

# Partitioning and Searching Dictionary for Correction of Optically Read Devanagari Character Strings

V. Bansal, R. M. K. Sinha

Indian Institute of Technology, Kanpur - 208016, U.P., India  
email: {veena, rkm}@iitk.ac.in

Received / Revised

**Abstract.** This paper describes a method for correction of optically read Devanagari character strings using a Hindi word dictionary. The word dictionary is partitioned in order to reduce the search space besides preventing forced match to an incorrect word. The dictionary partitioning strategy takes into account the underlying OCR process. The dictionary words at the top level have been divided into two partitions, namely: short words partition and the remaining words partition. The *short* word partition is sub-partitioned using the envelope information of the words. The envelope consists of the number of *top*, *lower*, *core* modifiers along with the number of core characters. Devanagari characters are written in three strips. Most of the characters referred to as core characters are written in the middle strip. The remaining words are further partitioned using *tags*. A *tag* is a string of fixed length associated with each partition. The correction process uses a distance matrix for assigning penalty for a mismatch. The distance matrix is based on the information of errors that the classification process is known to make and the confidence figure that classification process associates with its output. An improvement of approximately 20% in the recognition performance is obtained. For a short word, on an average, 590 words are searched from 14 sub-partitions of short words partition when an exact match is found. The average number of partitions and the average number of words go up to 20 and 1585 respectively when an exact match is not found. For tag based partitions, on an average, 100 words from 30 partitions are compared when either an exact match is found or a word within the preset threshold distance is found. If an exact match or a match within preset threshold is not found, the average number of partitions become 75 and 450 words on an average are compared. This is the first work to the best of our knowledge on using Hindi word dictionary for OCR post-processing.

**Key words:** Postprocessing - Devanagari OCR - Dictionary partitioning

## 1 Introduction

When a human being reads a document, he uses a wide spectrum of knowledge in making sense of what is written. On the other hand, machines when presented with optically digitized text, in the absence of such a knowledge, is prone to making mistakes in reading the text. These errors are primarily caused by noise either inherent in the document or introduced by the digitizing process. Due to the noise, the machine reading process may not be able to recognize a character ( reject error ) or may recognize a character incorrectly (substitution error). The noise sometimes also causes character fusions (i.e. two or more character images merge to appear as a single connected component) and/or character fragmentation (i.e. a character image is fragmented into more than one subimage). The character fusion and fragmentation may lead to substitution and rejection errors besides making the word length shorter or longer than the actual length. One of the common ways of correcting these errors is to make use of a word dictionary of the language of the text to check if the OCR output word is a valid dictionary word. A still higher level of knowledge such as grammar of the language of the text may also be employed to check the output of the OCR.

In this paper, we examine some of the issues concerning correction of optically read Devanagari character strings using a Hindi<sup>1</sup> word dictionary and suggest a partitioning strategy with results of our experimentation. Contextual post-processing have been extensively employed [5], [7], [10], [11], [12], [15], [17], [20], [21] for improving the performance of the OCR in case of Roman script. However, in case of Devanagari, only work reported on post-processing [14] uses the script composition grammar for correcting the output but it does not use a word dictionary. This is the first work to the best of our knowledge on using Hindi word dictionary for OCR

---

<sup>1</sup> Hindi is official language of India that is written in Devanagari script. It is also the script for Sanskrit, Marathi and Nepali languages. Devanagari script is used by more than 500 million people on the Indian subcontinent.

post-processing. Work on Bangla (another Indian script) has been reported [4].

Spell checkers provide a ready mechanism for post-processing the OCR output for corrections. A spell checker takes into account the process by which spelling errors usually get introduced [9], [13] while suggesting the corrections. A spell checker tries to model this process. A model is usually based on character phonetic proximity, character key-label proximity on the keyboard, character interchange, missing character, an extra character, character repetition etc. In another approach, one estimates the likelihood of a spelling by its frequency of occurrence [5], [7], [10], [11], [12], [21] that is derived from the transition probabilities between characters. This requires a priori statistical knowledge of the language. A hybrid [5], [12], [20] approach attempts to combine the language model with the dictionary.

One of the most commonly used model incorporating the OCR process, is a statistical model that incorporates confusion matrix. The confusion matrix captures the substitution errors that OCR makes during the testing phase. The confusion matrix thus obtained is representative only when tested over a large sample space. It also depends upon the OCR methodology and feature vector space used for classification. This confusion matrix is used for hypothesizing the substitution errors and suggesting corrections. The new hypothesized words have to be checked in the word dictionary. A classification process, that uses a distance measure to classify an unknown character, provides a confidence figure with its output. The confidence figure is inversely proportional to the distance. All the characters of the word in the output of OCR may not have equal confidence. The correction process, while correcting the substitution errors, has to take the individual character confidence figures into account. The character fusion and fragmentation pose further problems for the correction process.

The correction process needs an appropriate criterion to select the most likely candidate when more than one hypothesized words find match within the dictionary. A trie-structure has been employed [20], [15] to obtain a set of next possible characters of the dictionary words while traversing the OCR output word character by character. In another approach, a string matching technique [17] has been used to find the best match. A number of strategies have been suggested [16] that partition the dictionary to reduce the number of candidate words. One of the partitioning strategy suggested takes the performance of the OCR into account. It is reasonable to assume an accuracy of 70% or more at the OCR level. For a word of length 3, on an average at least two characters are correctly recognized. The rest of the characters may or may not be in error. Taking this into consideration, dictionary can be partitioned [16] based on all possible combinations of two characters of a word. Thus a word of the dictionary will lie in more than one partition. Within a partition, it is assured that every word will have these two characters in them. We investigate this scheme in detail and adapt it for Hindi words in this papers. Further partitioning is done based on word envelop, word length information and relative positioning of these characters

in the word. This way an hierarchy in dictionary partitioning gets constructed with each partition having a substantially reduced set of words. A similar technique has been used in [22].

The partitioning technique presented in this paper is primarily based on our earlier work for English [16] but it differs in many respects as it is applied for Devanagari script that is a two dimensional logical combination of symbols rather than mere juxtaposition of symbols as in case of Roman. There are a number of upper and lower modifier symbols (called "matra" symbols) used in Devanagari. These symbols play a major role in selection of candidate words for matching. A dictionary partitioning strategy for Devanagari makes use of this feature for possible correction. In this paper we present the partitioning and search strategy that is best suited for Devanagari script. Although Devanagari is a script used by more than 500 million people on the globe, for the benefit of many of the readers, some of the salient features of Devanagari script is presented briefly in the next section primarily from OCR viewpoint including an overview of the OCR stages. We describe the partitioning scheme that has been designed keeping in mind the special features of Devanagari script in section 3. The matching process has been described in section 4. The experimental results have been presented in section 5 followed by concluding remarks in section 6.

