Error Detection and Correction in Indic OCRs

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS in
Computer Science and Engineering
by Research

by

Vinitha V S
201307548
vinitha.vs@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
December 2017

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Error Detection and Correction in Indic OCRs" by Vinitha V S, has been carried out under my supervision and is not submitted elsewhere for a degree.

05-12-2017

Date

[illegible]

Adviser: Prof. C.V. Jawahar

To

my family and friends

Acknowledgements

First of all, I would like to express my sincere gratitude to Prof. C.V Jawahar, who introduced me to research. His vision and guidance have inspired me to dream bigger and aim higher.

I am blessed to have been taught by the Professors at IIIT-H who have helped imbibe in me curiosity and the urge to excel. I also extend my thanks to CVIT faculty, P. J. Narayanan, Anoop Namboodiri, Jayanthi Sivaswamy, Avinash Sharma and Vineet Gandhi whose presence in the lab creates a motivating environment. My sincere thanks to Dr.Girish Varma for providing me valuable insights and motivation during my work at lab. I am also thankful to Prof. Vineet Chaitanya, who has been generous to give me his valuable suggestions during my research work. I extend my thanks to CVIT lab staff and others including Silar, Phani, Satya, Shiva, Prathima, Rajan and Nandini who were always available for any work/academic related requirements. I would like to thank the lab and the institution for providing adequate facilities for minds to learn and prosper. A special thanks to the IIIT library staff for providing a good reading and learning environment.

I would also take this opportunity to thank my lab mates and friends at CVIT, especially Naveen Sankaran, Karthika Mohan, Rajvi Shah, Praveen Krishnan, Jobin K V, Pritish, Sourab, Thrupthi, Ajeet Kumar Singh, Koustav, Aniket, Suriya Singh, Udit Roy, Minesh Mathew, Aquib Jamal and Varun Bhargavan who always lent a helping hand whenever needed. During my times of self doubt, you have extended your support and helped me put myself back on track. Without my friends, my life at IIIT would not have been easy. I thank Falah, Vidya Naidu and Basil George for making the days memorable.

Finally, I thank my family for believing in me and supporting me in my decision to pursue my dreams.

Abstract

Indian languages have a rich literature that is not available in digitized form. Attempts have been made to preserve this repository of art and information by maintaining a digital library of scanned books. However, this does not fulfill the purpose as indexing and searching the documents is difficult in images. An OCR system can be used to convert the scanned documents to editable form. However, the OCR systems are error prone. These errors are largely unavoidable and occur due to issues like poor-quality images, complex font, unknown glyphs etc. A post-processing system can help in improving the accuracy by using the information about the patterns and constraints in the word and sentence formation to identify the errors and correct them.

OCR is considered to be a problem attempted with marked success in Latin scripts, especially English. This is not the case with Indic scripts as the error rates of various OCR systems available are comparatively high. The OCR pipeline includes three main stages, namely segmentation, text recognition and post-processing. We observe that Indic scripts have complex scripts and glyph segmentation itself is a challenge. The existence of visually similar glyphs also makes the recognition process difficult. The challenges faced in the post-processing stage are largely due to the properties of Indian languages. The inflectional properties of some languages like Telugu and Malayalam and agglutination of words creates issues due to the enormous and growing vocabulary in these languages. Unlike alphabet system in English, Indic scripts follow alphasyllabary writing system. Hence the choice of unicodes as the basic unit of a word is questionable. *Aksharas* which are a more meaningful unit is considered as a better alternative to unicodes. In this thesis, we analyze the challenges in building an efficient post-processor for Indic language OCRs and propose two novel error detection techniques.

The post-processing module deals with the detection of errors in the recognized text and correction of those detected errors. To understand the issues in post-processing in Indian languages, we first perform a statistical analysis of the textual data. The unavailability of huge corpus prompted us to crawl various newspaper sites and Wikipedia dump to obtain the required text data. We compare the unique word distribution and word cover of popular Indian languages with English. We observe that languages like Telugu, Tamil, Kannada and Malayalam tend to have huge number of unique words compared to English. We also observe how many words get converted to other valid words in the language, using the Hamming distance between the words as a measure.

We empirically analyze the effectiveness of statistical language models for error detection and correction. First we use an ngram model for detection of errors in the OCR output. We use *akshara* split words to create a bigram and trigram language model which gives the probability of a word. A word is declared as an error word if it has lower probability than a pre-computed threshold value. For error correction, we replace the lowest probability ngram with a higher probability one from the ngram list. We observe that *akshara* level ngrams perform better than unicode level ngram models in both error detection and correction. We also discuss why the dictionary based method, a popular method used in English, is not a reliable solution for error detection and correction in case of Indic OCRs. We use a simple binary dictionary method for error detection, wherein if the word is present in the dictionary, it is tagged as a correct word and error otherwise. The major bottleneck in using a lexicon is the enormous words in the languages like Telugu and Malayalam. In error correction, we use Levenshtein and Gestalt scores to select the candidate words from the dictionary for replacement of error word. Inflection of words causes issues in selecting the correct words as the candidate list consists of many words which are close to the error word.

