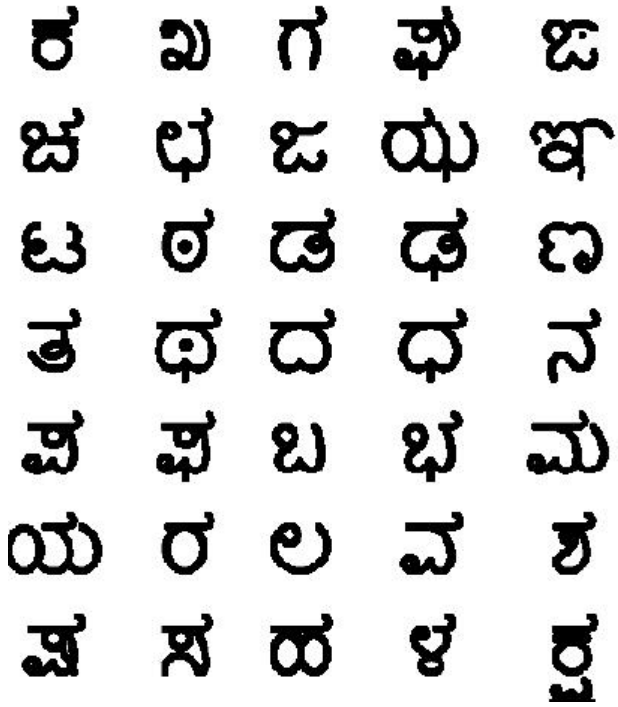

**Figure 2.** Consonants in Kannada.

When a vowel constitutes the whole *akshara*, the vowel normally appears at the beginning of a word. A consonant can also form the whole *akshara* and can come anywhere in the word. These *aksharas* appear in the middle region of the line and are represented by the same glyph as shown in figures 1 and 2.

A consonant *C* and a vowel *V* can combine to form an *akshara*. Here the *akshara* is composed by retaining most of the consonant glyph (cf. figure 2) and by attaching to it the glyph corresponding to the vowel modifier. The vowel modifier glyphs are different from those of the vowels and are shown in figure 3. The glyph of the vowel modifier for a particular vowel is attached to all consonants mostly in the same way, though, in a few cases the glyphs of the vowel modifier may change depending on the consonant. Figure 3 shows two of the consonants modified by all the 16 vowels. In this figure, the second row shows the vowels, the third row shows the glyphs of the vowel modifiers, the fourth and fifth rows show the consonant–vowel($C$–$V$) combinations for two consonants which phonetically correspond to *c* as in cat and *y* as in yacht. As can be seen from the figure, the vowel modifier glyphs attach to the consonant glyphs at up to three places corresponding to the top, right and bottom positions

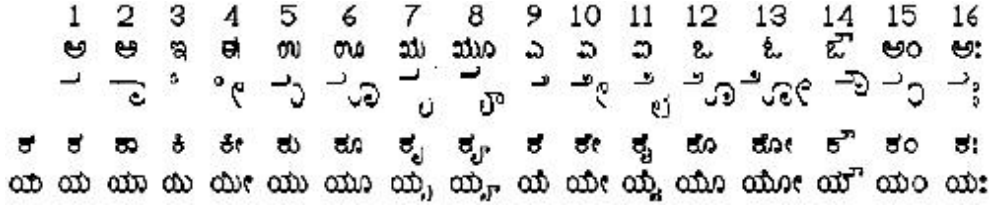

**Figure 3.** Example of consonant–vowel combinations.

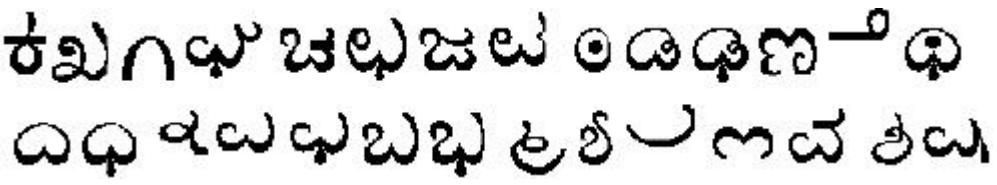

**Figure 4.** Consonant conjuncts in Kannada.

of the consonant. It can be observed that the widths of the $C{-}V$ combinations vary widely and also that the image of a single *akshara* may be composed of two or more disconnected components.

In the third form of *akshara* composition, consonants $C_1, C_2 \ldots C_j$ and a vowel $V$ can combine to form an *akshara*. In practice $j$ is limited to 3. The consonant $C_1$ forms the *base consonant* and the modifier for the vowel $V$ attaches to it. The rules for this consonant-vowel combination are the same as described above. The consonants $C_2 \ldots C_j$ are called the consonant conjuncts. The glyphs of many consonant conjuncts resemble those of the consonants though there are a sizeable number of exceptions. Some of the consonant conjunct glyphs are shown in figure 4. The consonant conjunct glyphs always appear below the $C{-}V$ combination formed by $C_1$ and $V$. A few examples showing *aksharas* formed by a consonant, a consonant conjunct and a vowel are shown in figure 5.

Thus *aksharas* in Kannada are formed by graphically combining symbols corresponding to consonants, consonant conjuncts and vowel modifiers using well-defined rules of combination. This general structure of forming characters of a word is a feature common to many other Indian scripts. It also necessitates some special ways of segmenting a word into its constituent symbols while designing OCRs for Kannada.

In a language like English (written in the standard Roman script), each word consists of a linear sequence of characters written next to each other in a line. There are only fifty two possible character symbols. Since there is always some space between characters of a word, a general strategy for handling such scripts would be to segment a word into individual characters and then recognize each character separately. However, such a strategy is not feasible for Kannada as explained below.

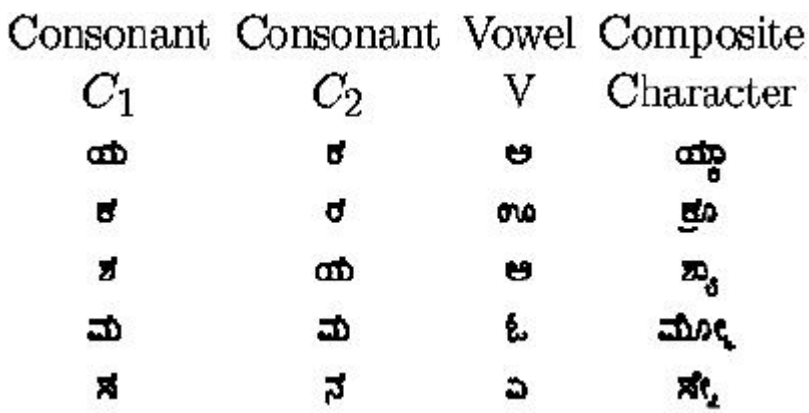


**Figure 5.** Some examples of consonant-consonant-vowel combinations.

As in English script, in Kannada also the glyphs of *aksharas* are placed next to each other; but the *aksharas* themselves are quite complicated with considerable variation in widths and heights (cf. figure 5). The number of possible consonant–vowel combinations are $35 \times 16 = 560$. The number of possible consonant–consonant–vowel combinations are $35 \times 35 \times 16 = 19600$. Thus, if we consider each *akshara* as a separate category to be recognized, building a classifier to handle these many classes is very difficult. Also, such an approach does not exploit the fact that the *aksharas* are formed through well-defined geometric combination of individual symbols. Many of the letters or *aksharas* are very similar and differ only in having an additional stroke or an attachment. Fortunately because of the structure of the script, it is feasible to break the *aksharas* into their constituents and recognize these components independently. This has been the chosen approach for most Indian scripts and is the chosen approach here.

As can be seen from figure 5, the image of an *akshara* may not be a single connected component. Hence correctly segmenting the image of a word into those corresponding to individual *aksharas*, prior to recognition, is very difficult. Hence, in our system we segment the word into components each of which can be only part of an *akshara*. Ideally, we may want to split the word so that each segment corresponds to the base consonant or vowel modifier or consonant conjunct. Even this is not generally feasible because some of the vowel modifiers themselves do not correspond to a single connected component. Further, due to the structure of some of the consonant and vowel symbols, it was observed that, for getting consistent segmentation, it is easier to allow even some of the consonant symbols to be split into two or more parts while segmenting a word. Once a word is split using our segmentation algorithm, we label each piece (using a pattern recognition technique) and then combine the labels on neighbouring segments to effect final recognition of *aksharas*. Due to the well-defined graphical combination rules of the script, this step of combining labels of individual segments into *aksharas* is fairly simple.