## 2 Devanagari Script and an Overview of OCR stages

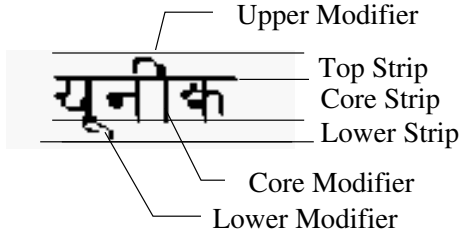
Devanagari is an alphabetic script with logical composition of meaningful constituents that are distinct from ideographic scripts like Chinese and Japanese. Devanagari script consists of a set of vowels and consonants along with various modifier symbols. The vowels and the consonants are referred to as core characters. The consonants are shown in figure 1(a) and the vowels are shown in 1(b)(i). Please notice the horizontal bar at the top of each character that we refer to as *header line* or *Shirorekha*. In a word, the characters are juxtaposed and their header lines join together resulting in a horizontal line at the top of the word. The consonants and vowels are used in the same manner as in English except for two critical differences.

**The first difference:** When a vowel is used to modify the sound of a preceding consonant, its shape changes completely. These changed shapes are referred to as modifier symbols or *matra*.

The modifier symbols are shown in figure 1(b)(ii). Some of the modifier symbols are placed on the top of the header line of the characters. We refer to these modifiers as *top modifiers* and the region that contains these modifiers as *top strip*. Some modifiers are placed below the character (*lower modifiers, lower strip*). A lower modifier may touch the consonant above it. Some of the modifiers are juxtaposed to the left or to the right within the core strip that contains the main characters (*core modifiers*). Figure 1(b)(iii) shows the placement of various modifiers with the help of the first consonant of Devanagari script

(a)		क	ख	ग	घ	ङ	
		च	छ	ज	झ	ञ	
		ट	ठ	ड	ढ	ण	
		त	थ	द	ध	न	
		प	फ	ब	भ	म	
		य	र	ल	व	श	
		ष	स	ह	क्ष	त्र	
		ज्ञ	श्र				
(b)	(i)	अ	आ	इ	ई	उ	
	(ii)		।	ि	ी	ु	
	(iii)		का	कि	की	कु	
	(i)	ऊ	ऋ	ए	ऐ	औ	औ
	(ii)			ॆ	ॆ	ॆ	ॆ
	(iii)	कू	कृ	के	कै	को	कौ
(c)		क	ख	ग	घ	ङ	
		च	छ	ज	झ	ञ	
		ट	ठ	ड	ढ	ण	
		त	थ	द	ध	न	
		प	फ	ब	भ	म	
		य	र	ल	व	श	
		ष	स	ह	क्ष	त्र	
(d)		क्य	ख्य	घ्य	च्य	ज्य	
		त्य	थ्य	प्य	फ्य	ध्य	
		स्य					
(e)		देवनागरी लिपि अत्यन्त सुन्दर है					

**Fig. 1.** Devanagari Script in Brief (a) Consonants; (b)(i) Vowels; (ii) Modifier Symbols corresponding to Vowels in (i); (iii) Modifier Symbols are attached to the first consonant to show their relative position; (c) The consonants in their pure form; (d) Some conjuncts (touching of the characters is clearly visible); (e) sentence written Hindi language using Devanagari script;



**Fig. 2.** The image of sample input text.

(figure 1(a)). The placement of modifiers is uniform over all the consonants (with a few exceptions that are not considered here).

Please refer to figure 2. It is the transliteration of the word *unique* in Devanagari. It is written as the consonant for the sound *y*, the modifier below it to add the sound of *u*; followed by the consonant for *n*, the modifier symbol to add the sound of long *i* that consists of a core modifier and an upper modifier; followed by the consonant for the sound *k*. The three strips and the modifiers have been marked in figure 2.

**The second difference:** It can be best explained with the help of an example. Consider the word *among*, the letter *n* is pronounced without any vowel sound including the vowel *a*. Devanagari script has pure forms

(a)	सुन्दर
(b)	सु न्द र
(c)	सुन्दर
(d)	सुन्दर

**Fig. 3.** Multistage Segmentation of a Devanagari Word (a) Word Image; (b) Units obtained by using the vertical space as delimiter; (c) Units obtained after segmenting the first unit of (b) in the horizontal direction; (d) Units obtained after segmenting the second unit of (b) in the vertical direction;

(also referred to as half characters) of the consonant that are used in such situations. Sometimes, instead of using the pure form of the consonant, a symbol called *halant* is placed below the consonant. In English, the pronunciation is empirical to some extent and people learn by practice. However, in Hindi language that uses Devanagari script, there is one-to-one correspondence between what is written and what is read. The pure form of a consonant is always written touching the following consonant (it is evident that a pure graphical form can not be used as the last character in a word). This brings us to the second difference- *the touching between characters is natural in Devanagari script*. The pure form of a character and the following character together are referred to as *conjuncts* or *touching character pair*. These conjuncts behave the same way as a character as far as attachment of modifiers is concerned. Figure 1(c) shows the pure form of the consonants. Figure 1(d) shows some of the conjuncts. We can form more than 300 conjuncts that is a large number compared to the number of characters in the script. Therefore, it is natural for an OCR for Devanagari script to segment a conjunct to get the constituent characters rather than treat it as a unit. Recall that the lower modifiers may also touch the carrying consonant. As a result, the segmentation may be required in the vertical direction as well as in the horizontal direction. A word and its multistage segmentation is shown in figure 3. In the first stage, the vertical space is used as delimiter to extract the character images ignoring the header line joins. In the next stage, based on the relative heights of image boxes, the *tall* image boxes are segmented horizontally for extraction of the lower modifiers. Finally, based on the relative width of image boxes, the *wide* image boxes are segmented vertically for extraction of the constituent characters of the conjuncts.

We need a mechanism to select the image boxes for further segmentation and an algorithm to perform the segmentation. A detail description of these may be found in [2].

The segmented units are recognized (this process has been described in [3]) and the output of the recognition process goes through an error detection and correction phase. This phase consists of the following three steps: 1. Select an appropriate partition of the dictionary based on the characteristics of the input word. Select the can-

didate words from the selected partition to match the input word with.

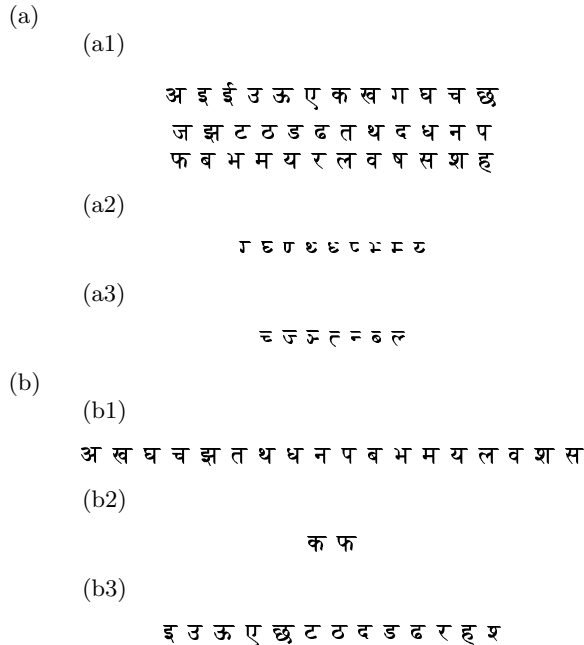
2. Match the input word with the selected words.