We propose two novel methods for detecting errors in the OCR output. Both the methods are language independent and does not require knowledge of language grammar. For detecting the errors in the OCR output, the first method proposed uses a recurrent neural network to learn the patterns of errors and correct words in the OCR output. The second method is using a Gaussian mixture model based clustering technique. Both methods use a language model of unicode as well as *akshara* split words in creating the features. We argue that *aksharas* are a better choice as the basic unit of a word than unicode. An *akshara* is formed by the combination of one or more unicode characters. We tested our method on four popular Indian languages and report an average error detection performance above 80% on a dataset of 5K pages recognized using two state of the art OCR systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Related Work

Indian languages have a long history. There is also rich literature collection in these languages. A bulk of this literature is available only in the form of printed text. If we do not preserve the printed documents we may lose this precious information. The best way to preserve this information is to store the data in digital format. Hence the Optical Character Recognition (OCR) problem for Indic scripts is of paramount importance. An OCR can be used to convert scanned document images to editable electronic documents [2]. Although efforts have been made to build OCRs for Indic scripts, the effectiveness of the techniques employed in English OCRs is not yet reproducible in Indic scripts. For instance, Tesseract [3] and ABBYY [4] provide accuracies exceeding 99% in English for reasonable quality of documents. This is not the case with Indic scripts. Indic OCRs are more error prone than their Latin counterparts. A deeper understanding of the problem is required to improve the accuracy of Indic OCRs and make them suitable for practical applications. In this thesis, we first try to enumerate the major bottlenecks in improving the OCR accuracy. We also explore the complexity of Indic scripts, the nature of fonts and their encoding standards.

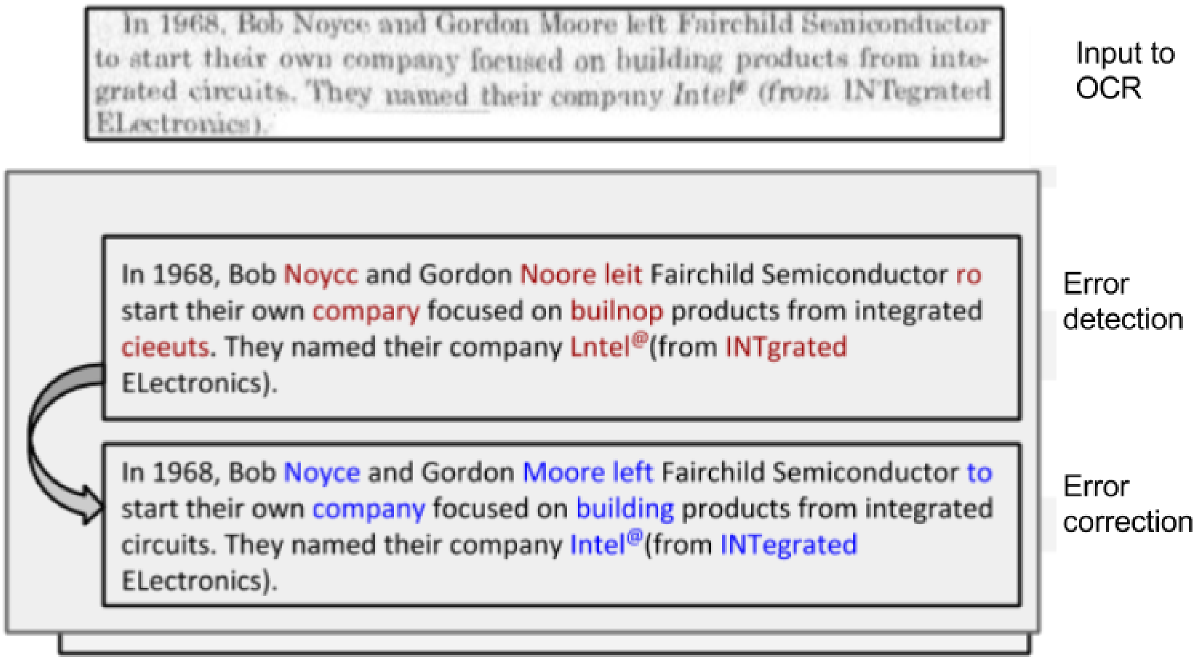
A basic OCR system pipeline includes three major stages.