In most OCR systems the final recognition accuracy is always higher than the the raw character recognition accuracy. For obtaining higher recognition accuracy, language-specific information such as co-occurence frequencies of letters, a word corpus, a rudimentary model of the grammar etc. are used. This allows the system to automatically correct many of the errors made by the OCR subsystem. In our current implementation, we have not incorporated any such post-processing. The main reason is that, at present we do not have a word corpus for Kannada. Even with a word corpus the task is still difficult because of the highly inflexional nature of Kannada grammar. The grammar also allows for combinations of two or more words. Even though these follow well-defined rules of grammar, the number of rules is large and incorporating them into a good spell-checking application for Kannada is a challenging task. We plan to address these issues in our future work.

## 3. Preprocessing and identifying words

In our document analysis system, the sequence of processing steps is as follows. The page of text is scanned through a flat bed scanner at 300 DPI and binarised using a global threshold computed automatically based on the specific image.[2] This image is first processed to remove any skew so that the text lines are aligned horizontally in the image. The lines and words are then separated using the appropriate horizontal and vertical projections. The words are

[2]We shall refer to all the pixels (in the binary image) corresponding to characters as ON pixels and those corresponding to the background as OFF pixels

then segmented into smaller parts. To achieve this segmentation, the word is first split into three vertical zones based on the horizontal projection for the word. The three zones are then horizontally segmented using their vertical projections. These segments are then input to the recognizer. The recognizer extracts a set of features from the bitmaps of the segments. The feature vector is then classified using a SVM classifier. The labels for the segments from the three zones output by the classifier are then combined and a ASCII text containing the transliterated version of recognized Kannada text is output. This ASCII file can then be uploaded into a Kannada typesetting package for viewing and editing.

In this section we briefly outline the methods used to process the image up to word segmentation. In the next two sections we describe the word segmentation and feature extraction and classification.

For skew correction we employed a windowed Hough transform technique (Gorman & Kasturi 1995). Some of the Indian scripts, such as Devanagari, have a dark top line linking all the characters in a word. This strong linear feature can be exploited for skew estimation, for example, using a projection profile based method (Gorman & Kasturi 1995). In Kannada, such a line linking all characters of the word is not present. However, a short horizontal line can usually be seen near the top of most of the characters. Hence the Hough transform technique for extracting lines works well.

Simple methods for separating lines and words are those based on projection profiles (Gorman & Kasturi 1995). A projection profile is a histogram giving the number of ON pixels accumulated along parallel lines. Thus a horizontal projection profile is a one-dimensional array where each element denotes the number of ON pixels along a row in the image. Similarly a vertical projection profile gives the column sums. It is easy to see that one can separate lines by looking for minima in horizontal projection profile of the page and then one can separate words by looking at minima in vertical projection profile of a single line. We have used such projection profile based methods for line and word segmentation.

The line segmentation poses some problems due to the fact that the consonant conjuncts, which appear below the base consonant, appear frequently disconnected from the base consonant which results in a false white space in the horizontal projection. Also, overlapping of the consonant conjuncts of one line with the vowel modifiers which appear towards the top of the next line can mask some of the minima that should have been seen in horizontal projection. To overcome these problems we first extract the minima of the horizontal projection profile smoothed by a simple moving average filter. The line breaks so obtained sometimes segment inside a line. Such false breaks are removed by using statistics of the line widths and the separation between lines. While segmenting the lines into words, it is to be noted that the inter-word spacing is not uniform over all the lines because the inter-word spacing is adjusted to align the text correctly at its ends. Also, by the nature of the script, sometimes inter-word gaps are as small as some of the inter-character gaps. A threshold to separate inter-word gaps from inter-character gaps is determined adaptively for each line of text. The threshold is obtained by analyzing the histogram of the widths of the gaps in a line. Using this threshold and the vertical projection profile of a line, we segment the line into its constituent words.

## 4. Segmenting words into smaller components

As described in §2, the letters in Kannada are composed by attaching to the glyph of a consonant the glyphs of the vowel modifiers and the glyphs of the consonant conjuncts. Due to the large number of letter combinations possible, building a classifier to recognize

a whole letter is very difficult. Therefore in our implementation we segment the letters into its constituents, i.e. the base consonant, the vowel modifier and the consonant conjunct. Our segmentation strategy is based on the following observations.

- The Kannada *akshara* shows three distinct vertical regions.
- The top region, which ends just below the short head line of the consonant symbol, contains the vowel modifiers and sometimes parts of the base consonant. Some letters may not have a head line, and, in that case, the location of the top zone of neighbouring letters can be used.
- The middle region contains the consonant glyphs and some vowel modifiers. The vowel modifier glyphs may appear as either connected or disconnected components to the right of the base consonant. The middle region ends where the base consonants end.
- The bottom region which extends below the base consonant consists of glyphs for the consonant conjuncts and the glyphs for some vowel modifiers. These glyphs generally appear disconnected from the base consonant and the vowel modifiers present in the middle zone.

The words are first vertically segmented into three zones. This segmentation is achieved by analyzing the horizontal projection profile of a word. A typical horizontal projection is shown in figure 6. Points (2) and (4) denote the locations corresponding to the ending of the top and middle zones respectively. They are identified by using the observation that the drops in the histogram between (1) and (2), and (3) and (4) (cf. figure 6) are two of the biggest drops in the histogram. As can be seen from the figure, separating the middle zone from the bottom zone is easier because of the fact that the consonant conjuncts are always disconnected from the base consonant. Separating the top zone from the middle zone is more difficult. While our heuristics work most of the time, there are a few situations where the top zone as segmented may contain some of the base consonant or the middle zone may contain a little bit of the top vowel modifier. However, many of these inaccuracies can be (and are) taken care of by training the pattern classifier properly.

The next task is to segment the three zones horizontally. The middle zone is the most critical since it contains a major portion of the letter. The middle zone is therefore the first to be segmented. As described earlier, in the middle zone we find the consonants and some of the vowel modifiers. Our aim is to separate the vowel modifier from the consonant. Achieving this exactly, prior to recognition, is very difficult. Therefore we allow certain base consonants to also be segmented into two or more parts. To achieve this segmentation we follow an oversegment-and-merge approach. The middle zone is first oversegmented by extracting points in the vertical projection (of the middle region) showing drops in the histogram value exceeding a fixed threshold. The threshold is kept low so that a large number of seg-


**Figure 6.** A word with its horizontal projection and vertical segmentation into three zones.

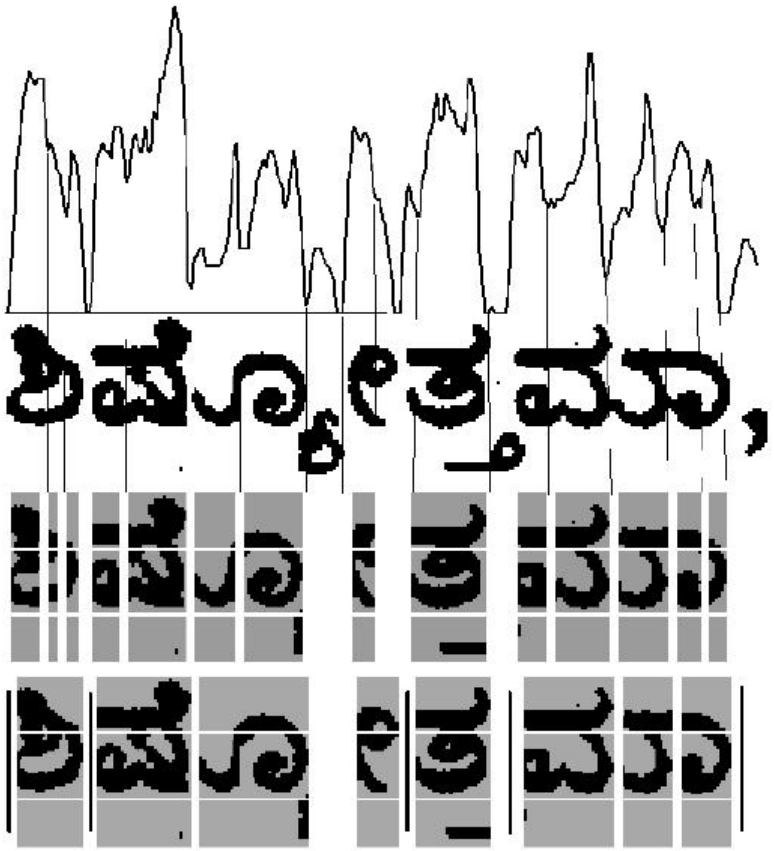

**Figure 7.** Segmentation in middle zone. (i) Vertical projection for a word, (ii) the word being considered, (iii) vertical segmentation and horizontal over segmentation, (iv) after merging of middle zone segments. The thick vertical lines show the locations where *aksharas* actually end. (To be able to draw these lines and also to show some gap between segments, part of the original image is erased in this figure. However, all that part is seen by the classifier.)

ments are obtained. This segmentation does not give consistent segments (i.e. the segments obtained may differ depending on the font and size of the letters) and also gives a large number of small segments. These segments are merged using two different strategies. In the first approach the region around a segment break is analysed and features of the boundary contours around the segment break points along with connectivity information were used to validate segment breaks. We shall refer to this as the heuristic merging algorithm. The second approach is a recognition-based approach. Here, for every pair of segments, the classification confidences of the classifier for the first segment in the pair and for the segment obtained by merging the pair are obtained. If the merged segment shows higher confidence, the segments are merged. Since we use an SVM classifier, the magnitude of the output of the SVM gives a good measure of the classification confidence. Both approaches have resulted in reasonable performance.