3. In case the input word is found in the dictionary, no more processing is done and the word is assumed to be correct. If the word is not found, there are two options available. The first option is: find the *best match* for the input word based on some distance measure criteria. The other option is: generate *aliases* for the input word and look for an exact match.

Takahashi et al. [22] characterize an input word according to a constant number of characters selected from the input word ignoring the relative position for selecting the candidate words from the dictionary. As a result, inappropriate words are selected as candidate words. The selected words are matched with the given word by approximate string matching and a penalty is assigned for the mismatch in the length, for mismatch in the position of the characters being matched and for mismatch between characters. Kahan [6], estimate the probability of a character being another character, as part of the training. These classes are used for forming the aliases and the probabilities are used for assigning a cost to the alias. These aliases are searched in the dictionary till an exact match is found or the cost of an alias is too high.

For Indian languages, only few works have been reported. Chaudhuri [4] in their Bangla OCR use a simple strategy for post-processing. Their effort is limited to dealing with only single character error in a word. The dictionary is used to look for an exact match. In case an exact match is not found, the candidate strings are generated by substituting the character in error by its confusions that are collected during the training phase. Words in the dictionary are kept in alphabetical order. A trie structure stores first three characters of the words and address of the first word of the dictionary of which the first three characters are the same as stored in the trie structure. In case, there is an error in the first three characters of the word, the search process looks at the wrong dictionary words for the match. The candidate strings are formed by using confusions for each character of the word. In case, the word is broken up in the root word and a suffix; under the assumption that there is only single character error, candidate strings are formed from the confusions of root word characters or suffix characters.

In this paper, we suggest general strategy for detecting and correcting errors in a word where the characters in error could be as high as 30%. We use structural properties of Devanagari script for deciding the penalty of a mismatch. For instance, the core characters are divided into three classes based on the region of the core strip covered. Most of the core characters cover the entire core strip and are referred to as FULL BOX characters. There are characters that cover only the upper region of the core strip and these are referred to as UPPER HALF BOX characters. LOWER HALF BOX characters are the characters that cover the lower region of the core strip. These sets are shown in figure 4(a). FULL BOX characters are further divided into three classes based on the presence and position of vertical bar (see figure 4(b)).



**Fig. 4.** Various Sets of Devanagari script based on Structural Properties (a) Three classes of core characters based on the coverage of core strip: (a1) FULL BOX characters; (a2) UPPER HALF BOX characters; (a3) LOWER HALF BOX characters; (b) Three classes of FULL BOX characters based on the presence and position of vertical bar: (b1) End Bar characters; (b2) Middle Bar characters; (b3) No Bar characters;

We use the above classification as a constraint while substituting a character with another in the case an exact match is not found in the dictionary for an input word. The substitution character must belong to the same class as the substituted character. The position of the vertical bar, if it is present, must also match. These restrictions have been imposed keeping in mind the nature of errors that occur in the character classification process. This prohibits the substitution process from making arbitrary substitutions. The penalty is comparatively less for a mismatch when it is part of the confusion matrix, obtained through training (see section 4 for details).

In case of substitution errors, the true character may be missing or may be present in the set of alternate choices. The OCR [3] generates multiple characters as output for a single image box. We form aliases from the top three choices for each character assuming that the true characters of a word are either the top choice or present in the subsequent two choices. In case, the true character is missing from the top three choices, incorrect aliases are formed that hopefully get corrected through minimum distance mapping from the word dictionary. The number of aliases may become large depending on the number of characters in the words. To avoid searching all the aliases, we set a threshold on the distance and terminate the search when a word within the threshold distance is found.

The correction method presented here uses a partitioned Hindi word dictionary. The partitioning scheme ensures that the right partition will be searched for a given input word as long as the errors are below a specified threshold. In the next section, we describe the partitioning scheme that has been designed keeping in mind the special problems that Devanagari script poses for OCR. The matching process has been described in section 4. The experimental results have been presented in section 5 followed by concluding remarks in section 6.

### 3 Dictionary Organization

The dictionary needs to be partitioned in order to reduce the search space besides preventing forced match to incorrect word. The partitioning strategy should be such that the true word corresponding to the input word is always found in the partition possessing the same partitioning feature as the input word as long as the number of errors in the input word are below a certain threshold. Our two level partitioning strategy is based on the length of the word, word envelop, character combination and presence of modifiers symbols.

Words of length 2 or less are referred to as *short* words. At the top level, there are two partitions:

- Short words
- Long words

*Short words* are further divided into various partitions. The *short* words are partitioned based on the word shapes [18], [19]. Word *envelope* [16] has also been used for referring to the word shape. In Roman script, there are ascenders, descenders and the core characters. For Devanagari script, a character descends due to the presence of a lower modifier and ascends due to the presence of a top modifier. However, the horizontal span of a modifier does not always correspond with the carrying character and may cover the adjacent character. Therefore, while partitioning the words, we do not take the position of modifiers into consideration. In addition to the number of each type of modifiers, we also use the number of the core modifier - *vertical bar* for partitioning. We have found from our experimentation that the recognition rate for this modifier symbol is very high. The partitioning feature consists of the number of modifiers and the core characters. Theoretically, a core character has zero or one modifier of each type. Taking all combinations of modifiers and two core character words into consideration, we require 27 partitions, one for each combination. Similarly, we need 8 partitions for single core character words while number of modifiers of each type is either zero or one. Some of the combinations of modifiers do not form valid syntactic words as per the script composition grammar. For the valid combinations, we require only 28 partitions that are listed in table 1. Sometimes, more than one modifier of a type is attached to a core character. However such words are not many and we put all of these in a single partition. Table 1 gives the number of words in each partition for 6,422 words. The average number of words in a *short* word partition is

- (a) अदरक  
(b) अद, अर, अक, दर, दक, रक.

**Fig. 6.** A Hindi word and its tags: (a) The word; (b) Six distinct tags.

tag	a few example words in the dictionary partition
अद	अदरक अपवाद अवसाद
अर	अदरक अक्षर अंतर
अक	अदरक अथक अवकाश
दर	अदरक आदर हैदर
दक	अदरक दमक पदक
रक	अदरक भरकम भरसक

**Fig. 7.** A few tag partitions and a few sample words of each partition.

Total number of words	107,900
Number of Partitions	5,378
Average Number of repetitions for a word	8
Average Number of words in a tag_partition	163

**Fig. 8.** Statistics of Partitions

220, minimum is just 3 and maximum number of words in a partition are 1082.

The hierarchical structure of the short word partitions is shown in figure 5.