1. Pre-processing
   This step involves preparing the image to ensure that the best quality image is input to the next stage. The scanned pages or images usually contain noises which hinder the recognition process. Methods used for pre-processing include skew correction, binarisation, layout analysis, script recognition etc. The segmentation of characters, words, lines etc. are also done in this stage.

2. Recognition
   In this stage, the text in the image is recognized and the image is converted into editable text format. Image features like HOG,SIFT, profile based features etc. are used for recognition. A classifier then assigns the glyph in the image to the closest matching glyph in the language. Popular classifiers used for this purpose include nearest neighbour, SVM and neural networks etc.

3. Post-processing
   This is the stage where the errors made in the recognition phase is identified and corrected. The

two stages in post-processing are error detection and error correction. The knowledge of grammar of the language and co-occurrence frequencies can help in this process. For example, if the word ”I am” is recognized as ”1 am”, the occurrence of former word sequence has more chance of existing together than latter. Statistical language models are widely used to obtain this co-occurrence probabilities of words or characters in words. This information along with the information about the characters which can be confused with each other can help us identify the mistakes and correct them. Figure 1.1 shows how post processing is done on the OCR recognized text. The stages of error detection and error correction and its effects on the recognized text is shown in the Figure.

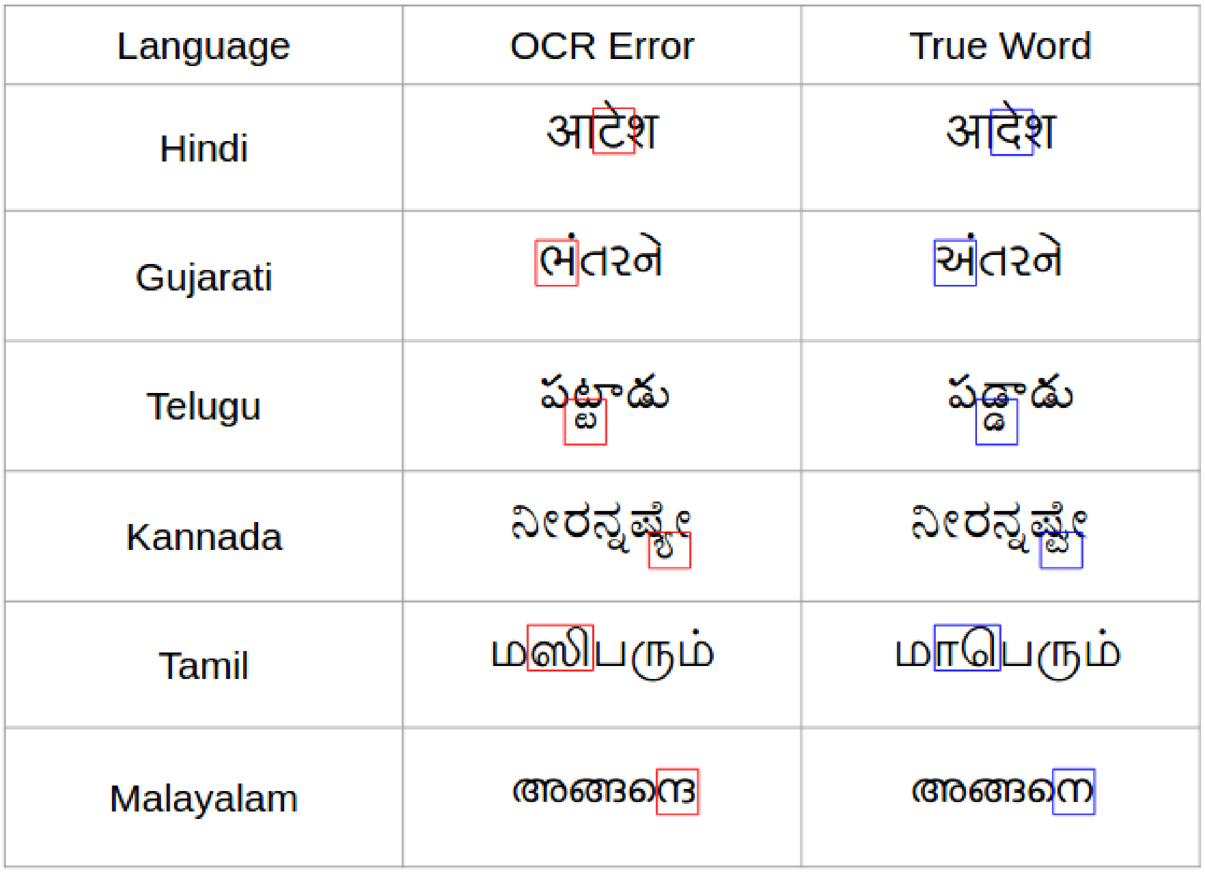

In 1968, Bob Noyce and Gordon Moore left Fairchild Semiconductor to start their own company focused on building products from integrated circuits. They named their company Intel® (from INTegrated ELectronics).