Figure 7 illustrates our segmenting algorithm. Here the word contains four *aksharas* which are shown separated by thick vertical lines in the bottom row of the figure. (The four *aksharas* can be roughly transliterated into English as: *shi shyo tta maa*.) It is easy to see from the figure

that it is nearly impossible to segment *aksharas* prior to recognition. It may also be noted that the second *akshara* is more than four times as wide as the first one. In the figure, the third row shows the oversegmentation of the word and the fourth row is the final segmentation obtained using our heuristic merging algorithm.

## 5. Design and implementation of the classifier

After segmenting the document as described in the previous two sections, we need to recognize each of the segments to effect final recognition. For this we use a pattern classifier to label each of the segments in a word with some category. Since each *akshara* consists of many segments, arranged in three vertical zones, we need to put together the labels for segments output by the classifier into *aksharas*. Achieving this is not very difficult because the *aksharas* are composed through well-defined rules of graphical composition. Since the processing involved here is fairly straight forward, we will not describe it in this paper.

Each of our segments contains a vowel modifier or a consonant conjunct or a base consonant (or a part of a base consonant). We use three different classifiers for classifying the segments in each of the three vertical zones. Thus the possible classes for the segments in the top two zones would be the vowels, base consonants or vowel modifiers. Hence the total number of classes here should be $16 + 35 + 16 = 67$. The bottom zone contains mostly consonant conjuncts and hence we should have another 35 classes. However, because of our segmentation strategy, some of the base consonants give rise to more than one segment which tends to increase the total number of classes. In our final implementation, it turned out that we had 106 different classes. We have used 106 two-class classifiers (each meant to decide whether or not a segment belongs to a specific class).

To train all the classifiers we need to generate a training set of patterns which is done as follows. We scanned many pages of Kannada text from different magazines, textbooks, pamphlets, newspapers etc. These images are then segmented using the algorithms described earlier. Some of the segments so obtained were then hand-labelled and added to the training set. To make this process user-friendly and also to have the capability of adding training patterns at anytime, an interactive tool was developed. Using this, the user can view any segmented document and then click on any segment to add to training set. All results reported in this paper are those obtained with a training set of about 3000 patterns. (The specific number of training patterns used for classifiers in different zones are given in § 6.)

### 5.1 *Feature extraction*

Features are a set of numbers that capture the salient characteristics of the segment image. Since we want to achieve font and size-independent recognition, we cannot rely on direct template matching. There are many different features proposed for character recognition (Trier *et al* 1996). One can consider features based on the image bitmap such as projection of the image function onto different lines, moments of the image etc. One can split the image into different zones and then extract simple features from each of the zones (Bosker 1992). One can also use features obtained through various transforms such as cosine transform, Zernike moments etc. In these cases, the image function is approximated by using a set of basis functions and the coefficients are taken to be the features. Another approach is to use features based on the strokes that make up the different letters. Here one obtains the skeletonised binary image that gives the contours of the letters. Then various ways of characterizing the

**Figure 8.** Division of segments into tracks and sectors.

contour give rise to different features (see, e.g., Pavlidis 1986, Sekita *et al* 1988, Lu *et al* 91, Lee & Chen 1992).

We have used a set of features obtained by splitting each segment image into a number of zones. Characters in Kannada have a rounded appearance. Therefore features which can extract the distribution of the ON pixels in the radial and the angular directions will be effective in capturing the shapes of the characters.

To extract the features, the segment bitmap is first area normalized so that the number of ON pixels in all the normalized bitmaps are equal. This helps in making the classifier immune to size changes in the characters. The normalized bitmap is then divided into smaller zones by using tracks and sectors as shown in figure 8. The tracks are the annular regions centred at the centroid of the ON pixels of the bitmap. The sectors are formed by drawing lines from the centroid at different angles. The number of ON pixels in each zone is then counted and the counts in all the zones are composed into a feature vector. (It may be noted that, since the image bitmap is rectangular, the final track has a rectangular boundary rather than a circular boundary). As can be seen from figure 8, there are 3 tracks and 16 sectors resulting in a feature vector of dimension 48. All results reported in this paper are obtained with this 48-dimensional feature vector. We have also experimented with a variant of these features. Instead of keeping all tracks to be of equal radial thickness, we can choose tracks with varying radial thickness so that the number of ON pixels in all tracks are approximately equal. This is achieved as follows: the bitmap is first split into a large number of tracks and the number of ON pixels in these tracks is counted. Adjacent tracks are merged, ensuring that the number of ON pixels in the tracks are approximately equal, till the desired number of tracks is obtained. This gave improvements in recognition rates for most cases.

For studying the effectiveness of our features, we have also experimented with the feature vector of Zernike moments as a benchmark. Zernike moments are the projections of the image onto the Zernike polynomials. The Zernike polynomials are a set of complex polynomials which form a complete orthogonal set over the interior of the unit circle. The image function is first scaled to lie inside the unit circle, Then Zernike moments corresponding to the projection of the image onto the set of Zernike polynomials are obtained. We used Zernike moments of up to $12^{th}$ order. It was empirically observed that the representation of the segment images in terms of Zernike polynomials, truncated at $12^{th}$ order terms, was sufficient to reconstruct the original images accurately. The details regarding extraction of Zernike moments can be found elsewhere (Ashwin 2000).

It is seen that the recognition accuracy obtained with our 48-dimensional feature vector is at least as good and often better than that obtained using the Zernike features. Computing the Zernike moments takes more than eight times as much computational time as computing our features (see discussion in § 6).

5.2 *Pattern classification*

The feature vector extracted from the segment bitmap has to be assigned a label using a pattern classifier. There are many methods for designing pattern classifiers such as Bayes classifier based on density estimation, using neural networks, linear discriminant functions, nearest neighbour classification based on prototypes etc. In this system we have used the Support Vector Machine (SVM) classifier. SVMs represent a new pattern classification method which grew out of some of the recent work in statistical learning theory (Vapnik 1995, 1999). The solution offered by SVM methodology for the two class pattern recognition problem is theoretically elegant, computationally efficient and is often found to give better performance by way of improved generalizations. In the next subsection we provide a brief overview of SVMs. A tutorial introduction to SVMs is given by Burges (1998).

5.2a *SVM classifier:* The SVM classifier is a two-class classifier based on the use of discriminant functions. A discriminant function represents a surface which separates the patterns so that the patterns from the two classes lie on the opposite sides of the surface. The SVM is essentially a separating surface which is optimal according to a criterion as explained below.

Consider a two-class problem where the class labels are denoted by $+1$ and $-1$. Given a set of $l$ labelled (training) patterns, $\mathcal{X} = \{(\mathbf{x_i}, y_i), 1 \leq i \leq l\}, \mathbf{x_i} \in \Re^d, y_i \in \{-1, +1\}$, the hyper-plane represented by $(\mathbf{w}, b)$ where $\mathbf{w} \in \Re^d$-represents the normal to the hyper-plane and $b \in \Re$ the offset, forms a *separating hyper-plane* or a *linear discriminant function* if the following separability conditions are satisfied.

$$\begin{aligned} \mathbf{w}^{\mathrm{t}}\mathbf{x}_i + b > 0, \quad & \text{for } i : y_i = +1, \\ \mathbf{w}^{\mathrm{t}}\mathbf{x}_i + b < 0, \quad & \text{for } i : y_i = -1. \end{aligned} \tag{1}$$

Here, $\mathbf{w}^{\mathrm{t}}\mathbf{x}_i$ denotes the inner product between the two vectors, and $g(\mathbf{x}) = \mathbf{w}^{\mathrm{t}}\mathbf{x} + b$ is the linear discriminant function.