The remaining words are further partitioned with the help of *tags*. A *tag* is a string of fixed length of characters that is associated with a partition. A word is added to a partition if the word contains the tag associated with that partition. We have fixed the length of the tag at two characters ( We give justification shortly). A word of length four has six tags; all of which may not be distinct. For instance, Hindi word for ginger is *adaraka*. Tags for this word are shown in figure 6.

Therefore, the word *adaraka* goes into six partitions that correspond to these tags. Figure 7 shows the 6 partitions and some of the words of each partition. This way some redundancy is introduced. This redundancy ensures that if any two characters of the word have been recognized correctly, at least one tag will point to the correct partition.

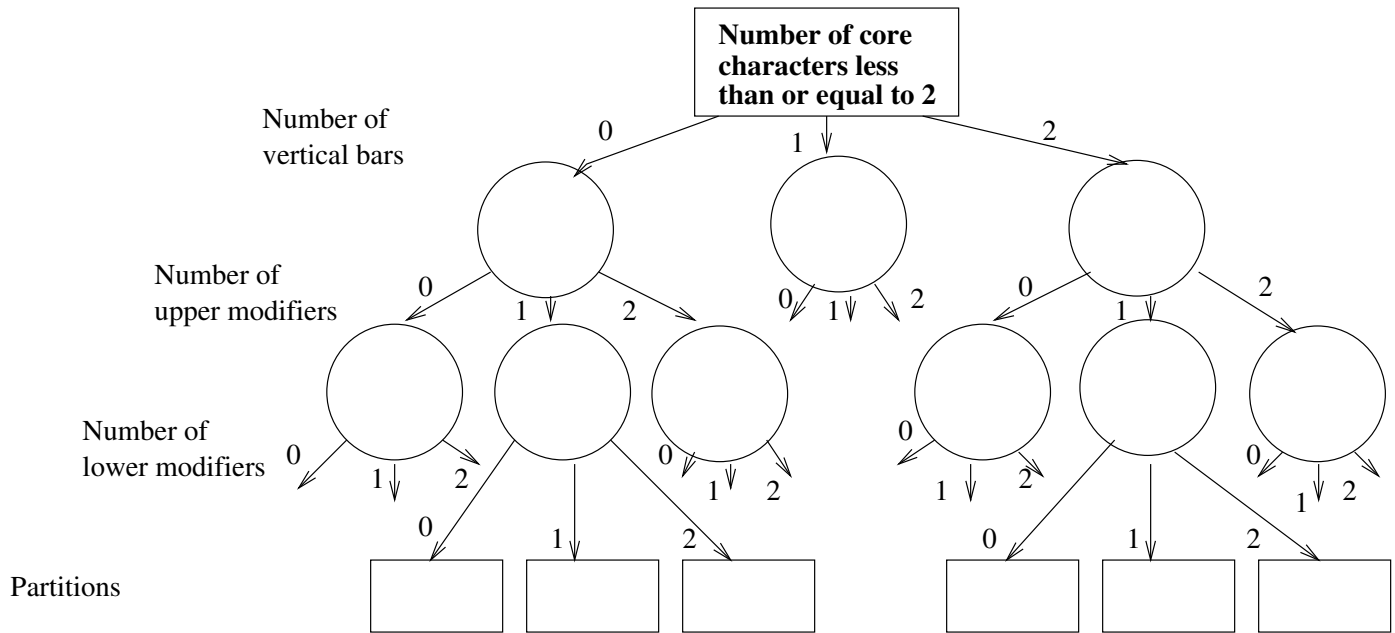
If the character classification process has a recognition rate of 70%, two characters out of the three on an average will be correctly classified. Our experimentation has shown that approximately 65% words by usage contain two core characters, 18% contain three core characters and the remaining words are of length four or more. In other words, tag length two is necessary if the recognition rate is approximately 70%.

Number of partitions, repetition count etc. for 107,900 words are shown in figure 8. The hierarchical structure of the long word partitions is represented in figure 9.

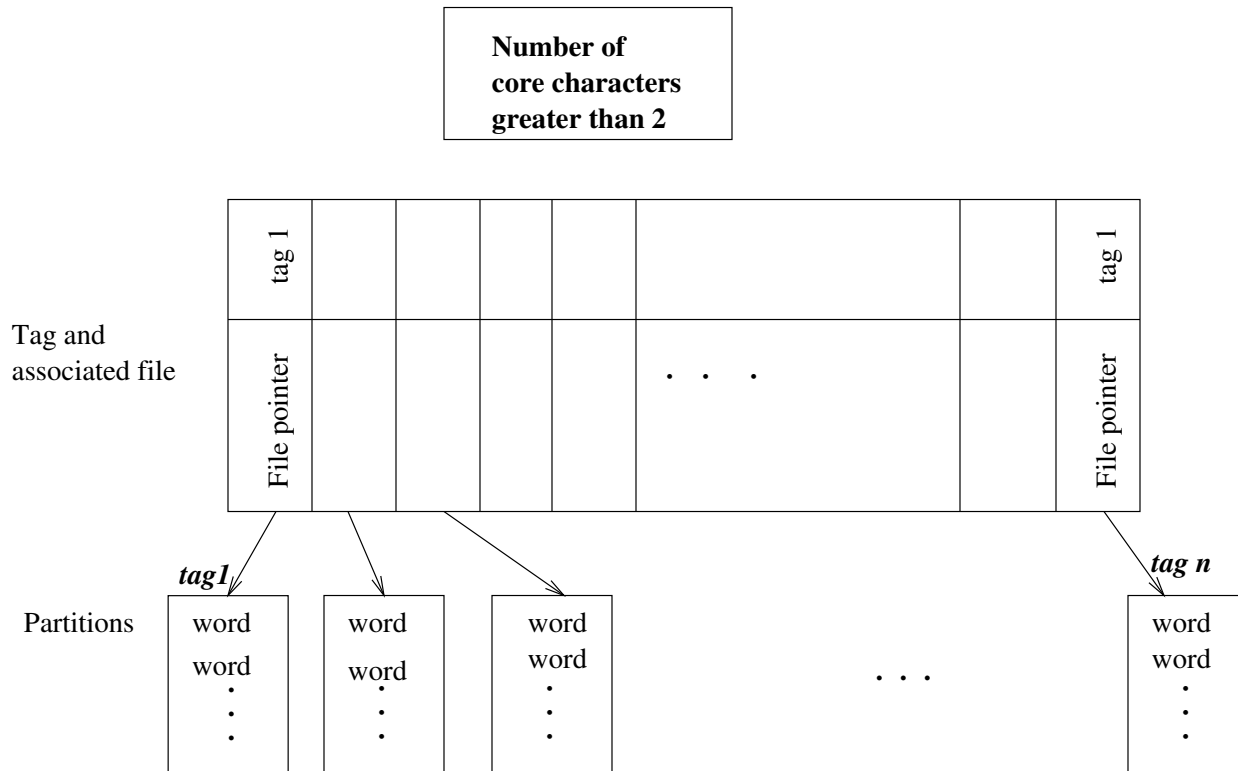
At the time of creation of the partitions, we first check the length of a word. If it is a *short* word, it is added to an appropriate *short* word partition according