In 1968, Bob Noycc and Gordon Noore leit Fairchild Semiconductor ro start their own compary focused on builnop products from integrated cieeuts. They named their company Lntel®(from INTgrated ELectronics).

In 1968, Bob Noyce and Gordon Moore left Fairchild Semiconductor to start their own company focused on building products from integrated circuits. They named their company Intel®(from INTegrated ELectronics).

Figure 1.1: Figure shows the text recognized by the OCR, passing through different stages of the post-processing module. The text recognized by the OCR contains errors which are detected in the detection stage. The error words detected are shown in red color. In the error correction stage, these error words are corrected(shown in blue color).

Figure 1.2 shows some of the errors which occurred during the OCR recognition in a 5K page data set [5]. While observing the errors in red boxes, and its corresponding correct characters, it is understood that many errors occur due to the visual similarity of the characters in Indic scripts. For example, in Hindi and Malayalam languages shown in Figure 1.2, the error has occurred due to the visual similarity of the characters shown in red and blue bounding boxes. The error in Kannada OCR shown in the Figure has occurred due to the complex representations of characters in the language. Two consonants when occur together are sometimes represented by placing the second consonant below the first one. This causes the size of the characters to be smaller than when presented normally, leading to errors. During such representations, the OCR is unable to distinguish the minor differences in the glyphs. This complicates the recognition process. A similar error is seen in Telugu also.



Language	OCR Error	True Word
Hindi	आटेश	आदेश
Gujarati	ભંતરને	અંતરને
Telugu	పట్టాడు	పడ్డాడు
Kannada	ನೀರನ್ನಷ್ಯೇ	ನೀರನ್ನಷ್ಟೇ
Tamil	மஸிபரும்	மாபெரும்
Malayalam	അങ്ങന്ദെ	അങ്ങനെ

Figure 1.2: The figure shows some errors which occurred during OCR recognition process. The red boxes show the error characters and blue boxes show the intended characters.

1.1 Overview of Popular Recognition Systems

We have used the output of three different recognizers in our experiments, namely an SVM based [6], RNN based [7] and Tesseract OCR [8]. Though the main functionality of these OCRs are the same i.e recognition, they differ in the method used for recognition. In the SVM based OCR, the shape classifier used is a support vector machine. SVM based classification is a supervised learning method, which separates the classes using a hyperplane or a set of hyperplanes between them. The classes here are the isolated characters in a language. Hence segmentation of the glyphs or characters is the first task done. These are then passed to the recognizer for classification. The segmentation is critical here because any issues in segmentation can cause the recognizer to incorrectly recognize the words. The second recognizer is a neural network based shape classifier which models the problem of recognition as transcription. This recognizer does not need segmentation of individual glyphs or characters before recognition. The bidirectional RNN architecture has the ability to access long range context, learn sequence alignment and work without segmenting the individual glyphs. These recognizers are getting much recognition in recent times due to the superiority in performance. Thirdly, we use Tesseract, a free-software, originally developed by Hewlett-Packard Laboratories and now maintained by Google. It uses a two-pass process for recognition. The first pass makes an attempt to recognize each word. Each word which is satisfactorily recognized is passed to the second stage consisting of an adaptive classifier. Recently Tesseract has made improvisations to exploit the use of neural networks for recognition.

1.2 Motivation

India has a rich collection of literature spread across various languages, civilizations and time periods. If we do not take effort to preserve this treasure, inherent vice will eventually destroy the books which have recorded our journey through time. Digitizing these texts can prevent their loss through eventual decay. Optical character recognition (OCR) is a technology which has helped in this regard. OCR systems are generally not perfect and at times can falsely identify the scanned text, causing misspellings and linguistic errors in the output. Since the quality of images cannot be guaranteed at all times, the performance of the OCR cannot be the improved by improving the classifier accuracy alone. A post-processing system added to a recognizer can help in this regard. It can identify and rectify those errors created in the recognition phase. The OCR pipeline available today is not at par with English and other Latin language OCRs. To address this problem, we first need to identify and understand the challenges unique to Indic scripts and ways to overcome them.