In general, the set $\mathcal{X}$ may not be linearly separable. In such a case one can employ the *generalized linear discriminant function* defined by,

$$g(\mathbf{x}) = \mathbf{w}^{\mathrm{t}}\phi(\mathbf{x}) + b \quad \text{where } \phi : \Re^d \to \Re^{d'}, \mathbf{w} \in \Re^{d'}. \tag{2}$$

The original feature vector $\mathbf{x}$ is $d$-dimensional. The function $\phi$ represents some nonlinear transformation of the original feature space and $\phi(\mathbf{x})$ is $d'$-dimensional. (Normally we would have $d'$ much larger than $d$.) By proper choice of the function $\phi$ one can obtain complicated separating surfaces in the original feature space. For any choice of $\phi$, the function $g$ given by (2) is a linear discriminant function in $\Re^{d'}$, the range space of $\phi$. However, this by itself does not necessarily mean that one can (efficiently) learn arbitrary separating surfaces using only techniques of linear discriminant functions by this trick of using $\phi$. A good class of discriminant functions (say, polynomials of degree $p$) in the original feature space may need a very high dimensional (of the order of $d^p$) vector, $\phi(\mathbf{x})$, and thus $d'$ can become much larger than $d$. This would mean that the resulting problem of learning a linear discriminant function in the $d'$-dimensional space can be very expensive both in terms of computation and memory. Another related problem is that we need to learn the $d'$-dimensional vector $\mathbf{w}$ and hence we would expect that we need a correspondingly larger number of training samples as well. The methodology of SVMs represents an efficient way

of tackling both these issues. Here we only explain the computational issues. (For a discussion on why use of SVMs does not demand larger training sets and other related issues of the generalization abilities of SVMs, the reader is referred to Burges 1998 and Vapnik 1999.)

Let $\mathbf{z}_i = \phi(\mathbf{x}_i)$. Thus now we have a training sample $\{(\mathbf{z}_i, y_i)\}$ to learn a separating hyper-plane in $\Re^{d'}$. The separability conditions are given by (1) with $\mathbf{x}_i$ replaced by $\mathbf{z}_i$. Since there are only finitely many samples, given any $\mathbf{w} \in \Re^{d'}$, $b \in \Re$, that satisfy (1), by scaling them as needed, we can find $\mathbf{w}$, $b$, that satisfy

$$y_i[\mathbf{w}^t\mathbf{z}_i + b] \geq 1, \ i = 1, \ldots, l. \tag{3}$$

Note that we have made clever use of the fact that $y_i \in \{+1, -1\}$ while writing the separability constraints as above. The $\mathbf{w}$, $b$, that satisfy (3) define a separating hyper-plane, $\mathbf{w}^t\mathbf{z} + b = 0$, such that there are no training patterns between the two parallel hyper-planes given by $\mathbf{w}^t\mathbf{z} + b = +1$, and $\mathbf{w}^t\mathbf{z} + b = -1$, The distance between these two parallel hyper-planes is $2/||\mathbf{w}||$, which is called the margin (of separation) of this separating hyper-plane. It is intuitively clear that among all separating hyper-planes the ones with higher margin are likely to be better at generalization. The SVM is, by definition, the separating hyper-plane with maximum margin.

Hence, the problem of obtaining the SVM can be formulated as an optimization problem of obtaining $\mathbf{w} \in \Re^{d'}$ and $b \in \Re$, to

$$\text{Minimize: } \frac{1}{2}||\mathbf{w}||^2 \tag{4}$$

$$\text{Subject to: } 1 - y_i(\mathbf{z}_i^t\mathbf{w} + b) \leq 0 \quad i = 1, \ldots, l. \tag{5}$$

Suppose $\mathbf{w}^*$ and $b^*$ represent the optimal solution to the above problem. Using the standard Lagrange multipliers technique, one can show that

$$\mathbf{w}^* = \sum_{i=1}^{l} \alpha_i^* y_i \mathbf{z}_i, \tag{6}$$

where $\alpha_i^*$ are the optimal Lagrange multipliers. There would be as many Lagrange multipliers as there are constraints and there is one constraint for each training pattern (cf. (5) ). From standard results in optimization theory, we must have $\alpha_i^*[1 - y_i(\mathbf{z}_i^t\mathbf{w}^* + b^*)] = 0, \ \forall i$. Thus $\alpha_i^* = 0$ for all $i$ such that the separability constraint (5) is satisfied by strict inequality. Define a set of indices,

$$\mathcal{S} = \{i \ : \ y_i(\mathbf{z}_i^t\mathbf{w}^* + b^*) - 1 = 0, \ 1 \leq i \leq l\}. \tag{7}$$

Now it is clear that $\alpha_i^* = 0$ if $i \notin \mathcal{S}$. Hence we can rewrite (6) as

$$\mathbf{w}^* = \sum_{i \in \mathcal{S}} \alpha_i^* y_i \mathbf{z}_i, \tag{8}$$

The set of patterns $\{\mathbf{z}_i : i \ s.t. \ \alpha_i^* > 0\}$ are called the *support vectors*. From (8), it is clear that the $\mathbf{w}^*$ is a linear combination of support vectors and hence the name SVM for the classifier. The support vectors are those patterns which are closest to the hyper-plane and are sufficient

to completely define the optimal hyper-plane.[3] Hence these patterns can be considered to be the most important training examples.

To learn the SVM all we need are the optimal Lagrange multipliers corresponding the problem given by (4) and (5). This can be done efficiently by solving its dual which is the optimization problem given by: Find $\alpha_i,\ i = 1, \ldots, l$, to

$$\text{Maximize} : \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{z}_i^{\mathrm{t}} \mathbf{z}_j,$$

$$\text{Subject to} : \alpha_i \geq 0, i = 1, 2, \ldots, l,$$

$$\sum_{i=1}^{l} \alpha_i y_i = 0. \tag{9}$$

By solving this problem we obtain $\alpha_i^*$ and using these we get $\mathbf{w}^*$ and $b^*$. It may be noted that the dual given by (9) is a quadratic optimization problem of dimension $l$ (recall that $l$ is the number of training patterns) with one equality constraint and nonnegativity constraints on the variables. This is so irrespective of how complicated the function $\phi$ is. Once the SVM is obtained, the classification of any new feature vector, $\mathbf{x}$, is based on the sign of (recall that $\mathbf{z} = \phi(\mathbf{x})$)

$$f(\mathbf{x}) = \phi(\mathbf{x})^{\mathrm{t}} \mathbf{w}^* + b^* = \sum_{i \in \mathcal{S}} \alpha_i^* y_i \phi(\mathbf{x}_i)^{\mathrm{t}} \phi(\mathbf{x}) + b^*, \tag{10}$$

where we have used (8). Thus, both while solving the optimization problem (given by (9)) and while classifying a new pattern, the only way the training pattern vectors, $\mathbf{x}_i$ come into picture are as inner products $\phi(\mathbf{x}_i)^{\mathrm{t}} \phi(\mathbf{x}_j)$. This is the only way, $\phi$ also enters into the picture. Suppose we have a function, $K : \Re^d \times \Re^d \to \Re$ such that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^{\mathrm{t}} \phi(\mathbf{x}_j)$. Such a function is called a Kernel function. Now we can replace $\mathbf{z}_i^{\mathrm{t}} \mathbf{z}_j$ in (9) by $K(\mathbf{x}_i, \mathbf{x}_j)$. Then we never need get into $\Re^{d'}$ while solving the dual. Often we can choose the kernel function so that it is computationally much simpler than computing inner products in $\Re^{d'}$. Once (9) is solved and $\alpha_i^*$ are obtained, during classification also we never need enter $\Re^{d'}$. We can calculate the needed value of $f$ defined by (10) once again by using the kernel function.

Given any symmetric function $K : \Re^d \times \Re^d \to \Re$, there are some sufficient conditions, called Mercer's conditions (Burges 1998), to ensure that there is some function $\phi$ such that $K$ gives the inner product in the transformed space. Some of the Kernels used in SVMs are listed in table 1. The polynomial Kernel results in a separating surface in $\Re^d$ represented by a polynomial of degree $p$. With the Gaussian Kernel, the underlying $\phi$ is such that $\phi(\mathbf{x})$ is infinite dimensional! However, by the trick of Kernel functions, we can get such arbitrarily complicated separating surfaces by solving only a quadratic optimization problem given by (9). The SVM with a Gaussian Kernel is equivalent to a radial basis function neural network and SVM with perceptron kernel is equivalent to a three-layer feedforward neural network with sigmoidal activations. In both cases the learning problem for SVM (namely, the optimization problem given by (9)) is much simpler computationally (Burges 1998).