NUMBER OF CORE CHARACTERS = 2							
no. of modifiers			no of words	no. of modifiers			no. of words
lower	top	core		lower	top	core	
0	0	0	296	0	0	1	233
0	0	2	16	0	1	0	449
0	1	1	216	0	1	2	16
0	2	0	163	0	2	1	50
1	0	0	498	1	0	1	253
1	0	2	16	1	1	0	1082
1	1	1	254	1	1	2	9
1	2	0	564	1	2	1	137
1	2	2	3	2	0	0	378
2	0	1	54	2	1	0	610
2	1	1	67	2	2	0	470
2	2	1	42				
NUMBER OF CORE CHARACTERS = 1							
0	0	1	74	0	1	1	87
1	0	0	15	0	1	0	29
1	1	0	12				
The Remaining Combinations of Lower, Top and Core Modifiers (not covered above)							425

**Table 1.** Partitions and number of words in each partition for *short* words based on the word envelop and selected characters partitioning feature.



**Fig. 5.** Hierarchical Organization of the Dictionary Words of Length Two or Less for Devanagari Text Recognition System.



**Fig. 9.** Hierarchical Organization of the Dictionary Words of Length More Than Two for Devanagari Text Recognition System.

to its shape. Otherwise, all the distinct tags for the word are used to add it to the corresponding partitions. The dictionary words occupy 1.2MB before partitioning. After partitioning, the dictionary occupies 6.4MB including all the overheads.

#### 4 Matching

We assume that the character units decomposed by the segmentation process have been composed back in valid syntactic character strings using Devanagari script composition grammar [14]. The modifiers have been placed appropriately with their carrying characters.

Two main sources of errors are incorrect segmentation and classification. Our segmentation process [2] consists of two distinct stages. In the first stage, a preliminary segmentation is performed that leads to isolation of character boxes that are vertically separate from their neighbours. The statistical information about height and width of image boxes obtained by the preliminary segmentation is used to mark the image boxes that need further segmentation. Some of the marked character boxes may contain characters under ‘shadow’ of each other. A character is said to be under the shadow of the other when the characters do not physically touch each other but it is not possible to separate them merely by drawing a vertical line. Some of the image boxes contain lower modifiers and some may contain conjuncts. All of these image boxes are segmented in the second stage. The second stage is invoked after the classification process has tried to classify an image box and has failed to classify

it into one of the known classes. This type of strategy is referred to as loose coupling between segmentation and classification.

Irrespective of the mechanism used for selecting the touching character boxes for further segmentation, two types of errors always occur. A fused character box remains unsegmented and a single character is wrongly segmented into more than one character box. In either case, the post-processing phase must ensure that the word corresponding to the correct segmentation, is always included in the words that are searched.

To ensure this, we form words with the OCR output corresponding to every alternative segmentation.

Output of a classifier, that computes distance of a feature vector from an unknown class to the known classes, is correct if the minimum distance known class is the true class. However, it is possible that instead of the minimum distance class, the true class is the second minimum or third minimum distance class. We refer to this error as ranking error. A ranking error pushes the true character to a lower position. Therefore, we form multiple aliases of the word from the output of OCR using top 3 choices for each character. Figure 10 shows an input word, output of the classifier and 10 words out of 27 possible words in the order of their formation.

The true word is one of the words formed if every character of the word is in top 3 choices. Number of aliases is at most 9 for *short* words and 27 for three character words and so on. The number of alias words goes up substantially if top 3 choices for modifiers are also used for forming the alias words. To restrict num-

- (a) Input word: हमारा  
 (b) Output of the classifier: (द, ह, ध) (म, भ, प) (।) (र, द, श) (।)  
 (c) Words formed: दमारा हमारा धमारा दभारा हभारा धभारा हमदा दमादा हभादा दभादा

**Fig. 10.** Aliases formed from top three choices, (a) true word; (b) ranked output of the classification process; (c) 10 out of 27 possible words in the order of formation

ber of alias words, we use the following two measures:

1. If the confidence figure associated with a character is above a certain threshold, second and third choices are not used for forming the alias words. The threshold is empirically decided but can be learnt through an artificial neural network.

2. Further search is not done if the closest match is within a preset threshold distance. The distance is the cumulative penalty for the substitutions required to map the OCR word to dictionary word.

Levenshtein distance generalized by Okada [8] uses three types of penalty - one for each substitution, deletion and addition of characters. Takahashi [22] uses two types of penalties - one for a mismatch and the other for addition/deletion.

We have extended the concept of uniform penalty for a mismatch to include different penalties for various kinds of mismatches. Our argument is that when 'a' is being substituted by 'o', the penalty has to be less than when 'a' is substituted by 'k' because 'a' is more likely to be confused with 'o' than with 'k'. We take an example from Takahashi's paper. The input word is *vight* and the candidate words selected from the dictionary are *eight*, *right*, *night*, *sight*, *might*, *tight*, *light*, *fight* with penalty '1' and several others with higher penalties. If the experimentation shows that 'v', 'r' and 'n' get confused, the penalty for words *right* and *night* could be made less. If the classification process is known to make certain errors, there is no reason for not making use of this information and continue to assign uniform penalty.

The penalty figures are shown in table 2. The penalty figures are decided by testing them on large set of test data. These could also be decided by training the system using an artificial neural networks. During the testing phase of our text reading system, we collected the character classes that the OCR tends to confuse. These confusions are collected and provided to the matching process. The penalty of a substitution is '1' when the substituted character is a known confusion of the character being substituted and the confidence figure associated is low. If the confidence figure is high, the penalty is increased to '3'. When the substitution is not a known confusion and only the bar property is same, the penalty figure goes up further to '4'. The penalty figure becomes '6' if the bar property is also violated. A substitution for the *vertical bar* (core modifier) incurs a penalty of '10' because the recognition rate for this modifier is very high. Table 3 shows the confusion classes for our system. A lower modifier is ignored if the two words do not have a lower modifier at matching positions. This incurs a penalty of '2'. Devanagari script has a lower modifier that looks like a '.'(dot). Sometimes, noise is detected

as a dot and sometimes a dot is missed. This rule takes care of both the situations. The difference in the length is multiplied by six and added to the distance. The accumulated penalty is divided by the word length in order to normalize the penalty score.