1.3 Contributions

In this thesis, we have examined the major challenges involved in post-processing of OCR output in Indic scripts. Extending the work done by Sankaran and Jawahar [9] and Sankaran [10], we explore the reasons why some of the post-processing methods employed with success in case of English do not work well with Indic scripts. We analyze the text corpus in major Indian languages to find the unique word coverage. Also we understand how the words in languages like Hindi can be converted to other valid words by changing just one glyph in the word. We have compared various language statistics for Indic scripts like unique word count, word cover, number of words at particular Hamming distance, etc. with those for English by including more languages and larger corpus in most languages. Since we required ample amount of data to create a good language model, we have created a huge corpus of data crawled from various online sources like newspapers and Wikipedia dump. This was necessary as the available corpus for Indic languages is nowhere near the huge corpus available for English, for example the BNC corpus [11]. The crawled corpus contained many undesirable characters such as other language words, typographical errors etc. These were carefully removed and a large corpus which is fairly clean is created. We also perform experiments to demonstrate why error detection and correction using popular methods like dictionary and statistical language models do not work well in Indic scripts. Finally, we have also proposed two novel error detection techniques which are language agnostic. The methods use bigram and trigram language models of *aksharas* in words as features to create the models. We have used a bidirectional recurrent neural network based classifier to learn two classes of words, namely correct and error words produced by the OCR output. In the second method, we have used Gaussian mixture model based clustering method in which depending on which cluster the word is close to, the word is tagged as a correct word or error word.

1.4 Thesis Organization

In this thesis, we have first analyzed in detail, the various issues due to which we are not able to produce high accuracy OCR output. In Chapter 1, we look into the previous works done in English for performing post-processing of the OCR output. We then present the related works done in Indic language OCR post-processing. Chapter 2 gives an overview of the multiplicity of languages and scripts in India. This Chapter also mentions about the two different encoding standards popularly used in Indic scripts, namely ISCII and Unicode encoding. Next we explore *akshara* as a more meaningful basic unit of a word and compare with unicode. In Chapter 3, we analyze various issues which pose challenges in creating a highly accurate post-processing system in Indic language OCRs. We analyze the unique word counts, word cover statistics, words at a Hamming distance etc. In Chapter 4, we empirically analyze the effectiveness of traditional approaches for error detection and correction in Indic scripts, namely, a statistical language model based method and a dictionary based method for Indic scripts. We also compare its performance with English. Finally, in Chapter 5, we showcase two novel and successful error detection approaches which are language agnostic. We show the results of these methods in four major Indic scripts.

1.5 Related Work

1.5.1 Post-processing in English OCR

Most early spell checking systems relied on dictionary based method for spell correction [12, 13]. The use of a dictionary can be thought of as the most straight forward approach for detecting the erroneous words in the OCR output [14, 15]. The presence/absence of a word in the dictionary is used to check the validity of the word. The availability of a nearly complete dictionary can give good results in this method, if one exists! Many spell checking and correction systems make use of this binary dictionary method efficiently [16]. With the advances in natural language processing techniques, especially the use of statistical language models(SLM) and noisy channel models opened doors for alternate methods for error detection and correction. In [17], Tong explores the use of statistical language models using letter ngrams, character confusion probabilities and word-bigram probabilities. The work presents letter ngrams and character confusion probabilities to find possible word candidates for replacement of an error word. A word-bigram model along with Viterbi algorithm is used to predict the word which would better place itself in the sentence. In [18], stochastic error-correcting parsing using Viterbi algorithm has been able to produce significant results in post-processing. Successful attempts have been made to make use of Shannon's [19] noisy channel model for error correction. In [20], Brill and Moore use a noisy channel model with source model and channel model (error model) to find the most likely word in the dictionary which may be converted to an error word. For the detection of real word errors, a trigram based noisy-channel model is employed in [21]. In noisy-channel model, the goal is to find the intended word given an erroneous word. In some cases, for example, when the error word is a proper noun which is not present in the dictionary, it may be better to accept the error word as the intended word rather than attempting to find a replacement word from the dictionary. Packer et al. [22] used Hidden Markov Models (HMMs) for detecting OCR errors. In [23], the author uses part-of-speech trigrams combined with Bayesian methods for context sensitive spelling correction. In [24], Smith uses a shape classifier model, a word ngram model and a binary ngram dictionary model to detect the errors in English. In [25], huge data available from Google is used to detect and correct misspelled words including non word and real word errors in the OCR output text. Non word errors are the errors which are invalid words in the language. Real word errors are errors which are valid words in the language. Detection and correction of real word errors is a more challenging task than non-word errors.