[3] It may be noted that once $\mathbf{w}^*$ is determined using (6), it is easy to determine $b^*$. By definition, if $\mathbf{z}_i$ is any support vector, then, $b^* = y_i - \mathbf{z}_i^{\mathrm{t}} \mathbf{w}^*$. In practice $b^*$ is taken to be the average of values obtained like this from all support vectors

**Table 1.** Some popular kernels for SVMs.

| Type of kernel | $K(x_i, x_j)$ | Comments |
|---|---|---|
| Polynomial kernel | $(x_i^{\mathrm{t}} x_j + 1)^p$ | Power $p$ is specified *a priori* by the user |
| Gaussian kernel | $\exp\left(-\frac{1}{2\sigma^2}\|x_i - x_j\|^2\right)$ | The width $\sigma^2$, common to all the kernels, is specified *a priori* |
| Perceptron kernel | $\tanh(\beta_0 x_i^{\mathrm{t}} x_j + \beta_1)$ | Mercer's condition satisfied only for certain values of $\beta_0$ and $\beta_1$ |

The optimization problem given by (9) has a lot of interesting structure and hence there are available many efficient algorithms for solving it (Joachims 1999a; Platt 1999; Mangasarian & Musicant 1999; Keerthi *et al* 2000).

So far, we have assumed that the optimization problem specified by (4)–(5), whose dual is given by (9), has a solution. This is so only if in the $d'$-dimensional $\phi$-space, the (transformed) pattern vectors are linearly separable. In general, this is difficult to guarantee. To overcome this, we can change the optimization problem to

$$\text{Minimize: } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi_i, \tag{11}$$

$$\text{Subject to: } 1 - y_i(\mathbf{z}_i^{\mathrm{t}}\mathbf{w} + b) - \xi_i \leq 0 \quad i = 1, \ldots, l.$$

$$\xi_i \geq 0, \quad i = 1, \ldots, l. \tag{12}$$

Here $\xi_i$ can be thought of as penalties for violating separability constraints. Now these are also variables over which optimization is to be performed. The constant $C$ is a user specified parameter of the algorithm and as $C \to \infty$ we get the old problem. It so turns out that the dual of this problem is same as (9) except that the nonnegativity constraint on $\alpha_i$ is replaced by $0 \leq \alpha_i \leq C$. The optimal values of the new variables $\xi_i$ are irrelevant to the final SVM solution.

To sum up, the SVM method for learning two class classifiers is as follows. We choose a Kernel function and some value for the constant $C$ in (11). Then we solve its dual which is same as (9) except that the variables $\alpha_i$ also have an upper bound, namely, $C$. (It may be noted that here we use $K(\mathbf{x}_i, \mathbf{x}_j)$ in place of $\mathbf{z}_i^{\mathrm{t}}\mathbf{z}_j$ in (9)).Once we solve this problem, all we need to store are the non-zero $\alpha_i^*$ and the corresponding $\mathbf{x}_i$ (which are the support vectors). Using these, given any new feature vector $\mathbf{x}$, we can calculate the *output* of SVM, namely, $f(\mathbf{x})$ through (10). The classification of $\mathbf{x}$ would be $+1$ if the output of SVM is positive; otherwise it is $-1$.

5.2b *SVM classifiers for OCR:* We have used SVM classifiers for labelling each segment of a word. As explained earlier, we have trained a number of two-class classifiers (SVMs), each one for distinguishing one class from all others. Thus each of our class labels has an associated SVM. A test example is assigned the label of the class whose SVM gives the largest positive output. If no SVM gives a positive output then the example is rejected. The output of the SVM gives a measure of the distance of the example from the separating hyper-plane in the $\phi$ space. Hence higher the value of the (positive) output for a given pattern higher is the confidence in classifying the pattern.

We have used Gaussian kernel function for all SVMs and used a single value for the penalty constant $C$. The SVMs are trained using the SVM$^{light}$ package (Joachims 1999). This is one of the efficient algorithms available for solving (9) or the dual of the problem specified by (11) and (12).

We have also experimented with hierarchical classifiers. Since each of our classifiers are meant to distinguish one against all other classes, we get somewhat poor generalization because of confusions among very similar looking characters. Often this is overcome by grouping of 'similar' classes. In our system we have used such hierarchical classifiers for the middle zone segments in the words. We first trained SVM classifiers, one for each class. Then looking at the *confusion matrix*, we grouped some of the classes together. Now we retrained a two-level hierarchical classifier. The first level classifier categorizes the input into one of the groups of classes. Then the second level classifier (there is one such classifier for each group) distinguishes between the classes in that group. We have used SVM classifiers at each level and used the same features for both levels of the hierarchy.

### 5.3 *Recognized output*

The sequence of segments corresponding to a given *akshara* is not necessarily unique. This is because, during the segmentation some base consonants may be split into different number of segments for different fonts. (It may be noted that the system has no information about the font etc.) However, all the possible modes of splitting of the *aksharas* is known and this information is encoded in a lexicon. The sequence of labels output from the classifier is checked against this lexicon for final recognition of all the *aksharas*. If an inconsistency is observed an attempt is made to remove this inconsistency by replacing the best label for the current segment by the second best label. If the inconsistency could not be removed by this, then that piece is labelled as unrecognizable.

The output after classification has to be transformed into a format which can be loaded into a Kannada editing package. The method of composition of *aksharas* in all Kannada typesetting packages is similar. The input is usually ASCII text in which the *aksharas* are encoded as ASCII strings. The string representing an *akshara* is composed from the ASCII character codes corresponding to the different components of the *akshara* as follows: the codes for the base consonant appear first followed by the codes for the consonant conjuncts; the codes for the vowel modifier appear at the last and signify the end of an *akshara*. The ASCII codes are such that the final text looks like transliteration of the Kannada word into English. In our system we currently output a file which is compatible with the *Kantex* typesetting software (Jagadeesh & Gopinath 2000). *Kantex* is essentially a collection of fonts and macros to typeset Kannada using LaTeX typesetting tools.

## 6. Results

The sequence of operations in our system to process a document is as follows. A page of *Kannada* text is scanned through a flat bed scanner at 300DPI. The image format used throughout is the PGM raw format. The first step is *skew correction*. We used the *Windowed Hough Transform* using a window size that roughly corresponds to 100000 pixels. If we assume a 50% pixel occupancy in each scan line and if the height of text line is 100 pixels, then this window corresponds to roughly 2 text lines which should be sufficient for a good enough skew estimate. Except for separating out lines etc., we have not implemented any other page layout analysis. (That is why the final outputs shown in this section do not show

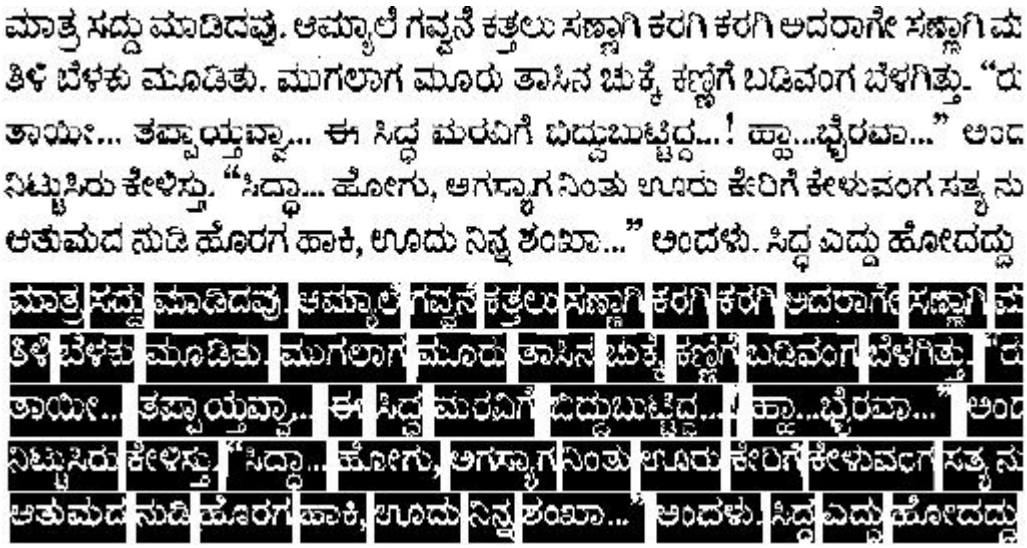

**Figure 9.** Example of word segmentation, it can be seen that the interword spacing varies widely across lines, and is also almost indistinguishable from the intercharacter gaps. The algorithm identifies all words correctly.

even paragraph breaks.) We expect that any of the standard page layout analysis schemes used for English will work equally well with *Kannada* for tasks such as recognizing headings, titles etc.