The algorithm for the search strategy is given in figure 11. A word is checked in the selected short word partition if the word contains two or less core characters. The search ends if an exact match is found. If the word is not short, tags of the OCR word are used to select partitions for search. While extracting the tags from the OCR word, we also note the position of both the characters of the tag in the OCR word. In other words, the two character tag becomes  $n_1ch_1n_2ch_2$  where  $n_1$  and  $n_2$  are the positions of  $ch_1$  and  $ch_2$  with respect to the beginning of the word. While matching a word of the selected partition with OCR word, we use only those words whose length is at most one less or more than the OCR word. The candidate words are selected from the partition based on the length of the input word. This way, we avoid matching input word with each word of the selected partition. Recall that a short word partition contains words of same length, where as the partitions for other words is not based on the length of the word. The experimentation with the system shows that average number of dictionary words used for matching for a short word partition is the average number of words in short word partitions. For other words, the average number of dictionary words used for matching is about half the average number of words in the partitions. After checking the length, we check the relative positions of the tag characters. If the relative positions are off by at most one, further matching is done. Figure 12 illustrates this process with the help of an example. After exhausting all words of the selected partition, the next tag is used to select another partition. Whenever an exact match is found or the distance is less than the preset threshold, the search is terminated. The effect of the threshold on the performance, number of partitions probed and words compared are described in the next section.

The algorithm for comparing characters of the OCR word and the dictionary word is given in figure 13. Initially, the characters at the beginning positions of words are compared. The penalty is decided based on the rules given in table 2. If the penalty is less than one, the positions of characters to be compared next is advanced by one for the OCR word and the dictionary word. If the penalty is more than one and the mismatch is in half and full form, we advance the position of the character to be compared to the next character only for dictionary word. If we find a lower or upper modifier in the OCR word and no corresponding modifier in the dictionary word, we advance the OCR character position by one. In each

situation	penalty assigned
1. The difference in length is d	6 * d
2. The character being mapped has low confidence figure and the character being substituted is a known confusion	1
3. The character being mapped has high confidence figure and the character being substituted is a known confusion	3
4. The vertical bar property of the character being substituted and the substitution matches	4
5. The vertical bar property of the character being substituted and the substitution does not match	6
6. The core modifier   is being substituted	10
7. A top modifier is being substituted	2
8. A lower modifier is being substituted	2
9. A half character or an upper or lower modifier is ignored	2
10. all other substitution	10

**Table 2.** Distance calculation rules used by matching process.

Character Confusion Matrix for the output of OCR			
OCR output	Possible True Chars.	OCR output	Possible True Chars.
अ	ख क्ष भ	इ	ह ड
उ	र	ऊ	रु
र	द	घ	अ ध
ज	न व ब	म	य प
ट	द ड	य	प म
त	ल	व	ब
ल	त	थ	य
न	व ब	प	य म भ
भ	म प	ब	न

**Table 3.** Character Confusion Matrix for the output of OCR.

case, a penalty is added to the distance figure according to table 2. The difference in the length is checked at the end and the penalty is added to the distance.

## 5 Experimental Results

A sample input text is shown in Figure 14. The corresponding output of the classification process is shown in figure 15. Only the words formed from top ranked characters are included in figure 15. The output after corrections using dictionary is shown in figure 16. The word is shown in curly brackets if it is not the top choice of the search process.

The system has been tested on large number of printed documents taken from magazines and news papers. We have tested the system on two different fonts and on 15,000 words. We tested the system with the threshold of 3.0 and 1.0 for a word for terminating the search. The results are presented in table 4. The number of words formed per true word for each document is given in first column of these tables. The number of partitions and words that are probed are also given in the same table. When an exact match is found, number of partitions and words searched are less compared to that searched when an exact match is not found. Words that contain reject errors or substitution errors cause the selected partition to be searched exhaustively. We have counted these numbers for *short* words, 3 character words and the remain-

ing words separately. Graph in figure 17 shows the performance at word level before and after post-processing for word length two, three and greater than three for threshold 1.0 for terminating the search. The improvement in the performance is minimum for short words (about 3%) and maximum for words that have more than four core characters(about 30%). The long words are more likely to have conjuncts that are major source of incorrect segmentation. Among all the words, it is most difficult to provide correct alternative in case of *short* words. The alternatives are usually too many with same distance. For instance, for the OCR word आना, the alternatives are the following:

आना, खाना, जाना, ठाना, ताना, थाना, दाना, नाना, पाना, बाना, भाना, माना, लाना, साना.

All of the above words are valid dictionary words and have same distance from the true word. In other words, if first letter is not the true character in आना, it cannot be corrected. This is the reason, we have not used any threshold for stopping the search in case of short words. Sometimes a ranking or substitution error maps the true word to another dictionary word. These errors go undetected at the word level. An example sentence is the following:

input sentence: हमें आचार्यों का आदर करना चाहिए

ioer output sentence: हमें आचार्यों का चादर करना चाहिए

```

initialize top_three_words;
initialize top_three_dis;
for every alias word formed from the output
    of OCR for the true word
{
  if ( no. of core characters =< 2 )
  {
    use no. of core characters, lower,
    top and core modifiers to select
    short word partition;
    search selected short word partition
    retaining top three choices in
    local_top_three based on the distance;
    if ( the distance of top choice is zero )
      exit; /* from for loop */
    else
    {
      store top three choices from
      top_three_choices and local_top_three
      into top_three_choices;
      store the corresponding distances
      in top_three_dis;
      repeat the search process for next alias;
    }
  }
  else
  {
    extract tags from OCR word;
    for each tag
    {
      search selected tag word partition
      retaining top three choices in
      local_top_three based on the distance;
      if ( the distance of top choice is zero or
          less than a preset threshold)
        exit; /* from for loop of tags and for words */
      else
      {
        store top three choices from
        top_three_choices and local_top_three
        into top_three_choices;
        store the corresponding distances
        in top_three_dis;
        repeat the search process for next tag;
      }
    }
    repeat the search process for next alias;
  }
}
return top_three_choices and top_three_dis;

```

Fig. 11. Algorithm for partitioned dictionary search.

- (a) Input Word : बनाओ  
 (b) Tags : बन, बअ, नअ  
 (c) Tags with character positional information:  
     1 ब 2 न, 1 ब 4 अ, 2 न 4 अ  
 (d) Some of the words in the partition associated with tag  
 नअ along with their length
- |          |   |         |   |
|----------|---|---------|---|
| भावनाओं  | 7 | घटनाओं  | 6 |
| बनाओ     | 5 | बनाओगी  | 8 |
| बनाओगे   | 7 | कामनाओं | 7 |
| कल्पनाओं | 7 | नेताओं  | 5 |
| वेदनाओं  | 6 |         |   |
- (e) Some of the selected words based on the length of the  
 input word:  
     घटनाओं बनाओ नेताओं वेदनाओं  
 (f) Penalty for each word when matched with input word:
- |        |   |         |   |
|--------|---|---------|---|
| घटनाओं | 7 | बनाओ    | 0 |
| नेताओं | 5 | वेदनाओं | 7 |

Fig. 12. Words in the partition associated with tag नअ अ and length of each word ( only core characters are counted )

In this example sentence the fourth word has become another dictionary word due to the substitution error of first character of the word at the OCR level.

Such errors go undetected at the word level and can be corrected only by using syntactic and semantic knowledge [1]. We are using only upto word level knowledge in our system. The domain knowledge and sentence level knowledge could be integrated to further enhance the performance besides making it more robust.