1.5.2 Post-processing in Indic OCRs

The earliest works in error detection and correction were mainly focused on languages like English [17] and few other Latin languages. There have also been a few attempts made to develop post-processing modules for Indian languages to improve the overall OCR accuracy. The scope of applying part of speech taggers and other NLP techniques in Indic OCR post-processing is restricted due to the unavailability of such reliable models for these languages. Hence we can find most early works pivoting around language specific features such as the morphology of the words or size and shape of the words. In [26], a shape based post-processing system for Gurumukhi OCR was employed. Here the size and shape of Gurumukhi words were used to create partitions of words. The visual similarity between words is used to correct them. In [27], an error correction system for Bangla language is proposed. The approach involves morphological parsing of the word to split the word into root and suffix. Then a check is done to know if the root and suffix part of the word can exist together grammatically. In [28], a multi-stage graph based reasoning, aided by sub-character level language model is used to correct errors in the OCR output in Malayalam. Here unicode characters are used as the basic unit of a word to create the language model. Sankaran [10] applies various techniques for error detection in two major Indian languages, namely Malayalam and Telugu. This work compares various approaches for error detection like using a simple dictionary look up along with an SVM classifier to make predictions. An ngram based error detection scheme is also explored in which each ngram in an input string is looked up in a pre-compiled table of ngram statistics to ascertain its existence or its frequency. We can observe that early post-processing works were focused on a particular language. One of the reasons for this is that the techniques applied for error detection and correction required some knowledge of the language used. For instance in [29], splitting the word into root and suffix will be difficult without knowing the grammar details of Bangla. We will see in Chapter 2 that most Indian languages belong to two major language families, Indo-Aryan and Dravidian. These two languages families are similar in the sense that there is a significant overlap of the words used and there exist a correspondence between the unicodes in these languages. However, the properties like inflection due to gender, agglutination etc. displayed by these languages are different. The language properties like agglutination is severe in some languages like Telugu and Malayalam, but not a major issue in languages like Hindi. On the other hand, inflectional variations due to gender is prominent in Hindi and Gujarati. Hence the post-processing techniques also need to be carefully chosen. An attempt to develop a generic solution which could be applied to all Indian languages was first seen in [10]. Though this work explores only two languages, the technique does not employee any language specific method for error detection and is language agnostic.

Chapter 2

Indian Languages: Overview

2.1 Indian Language Structure

Indic scripts have their origin traced back to the ancient *Brahmi* script. According to the census of India in 2001[1], there are 1365 rationalized mother tongues, 234 identifiable mother tongues and 22 major languages in India. It also identifies 27 native languages which are mostly dialects/variants grouped under the language Hindi. The Census of India reports only those languages which have more than 10,000 native speakers. Hindi is the mother tongue for 41% of Indians, followed by Bengali (8%) and Telugu (7%). Nepali, Sindhi, Konkani, Dogri, Manipuri, Bodo and Sanskrit are spoken by less than 1% of Indians. Figure 2.1 shows the percentage of native speakers of some scheduled languages in India. The number of speakers of Bodo and Sanskrit is negligible and is omitted in the Figure. There are

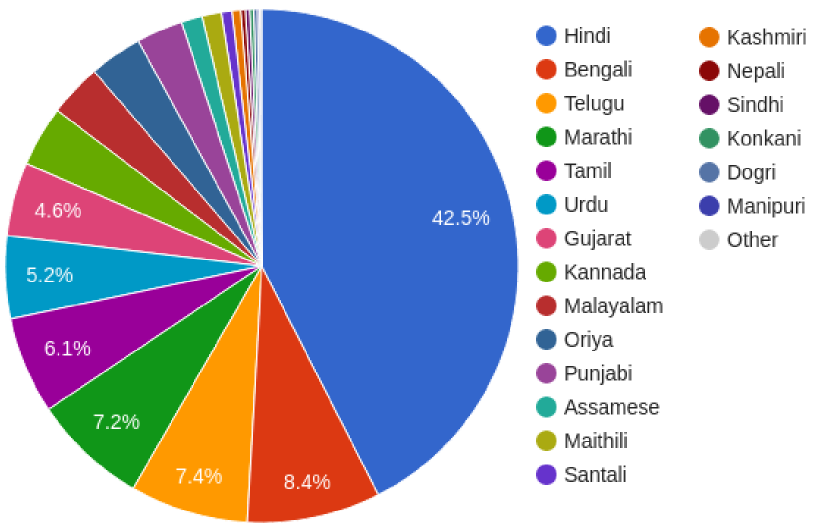


Figure 2.1: Percentage of native speakers of the scheduled languages in India, according to 2001 Census of India [1]

two major language families in India, namely Indo-Aryan and Dravidian language family. About 75% of the population are speakers of Indo-Aryan languages which includes Hindi, Bengali, Marathi, Urdu, Gujarati, Punjabi, and Assamese. Dravidian language family is the second largest language family accounting for some 215 million speakers, or approximately 20% of the population. Telugu, Tamil, Kannada and Malayalam have the most number of speakers and is predominant in the southern part of India. The huge diversity poses challenges in the attempts for building a robust OCR and post-processor. The Figure 2.2 shows the word 'yamuna' in different Indic scripts.