The skew corrected image is then input to the segmentation program which separates lines, words and then segments words into smaller parts as explained in §§ 2 and 3. We first separate lines and then words. An example of word segmentation is shown in figure 9. The words are then segmented into three vertical zones and each of the vertical zones are then horizontally segmented. As explained in § 3, we first oversegment the word and then merge the segments. An example of an over-segmented page of text is shown in figure 10a. The final results of segmentation using the two merging strategies are shown in figures 10b and c. For this figure, the document image is obtained by combining subimages from many scanned pages. This is done so as to be able to illustrate the font and size independence of our system in a single figure.

In the final step we need to label each of the segments using our classifier and then effect final recognition of the *aksharas* based on these labels. Figure 11 shows an example of the final output of our system.

Before presenting the final recognition accuracies obtained on some test data with our system, we briefly summarize the various options available at the classification stage.

As explained in § 3, we experimented with three types of features. Two feature sets are based on dividing the image into radial tracks and sectors. The two feature sets are distinguished by whether the radial tracks are equally spaced or adaptively spaced to achieve roughly equal number of ON pixels in each annular region. We refer to these as *structural features* and *modified structural features* respectively. In both cases we have used 3 tracks and 16 sectors so that we have a feature vector of dimension 48. We have also experimented with Zernike moments as the features.

We have used the SVM classifier for recognizing the segments in all zones. To train the SVM classifier we used the SVM$^{light}$ (Joachims 1999b) package. As explained in § 5.2b, we

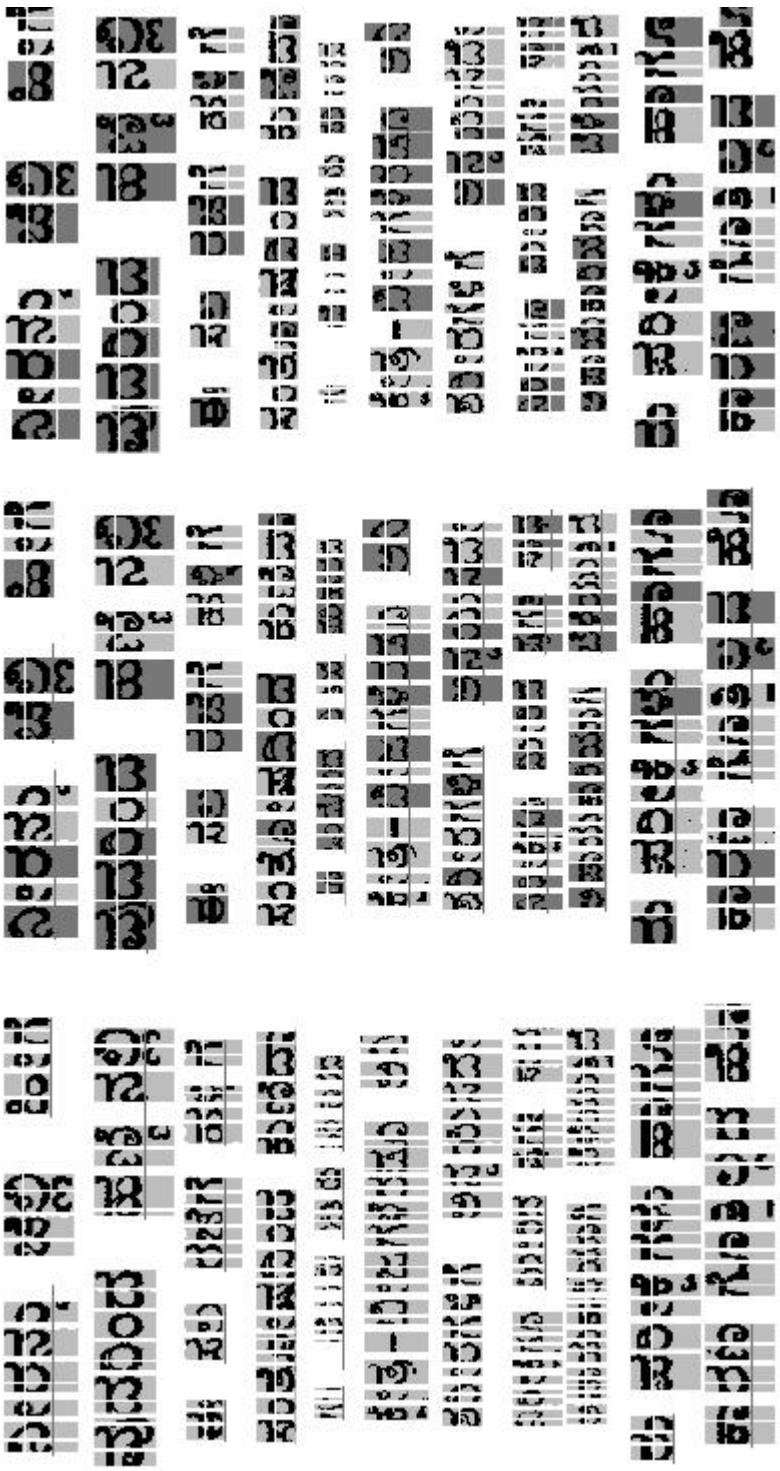

**Figure 10.** Example of character segmentation, **(a)** segments against a darker background in **(b)** and **(c)** are merged segments.

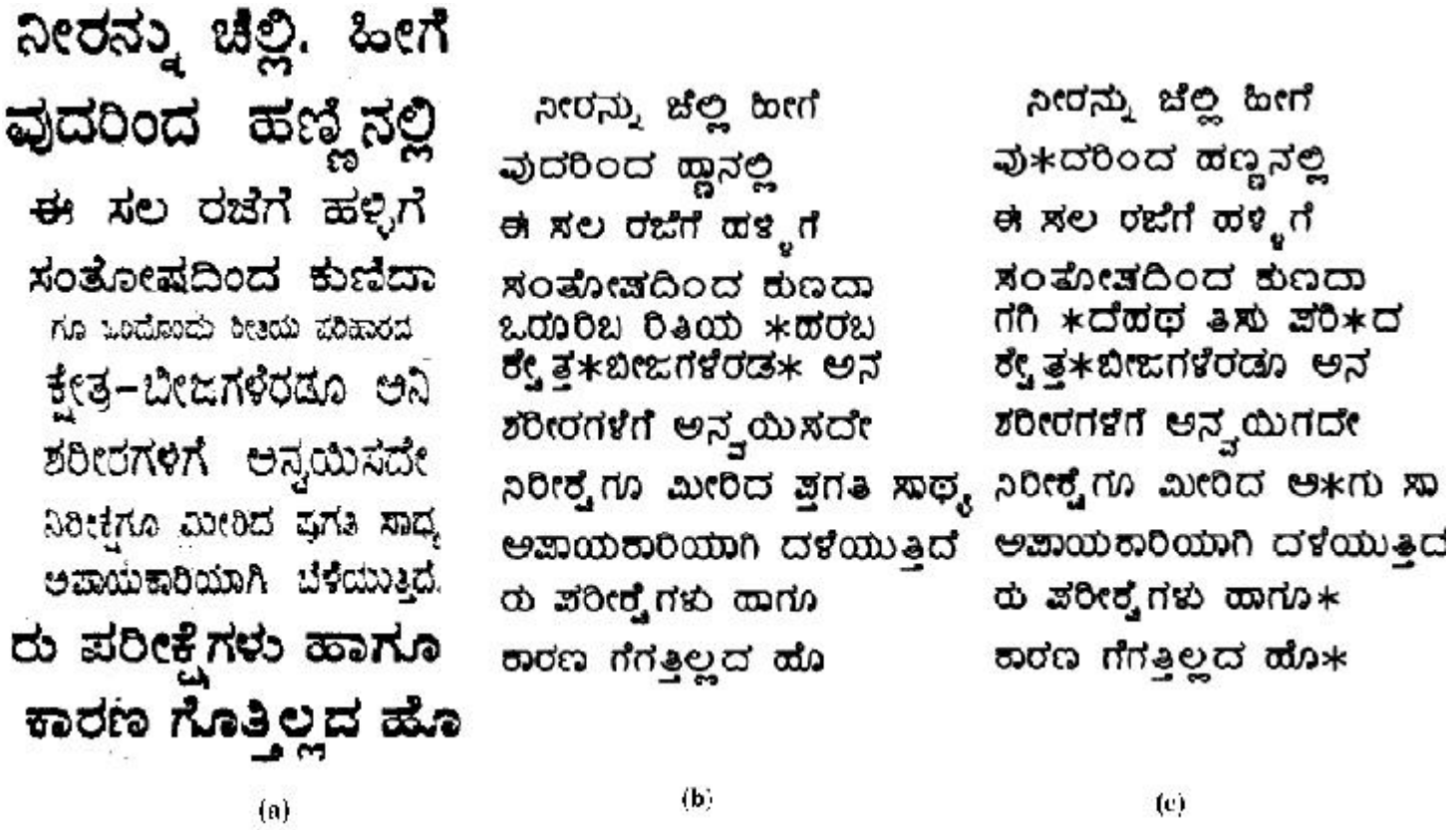


**Figure 11.** Recognized output: **(a)** input page of text; **(b)** recognised output–heuristic merging; **(c)** recognised output–recognition based merging.