A question we have not addressed so far is: should a non-dictionary word be always mapped to a dictionary word? The answer is no, and to avoid unnecessary mappings, we have made the penalty of substitution higher for the characters that have a high confidence figure associated. In addition, the mapping is not accepted if the minimum distance is more than a preset threshold.

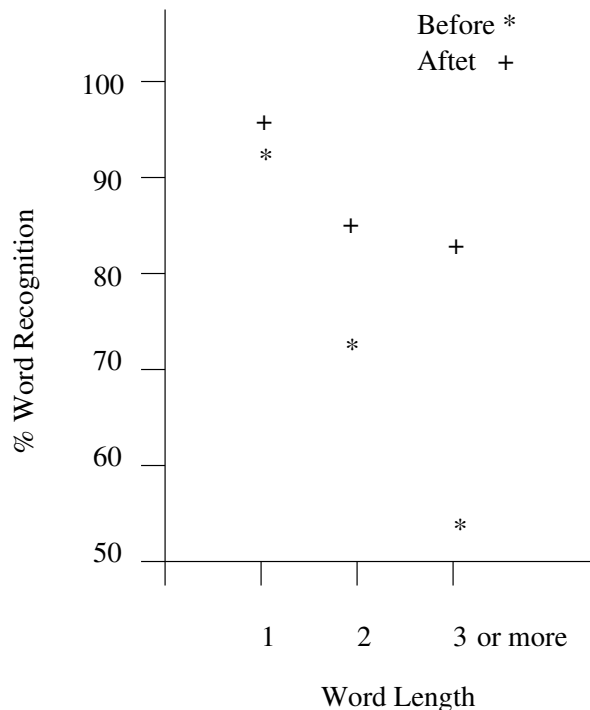
## 6 Conclusions

We have presented a two level partitioning scheme and search algorithm for the correction of optically read Devanagari characters of text recognition system for Devanagari script. We have extended the concept of uniform penalty for a mismatch to include different penalties for various kinds of mismatches. The preference is given to the mappings that are known OCR confusions. At present, we store all forms of a word in the dictionary and do not worry about inflection. The methodology described here makes use of the structural properties of the script, namely - presence of top modifiers, lower modifiers and core modifiers that is unique to Indian scripts.

*Acknowledgements.* Part of the work has been supported by Department of Electronics, Govt. of India.

## References

1. Bansal V, Sinha RMK (1996) Designing a Front End OCR System for Indian Scripts for Machine Transla-



**Fig. 17.** Performance of the system at word level before and after post-processing stage; for search terminating threshold is set to 1.0. The graph shows the trend.

word length	distance threshold	Average no. of words formed per true word with consideration of touching/fragmented characters	search			
			match is found		match is not found	
			partitions probed per true word	words matched per true word	partitions probed per true word	words matched per true word
$\geq 2$	3.0	9.51	3.11	444.88	19.60	1584.11
$\geq 2$	1.0	17.02	14.05	590.66	39.44	2379
3	3.0	6.98	5.88	17.92	35.35	162.83
3	1.0	18.07	33.36	148.77	98.63	728.66
$\geq 4$	3.0	4.95	8.53	9.14	32.78	67.45
$\geq 4$	1.0	9.91	26.05	61.34	55.25	115.18

**Table 4.** Dictionary Search Performance for two, three and four or more core character words for search termination distance threshold 3.0 and 1.0

- tion - A Case Study for Devanagari. In: Symposium on Machine Aids for Translation and Communication SMATAC-96, New Delhi, India
- Bansal V, Sinha RMK (1998) Segmentation of Touching Characters in Devanagari. In: Proceedings of the Indian Conference on Vision, Graphics and Image Processing, Delhi, pp 371-376
- Bansal V, Sinha RMK (2000) Integrating Knowledge Sources in Devanagari Text Recognition. IEEE Transactions on System, Man and Cybernetics - Part A: Systems and Humans 30(4):500-505
- Chaudhuri B B, Pal U (1997) A Complete Printed Bangla OCR System. Pattern Recognition 31(5):531-549
- Hull JJ, Srihari SN (1980) Experiment in text recognition with binary n-gram and viterbi algorithm. IEEE Transactions on Pattern Analysis and Machine Intelligence 4:520-530
- Kahan S, Pavlidis T, Baird HS (1983) On the recognition of printed characters of any font and size. IEEE Transactions on Pattern Analysis and Machine Intelligence 5:384-395
- Neuhoff DL (1975) The Viterbi Algorithm as an aid in text Recognition. IEEE Transactions on Information Theory 21:222-226
- Okada T, Tanaka E, Kasai T (1976) A method for the correction of garbled
- Peterson JL (1980) Computer program for detecting and correcting spelling errors. Communications 23:667-687
- Riseman EM, Ehrich RW (1971) Contextual Word Recognition using binary diagrams. IEEE Transactions on Computer 20(4):397-403
- Riseman EM, Hanson AR (1974) A Contextual Post Processing System for Error Correction using binary n-grams. IEEE Transactions on Computer 23:480-493

```

in_pointer = 0;
dic_pointer = 0;
while ( end of dic_word or
        end of ocr_word is not reached )
{
    dis= chmatch(ocr_word[in_pointer],
                 dic_word[dic_pointer]);
    if ( dis < 1 )
    {
        acc_dis = acc_dis + dis;
        inc in_pointer;
        inc dic_pointer;
    }
    else if (!(ishalf(ocr_word[in_pointer])) &
             ishalf(dic_word[dic_ointer]))
    {
        acc_dis = acc_dis + dis;
        inc in_pointer;
    }
    else
        if(lower_modifier(ocr_word[in_pointer])) &
           !(lower_modifier(dic_word[dic_ointer]))
        {
            acc_dis = acc_dis + dis;
            inc in_pointer;
        }
    else
        if(upper_modifier(ocr_word[in_pointer])) &
           !(upper_modifier(dic_word[dic_ointer]))
        {
            acc_dis = acc_dis + dis;
            inc in_pointer;
        }
    else
    {
        acc_dis = acc_dis + dis;
        inc in_pointer;
        inc dic_pointer;
    }

    acc_dis = acc_dis +
        absolute ( in_pointer - dic_pointer );
}