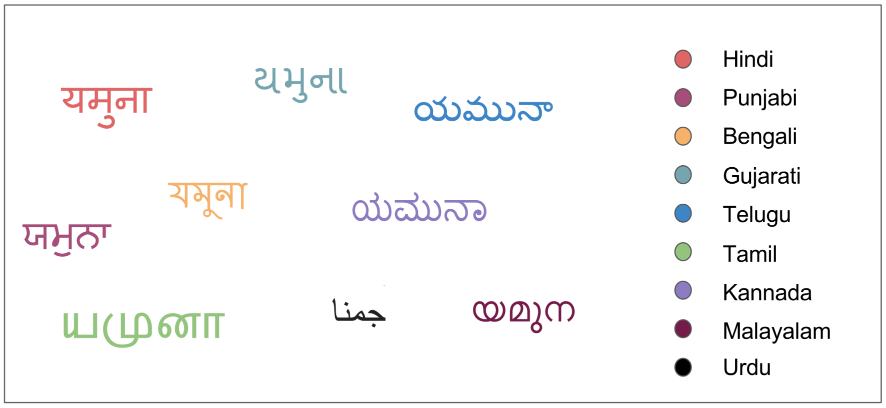


Figure 2.2: Figure shows the word 'yamuna' written in different Indic scripts namely Hindi, Punjabi, Bengali, Gujarati, Telugu, Tamil, Kannada, Malayalam and Urdu

2.2 Encoding standards

The two popular encoding standards available for Indic scripts are Indian Standard Code for Information Interchange (ISCII) and Unicode encoding [30]. ISCII is an 8-bit encoding that uses escape sequences to indicate the particular Indic script represented by a following coded character sequence. Unicode is now widely used as the encoding standard and ISCII is now rendered largely obsolete. According to the Unicode Consortium, except for a few minor differences, ISCII and unicode correspond directly. Unicode is designed to be a multilingual encoding that does not require any escape sequences or switching between scripts. For any given Indic script, the consonant and vowel letter codes of Unicode are based on ISCII. ISCII allows control over character formation by combining letters with the characters *nukta, inv* and *halant*. Unicode provides similar control with the Zero Width Joiner (ZWJ) and Zero Width Non Joiner (ZWNJ) characters. It is notable that in some cases, there can be more than one ways in which a character can be represented in Unicode. For example, the Malayalam glyph represented by hexadecimal value '0xd7b' can also be written as a combination of '0xd28','0xd4d' and '0x200d'.

2.3 Dilemma: What is the basic Unit?

In Indian languages, the question of choosing the basic recognizable unit of a word is debatable. Each character or glyph has a unicode value associated with it. Decomposition of a word into its constituent unicode characters certainly ensures atomicity, but fails to give insight as to how these bigrams or trigrams can build a meaningful word. This means that the bigram or trigram unicodes may not really say anything about the correctness of a word. This prompts us to go for the *akshara* level splitting of the word.

2.3.1 Akshara

Indic languages are syllabic in nature and follows *abugida* or alphasyllabary writing system [31]. The alphasyllabary writing system differs from the alphabet writing system (followed by languages like English). In alphabet system, vowels have status equal to consonants whereas in alphasyllabary system, consonant-vowel sequences are the basic unit of a word. This also implies that when in English a word can begin with any of the alphabets, in Indic scripts it is not allowed. Each unit is based on a consonant letter and vowel notation is only secondary. In Indic scripts the basic unit of a word is called *akshara*. *Akshara* is the basic writing unit in Indic scripts which is also an orthographic representation of a speech sound in Indian language [32]. An *akshara* follows the pattern $C * V$, where C is a consonant and V is a vowel. All scripts have their own number shapes also. Most scripts now follow Western punctuation, but there are also some special punctuation, for example DANDA marks the end of a sentence in Hindi. *Aksharas* are similar to syllables in English language. In certain languages like Malayalam, a syllable may be composed of more than one *akshara*. We would be using the terms *akshara* and syllable interchangeably.

2.3.2 Unicode Representation

Unicode character encoding standard, defined by the Unicode Standard [30], is widely used to represent Indic characters today. In Indic scripts, the unicode values are not associated with each *akshara*, but several unicode characters may be present in one *akshara*. This is shown in Figure 2.3 in which the constituent *aksharas* and unicodes of a Malayalam word is seen.