**Table 2.** Performance on segments of *aksharas*.

| Base character/*matra*/ consonant conjunct | Features | NNC accuracy(%) | SVM accuracy(%) |
|---|---|---|---|
| Base character | Zernike | 92.76 | 92.63 |
| Base character | Structural | 92.56 | 93.78 |
| Base character | Modified structural | 93.28 | 93.34 |
| Top *matra* | Zernike | 88.13 | 88.43 |
| Top *matra* | Structural | 86.91 | 87.98 |
| Top *matra* | Modified structural | 88.28 | 87.21 |
| Consonant conjuncts | Zernike | 92.22 | 91.95 |
| Consonant conjuncts | Structural | 89.27 | 93.83 |
| Consonant conjuncts | Modified structural | 92.76 | 94.90 |

have also developed a hierarchical SVM for classifying the segments in the middle zone. The classifiers for top and bottom zones are not hierarchical. We have also experimented with a nearest neighbour classifier (NNC) using all training examples as prototypes (thus having multiple prototypes per class).

As explained earlier, we generated the set of training patterns by scanning a number of pages of Kannada text from different magazines, text books etc. These images are segmented by our system and then some of the segments are labelled by us to make-up the training set. Since we have to choose training patterns only from the *aksharas* that appear in the scanned pages, we had different number of training patterns for different classes. The total number of training patterns for the middle zone is 2824. The total number of training patterns for the top and bottom zones are 1332 and 758.

We measured the performance of our system on 16 pages of text scanned from different *Kannada* magazines and books. These pages are taken from two weekly magazines, two monthly magazines, three school texts, two newspapers and three booklets. While the same books are used for generating the training patterns also, it was ensured that the training and test sets of patterns are disjoint.

In our system, we can distinguish between two types of recognition accuracies[4]: the segment recognition accuracy and the *akshara* recognition accuracy. The segment recognition accuracy is actually the accuracy achieved by different classifiers. However, what the user desires is a correct readable output. An error in any component of the *akshara* renders it incorrect. So, from a user point of view, a more meaningful measure of accuracy is the *akshara* recognition accuracy which is the fraction of *aksharas* correctly recognized. Here an error in any component of an *akshara* is flagged as an error and hence the segment recognition accuracies would always be higher. The segment recognition accuracies are useful for validating and/or improving the classifier.

The final results obtained for segment recognition accuracy are shown in table 2 and the those for *akshara* recognition accuracy are shown in table 3 below. All results are average accuracies over the sixteen pages of text.

From table 2, we can see that the classifiers for individual segments deliver reasonably good performance. The recognition accuracy obtained with our 48-dimensional feature vector is at least as good as or better than that obtained with the Zernike features, which are computationally very expensive. The performance of SVM classifiers is a little better than

[4]Whenever we talk of recognition accuracies, we mean the accuracies obtained on the test set of patterns

**Table 3.** Summary of *akshara* recognition accuracies.

| Merging strategy | Feature | Classifier | % Wrong | % Reject | % Error = %wrong + %reject |
|---|---|---|---|---|---|
| Heuristic | Zernike | NNC | 15.60 | 1.22 | 16.82 |
| Heuristic | Zernike | NH-SVM | 11.86 | 6.11 | 17.97 |
| Heuristic | Zernike | H-SVM | 10.74 | 4.68 | 15.42 |
| Heuristic | Structural | NNC | 15.32 | 1.30 | 16.62 |
| Heuristic | Structural | NH-SVM | 12.56 | 5.69 | 18.25 |
| Heuristic | Structural | H-SVM | 11.12 | 4.57 | 15.69 |
| Heuristic | Modified structural | NNC | 15.48 | 1.37 | 16.85 |
| Heuristic | Modified structural | NH-SVM | 13.47 | 5.19 | 18.67 |
| Heuristic | Modified structural | H-SVM | 11.09 | 3.75 | 14.84 |
| Recognition based | Zernike | H-SVM | 10.49 | 3.42 | 13.91 |
| Recognition based | Structural | H-SVM | 11.86 | 3.64 | 15.50 |
| Recognition based | Modified structural | H-SVM | 11.26 | 2.63 | 13.89 |

H-SVM => hierarchical-SVM classifier, NH-SVM => non hierarchical-SVM classifier, NNC => nearest neighbour classifier

that obtained using Nearest Neighbour Classifier (NNC). Here it may be noted that we have used all the training samples as prototypes for the NNC. When the training set size increases, this type of classifier would be inefficient in terms of both memory and computation time.

In table 3 we have shown two types of errors in *akshara* recognition: wrong classifications and rejects. As explained earlier, with the SVM classifier we reject a segment if none of the SVMs give positive output. On the other hand, at segmen level, the NNC does not reject any pattern. However, at the *akshara* level, the labels given to different segments may be incompatible in the sense that they cannot be put together into an *akshara*. We count such errors also as rejects. For the SVM classifier, a reject can result both by rejecting a segment or by incompatibility of class labels of the segments. For the NNC, the rejects can occur only by incompatibility of labels on successive segments.

From table 3 it can be seen that, of the three classifiers (i.e. the NNC, SVM, hierarchical SVM), for the same set of features, the hierarchical SVM shows the lowest overall error rate. Comparing the SVM with the nearest neighbour classifier (NNC), it can be seen that the misclassification rate of the SVM classifier is lower, but the SVM has a higher reject rate. In some applications the cost of misclassification can be much higher than that of rejection. In such cases the SVM classifier is better. Also it is seen that the hierarchical structure considerably reduces both the misclassification and the reject rate of the SVM classifier.

Of the three feature extraction schemes (the Zernike features, the structural features and the modified structural features) it can be seen that the modified structural features give the best performance.

The recognition-based merging scheme shows a definite improvement over the heuristic merging scheme. In our current implementation, the recognition based merging scheme suffers from the disadvantage that it can only merge pairs of successive segments and hence the degree to which the characters are over-segmented is very critical (i.e. if a character is split into more than 2 parts this scheme will reject the character). The heuristic scheme does not suffer from this problem since it validates each segment break point individually and hence is more rugged.

**Table 4.** Computational time for different classifiers with heuristic segment merging and structural features .

| Classifier | Total time (s) |
|---|---|
| NNC | 183.22 |
| Non-hierarchical SVM | 171.92 |
| Hierarchical SVM | 169.54 |

The overall computational time for feature extraction (including the time for line, word and character segmentation, and merging), classification and final output generation for all 16 pages are shown, for different configurations of the system, in tables 4, 5, 6.

Referring to table 4, of the three classifiers, the Hierarchical SVM shows the least computational time. We can compare the computational requirements of NNC and SVM classification methods as follows. In the case of the NNC, to classify a given test pattern, computation of distance with *every feature vector in the training set* is required. In a SVM classifier a dot product computation with *support vectors extracted for every class* is required. If it is assumed that a dot product computation and a distance computation are roughly equal, then the SVM has an advantage in cases where the total number of support vectors for all the classes is less than the number of training examples. This is usually true for most training sets and hence SVM would be advantageous. In our current implementation we have only about 3000 samples for a hundred class problem which works out to only about 30 samples per class. That is the reason why the difference in computational expenses between SVM and NNC is not very significant here. For a very robust (font and size independent) recognition we should have many more training samples. When the sample size is appropriately increased, we can expect the computational advantage of the SVM method to be more pronounced. The training algorithm for the SVM classifier is also very efficient. The training of all the hundred classifiers took less than 30 minutes on a Pentium II system running at 330MHz.

Of the three feature extraction strategies, the modified structural features are only slightly more expensive than the structural features. The Zernike features are roughly 8 times more expensive.

Comparing heuristic merging with the recognition based merging approach, recognition based merging is more expensive. In the recognition based merging scheme, for every pair of segments, the first segment and the merged segment are sent for classification. Therefore the number of segments input to the classifier is roughly twice that of the heuristic approach. The overall time required for the recognition based approach is roughly double that required for the heuristic approach.