```

**Fig. 13.** Algorithm for comparing two words and calculating the distance.

12. Shinghal R, Toussaint GT (1979) Experiments in text recognition with modified Viterbi algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1:184-192
13. Sinha RMK, Singh KS (1984) A program for correction of single errors in Hindi words. *Journal of Institution of Electronics and Telecommunication Engineers* 30(6):129-251
14. Sinha RMK (1987) Rule based contextual post-processing for Devanagari text recognition. *Pattern Recognition* 20(5):475-485
15. Sinha RMK, Prasada B (1988) Visual Text Recognition through Contextual Processing. *Pattern Recognition* 21:463-479
16. Sinha RMK (1989) On partitioning a dictionary for visual text recognition. *Pattern Recognition* 23(5):497-500
17. Sinha RMK, Prasada B, Houle G, Sabourin M (1993) Hybrid Contextual Text Recognition with String Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15:915-925
18. Spitz LA (1995) An OCR Based on Character Shape Codes and Lexical Information. In: *Proceedings of International Conference on Document Analysis and Recognition* 723-728
19. Spitz LA (1997) Multilingual Document Recognition. In: Bunke W, Wang PSP (eds) *Handbook of Character Recognition and Document Image Analysis*. World Scientific, pp 259-284
20. Srihari SN (1983) Integrating diverse knowledge sources in text recognition. *ACM Transaction, Office Information System* 1:68-87
21. Suen CY (1979) n-gram statistics for natural language understanding and text processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1:164-172
22. Takahashi H, Itoh N, Amano T, Yamashita A (1990) A Spelling Correction Method and its Application to an OCR system. *Pattern Recognition* 23:363-377

## The input text

या आदेश देने से पूर्व, यदि आप अपने सहयोगियों से विचार-विमर्श करते हैं तथा उनके विचारों को जानने के बाद कोई निर्णय लेते हैं और उस निर्णय का औचित्य भी उन्हें बताते हैं तो उसका पालन हमेशा ज्यादा अच्छा होगा। तानाशाही और मनमानी ज्यादा दिन तक नहीं टिकती। सलाह-मशविरे के बाद लिये गये निर्णय का पालन सदैव ज्यादा अच्छा होता है।

कई बार माता-पिता को परिवार के हित में, अपनी संतान के हित में उन्हीं से संबंधित कुछ कठोर निर्णय लेने होते हैं। कई बार अनुशासनहीन बच्चे को हॉस्टल में भी डालना पड़ता है। कई बार सजा देनी पड़ती है। इसी प्रकार से, एक नेता को अक्सर अपने अधीनस्थ कर्मचारियों से संबंधित कठोर निर्णय भी लेने पड़ते हैं। जब कभी कठोर निर्णय उचित जान पड़ें तो एक नेता को उससे हिचकिचाना नहीं चाहिए। यदि आप हिचकिचा गये तो समझिये कि आपका नेतृत्व खत्म हो गया। यदि कर्मचारियों को यह एहसास हो जाए कि आप कठोर निर्णय लेने से डरते हैं तो फिर बिगड़े हुए कर्मचारियों की हिम्मत बढ़ जाएगी। अच्छे कर्मचारी निराश हो जाएंगे क्योंकि उनमें और दूसरे कर्मचारियों में कोई फर्क नहीं रह जाएगा। अनुशासनहीनता को यदि कमजोर नेता सहन करता है तो अनुशासनहीनता कई गुना बढ़ जाती है। यदि आप अनुशासनहीनता को शुरू में ही कठोरता से दबोच देते हैं तो कुछ समय के लिए काफी

Fig. 14. The image of sample input text.

## Output of OCR

या आउटा रेने से पूर्व यदि आप अपने सहयोगियों से विचार-विमर्श करते हैं तथा उनके विचारों को जानने के बाद कोई निर्णय लेते हैं और ठक निर्णय का औचित्य भी उन्हें बताते हैं तो उसका पालन हमेशा ज्यादा अन्ना होगा तानाशाही और मनमानी \*ादा दिव तक नहीं टिकती सलाह-मशविरे के बाद लिये गये निर्णय का पालन सदैव कारा अला होता है

कई बार माता-पिता को परिवार के हित में अपनी संतान के हित में उन्हीं से संबंधित कुछ कठोर निर्णय भी लेने होते हैं कई बार अनुशासनहीन बच्चे को हास्टल में ठी डालना पड़ता है कई सार क्षवा रेवी पड़ती है इसी प्रकार से एक नेता को अक्सर अपने अधीनस्थ कर्मचारियों से संबंधित कठोर निर्णय मी लेने पड़ते हैं जब कभी कठोर निर्णय उचित जान पड़े तो एक नेता को उससे हिचकिचाना नहीं चाहिये यदि आप हिचकिचा गये तो समझिये छि आपका नेतृत्व अअ हो गया यदि कर्मचारियों को यह एहसास हो जाए कि आप कठोर निर्णय लेने से डरते हैं तो फिर बिगड़े हुए कर्मचारियों की हिम्मत वद जाएगी अच्छे कर्मचारी निराश हो जाएंगे क्योंकि उनमें और दूसरे कर्मचारियों में कोई फर्क नहीं रह जाएगा अनुशासनहीनता को यदि कमजोर नेता सहन करता है तो अनुशासनहीनता कई गुना बढ़ जाती है यदि आप अनुशासनहीनता को शुरू में ही कठोरता से दबोच देते हैं तो कुछ समय के लिए काफी

Fig. 15. The output of the classification process; only the top choice for each character has been shown here, \* represents a rejected character, the words that contain one or more incorrect characters are underlined.

## Output after correction phase

या आउटा देने से पूर्व यदि आप अपने सहयोगियों से विचार-विमर्श करते हैं तथा उनके विचारों को जानने के बाद + कोई निर्णय लेते हैं और ठकश {निर्णय} का औचित्य भी उन्हें बताते हैं तो उसका पालन हमेशा ज्यादा अन्ना होगा तानाशाही और मनमानी \*ादा दिन तक नहीं टिकती सलाह-मशविरे के बाद लिये गये निर्णय का पालन सदैव खारा + अला होता है

कई बार माता-पिता को परिवार के हित में अपनी संतान के हित में उन्हीं से संबंधित कुछ कठोर निर्णय भी लेने होते हैं कई बार अनुशासनहीन बच्चे को हास्टल में {भी} डालना पड़ता है कई सार + सभा + देनी पड़ती है इसी प्रकार से एक नेता को अक्सर अपने अधीनस्थ कर्मचारियों से संबंधित कठोर निर्णय मां + लेने पड़ते हैं जब कभी कठोर निर्णय उचित जान पड़े तो एक नेता को उससे हिचकिचाना नहीं चाहिये यदि आप हिचकिचा गये तो समझिये कि आपका नेतृत्व अद हो गया यदि कर्मचारियों को यह एहसास हो जाए कि आप {कठोर} निर्णय लेने से डरते हैं तो फिर बिगड़े हुए कर्मचारियों की हिम्मत बद + जाएगी अच्छे कर्मचारी निराश हो जाएंगे क्योंकि उनमें और दूसरे कर्मचारियों में कोई फर्क नहीं रह जाएगा अनुशासनहीनता को यदि कमजोर नेता सहन करता है तो अनुशासनहीनता कई गुना बढ़ जाती है यदि आप अनुशासनहीनता को शुरू में ही कठोरता से दबोच देते हैं तो कुछ समय के लिए काफी

Fig. 16. The corresponding output after correction; the word is shown in {} if the true word is the second or third choice; incorrect words are underlined; a + mark has been placed next to the word if it has been incorrectly mapped to a dictionary word, \* represents a rejected character.