Figure 12 shows another example of the recognized output for some paragraphs extracted from the test pages. The '*'s show the characters rejected by the system. The system as

**Table 5.** Computational time for different features with heuristic segment merging and hierarchical SVM.

| Feature | Total time (s) |
|---|---|
| Zernike | 1258.46 |
| Structural | 169.54 |
| Modified structural | 172.27 |

**Table 6.** Computational time for different segment merging strategies with modified structural features and hierarchical SVM classifier.

| Merging method | Total time (s) |
|---|---|
| Heuristic merging | 172.27 |
| Recognition based | 320.24 |

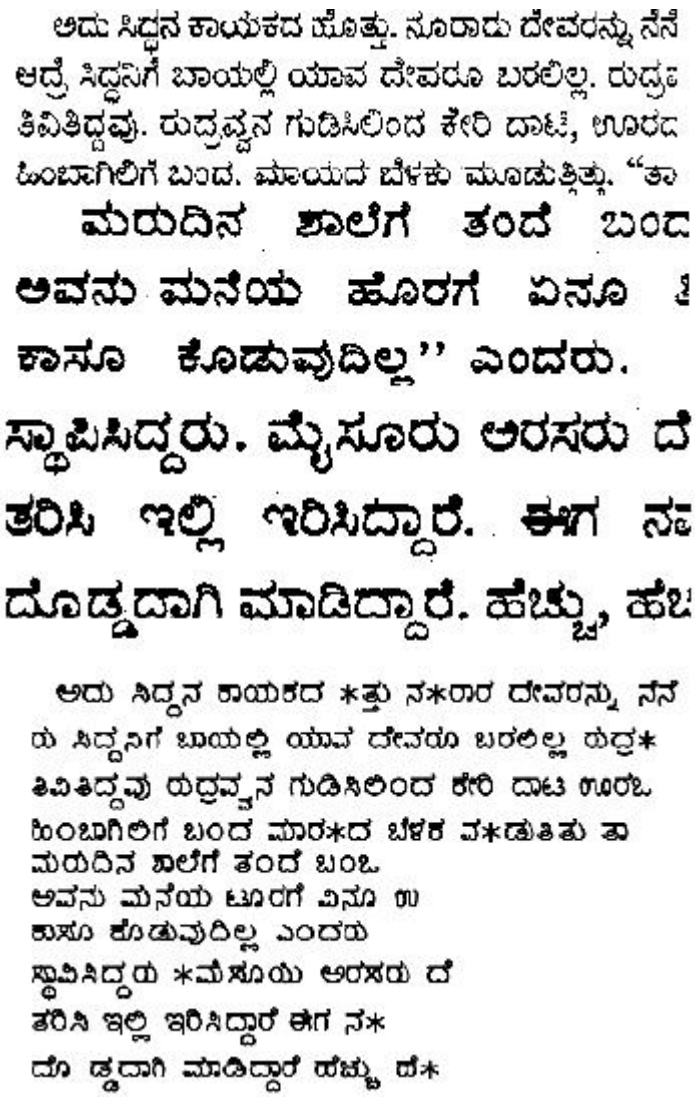


**Figure 12.** Test image and recognized text.

described here does not recognize anything other than Kannada *aksharas*. That is why all punctuation marks also go unrecognized in the output. The results in this figure are obtained using heuristic merging algorithm, modified structural features and hierarchical SVM classifier.

From table 3, recognition based merging seems to perform better than heuristic merging. As remarked earlier, our current implementation is inadequate because we only consider pairs of segments for merging. If we follow a strategy of merging segments in various groups to increase the confidence of the classification we should get better performance. The main strength of such a strategy is that final recognition would be quite robust to oversegmentation or other similar shortcomings of the segmentation algorithm. The contribution of segmentation errors in the final recognition performance is difficult to estimate. But we feel that segmentation errors do have a major contribution. Hence an improved version of recognition based merging may be very effective and we plan to pursue this in our future work.

Looking at recognition rates it can be seen the recognition-based merging approach with modified structural features and the hierarchical SVM classifier gives the best performance and has moderate computational requirements. In the current system we have used the same set of features in the SVM classifier for all the levels of the hierarchical classifier. The segment level recognition rates could be improved by designing a more complex hierarchy with different features at different levels in the hierarchy.

Our structural features essentially capture the distribution of the black pixels around the centre of the character. South Indian scripts such as Kannada, Telugu, Malayalam have a generally rounded appearance. Scripts like Kannada and Telugu have very few strong linear strokes in the characters. We feel that the type of features we have used here should be useful for character recognition in many such scripts. At present we are trying to modify this system (by retraining the classifiers) so that we can handle printed Telugu documents also.

## 7. Conclusions

In this paper we describe an OCR system for processing printed text documents in Kannada, a South Indian language. Starting with the image of a page of printed Kannada text, the system first separates lines and words using standard methods. Then the words are segmented into sub-character-level pieces using a strategy that is tailored to the special features of the script. We propose a novel feature vector that characterizes the distribution of foreground pixels in radial and angular directions. Using these features, each of the segments of a word is classified using a support vector machine classifier. The labels so obtained are then put together to effect recognition of individual *aksharas* in the word. The system is designed to be independent of the font and size of text. We have presented some results with our system which delivers reasonable recognition accuracy.

The work reported in this paper is partially supported by a project under the *Indian Language Technology Solutions – Kannada* programme of the TDIL Group of the Ministry of Information Technology, Government of India.

## References

Antani S, Agnihotri L 1999 Gujarathi character recognition. In *Proc. Fifth Int. Conf. on Document Analysis and Recognition,* Bangalore (IEEE Computer Society Press) pp 418–421

Ashwin T V 2000 *A font and size independent OCR for printed Kannada using SVM*. M E Project Report, Dept. Electrical Engg., Indian Institute of Science, Bangalore

Bansal V, Sinha R M K 1999 On how to describe shapes of Devanagari characters and use them for recognition. In *Proc. Fifth Int. Conf. on Document Analysis and Recognition*, Bangalore (IEEE Computer Society Press) pp 410–413

Bosker M 1992 Omnidocument technologies. *Proc. IEEE* 80: 1066–1078

Burges C 1988 A tutorial on support vector machines for pattern recognition. *Data Mining Knowledge Discovery* 2: 121–167, available at http://svm.research.bell-labs.com/papers/tutorial_web_page.ps.gz.

Choudhury B B, Pal U 1997 An OCR system to read two Indian language scripts: Bangla and Devanagari. In *Proc. Fourth Int. Conf. on Document Analysis and Recognition* (IEEE Computer Society Press) pp 1011–1015

Jagadeesh G S Gopinath V 2000 Kantex, a transliteration package for Kannada available at http://langmuir.eecs.berkeley.edu/˜ venkates/kantex_1.00.html).

Joachims T 1999a Making large-scale support vector machine learning practical. In *Advances in kernel methods – support vector learning* (eds) B Scholkopf, C J C Burges, A Smola (Cambridge, MA: MIT Press) available at http://www-ai.cs.uni-dortmund.de/DOKUMENTE/joachims_99a.ps.gz

Joachims T 1999b *SVM$^{light}$*. http://www-ai.informatik.uni-dortmund.de/FORSCHUNG/VERFAHREN/SVM_LIGHT/svm_light.eng.html

Keerthi S S, Shevade S K, Bhattacharyya C, Murthy K R K 2000 A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Trans. Neural Networks* 11: 124–136

Lee H J, Chen B 1992 Recognition of handwritten Chinese characters via short line segments. *Pattern Recogn.* 25: 543–552

Lu S W, Ren Y, Suen C Y 1991 Hierarchical attributed graph representation and recognition of handwritten Chinese characters. *Pattern Recogn.* 24: 617–632

Mangasarian O L, Musicant D R 1999 Successive overrelaxation for support vector machines. *IEEE Trans. Neural Networks* 10: 1032–1037

O'Gorman L, Kasturi R 1995 *Document image analysis* (IEEE Computer Society Press)

Pavlidis T 1986 A vectorizer and feature extractor for document recognition. *Comput. Vision Graphics Image Process.* 35: 111–127

Platt J C 1999 Sequential minimal optimisation: A fast algorithm for training support vector machines. In *Advances in kernel methods – support vector learning* (eds) B Scholkopf, C J C Burges, A Smola (Cambridge, MA: MIT Press) available at http://www.research.microsoft.com/˜ jplatt

Sekita I, Toraichi K, Mori R 1988 Feature extraction of hand written Japanese characters using spline functions and relaxation matching. *Pattern Recogn.* 21: 821–828

Sinha R M K, Mahabala H 1979 Machine recognition of Devanagari script. *IEEE Trans. Syst., Man Cybern.* 9: 435–449

Trier O D, Jain A K, Taxt T 1996 Feature extraction methods for character recognition - a survey. *Pattern Recogn.* 29: 641–662

Vapnik V N 1995 *The nature of statistical learning theory* (New York: Springer-Verlag)

Vapnik V N 1999 An overview of statistical learning theory. *IEEE Trans. Neural Networks* 10: 988–999