The question now becomes which unit should be considered as the basic recognition unit of a word in Indic scripts. As against the alphasyllabary system, the alphabet system has a one-to one mapping for each alphabet to a unicode value. In alphasyllabary system, each glyph has a unicode value associated with it, hence an *akshara* has multiple unicode values, one for each of its constituent glyph. Following the post-processing works in English and other Latin languages which use alphabet system, it is tempting to consider unicode values as the basic recognition unit of a word. The primary issue with unicode is that some unicode values like *halant* in Hindi and dependent vowel signs does not have any existence alone and can only be used along with a consonant. *Akshara* on the other hand, is a phonological unit which

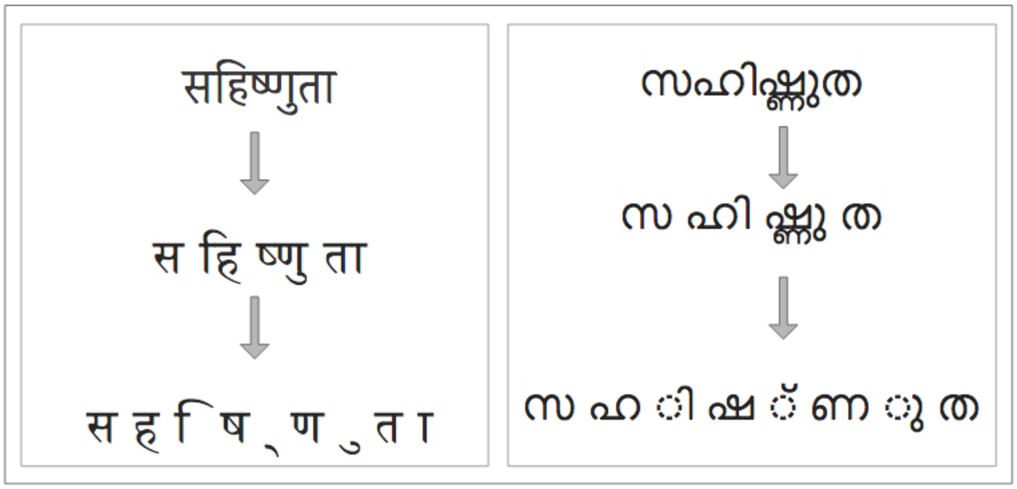


Figure 2.3: The figure shows the *akshara* and unicode level splitting of words in Hindi and Malayalam. The *akshara* level split gives components each of which are composed of one or more unicode symbols. In the unicode level split(bottom level) it can be observed that the unicode character *halant*, is not visible in the *akshara* level split or in the original word.

contains more information regarding the word formation. In order to split words into *aksharas*, a simple regular expression which recognizes the pattern of zero or more consonants followed by a vowel can be used. For error detection in OCR output, we find that syllable level splitting better suits our requirement. Word formation is related to morphological as well as phonological features [33]. Syllables provide phonological information and are widely used in speech recognition systems [34].

In our work, we have used *akshara* as the basic unit of a word. While considering language model creation, there were two options: 1)using unicode level splitting of words and 2) *akshara* level splitting of words. We have used *akshara* level splitting since our goal is to identify the patterns in correct word formation and incorrect word formation. In our experiments, we used both unicode and *akshara*s as basic unit of a word. However, we prefer to use *akshara* over unicode. The first reason is that *akshara* serves as a more meaningful unit of recognition as it captures a valid phonological unit. Secondly, each language has a nearly finite list of valid *aksharas*. Presence of an error in the word creates invalid *aksharas* to be formed. This calls out the presence of error in the word.

Chapter 3

Challenges in Post-Processing in Indic OCRs

Errors generally occur in the OCR output when the OCR mis-recognizes a glyph or a grapheme cluster with a visually similar glyph or grapheme cluster. To understand the challenges of post-processing, we first need to understand the types of errors which occur in the recognizer output.

3.1 Errors in OCR Output

There are two main categories of OCR errors, namely real word error and non-word error. A real word error occurs when the incorrectly recognized word is another valid word in the language. For example if the OCR recognizes the word 'fine' as 'tine' which is another valid word, then a real word error occurs. On the other hand, if 'fine' is recognized as 'iine', the error created is called a non-word error. Figure 3.1 (a) shows a real word error in Telugu language where a word is incorrectly recognized, and the output is also a valid word. Real word errors are difficult to detect unless we have enough context information because these words belong to the dictionary but are not the intended words. Figure 3.1 (b) shows an example of a non-word error in Gujarati language, where a unicode glyph is mis-recognized as a group of grapheme.

In [14], Kukich mentions about the difficulty of context depended word correction in English, without full-blown NLP capabilities such as robust natural language parsing, semantic understanding, pragmatic modeling and discourse structure modeling. Context dependent error correction is still an open problem in Indian languages as the advances in NLP are not sufficient to aid in improved error correction. Figure 3.2 below shows some real word errors and non-word errors in OCR output. In case of both real word and non-word errors, run-on or split words errors which occur when errors cross word boundaries can complicate the post-processing. Insertion of new characters (insertion error) , deletion of any valid character in the word (deletion error) and framing errors (error which occur when a single letter is substituted for multiple letter sequence or vice-versa) can all result in changing the length of the intended word. Hence, error correction of words with deletion and insertion error is much more challenging than a substitution error.

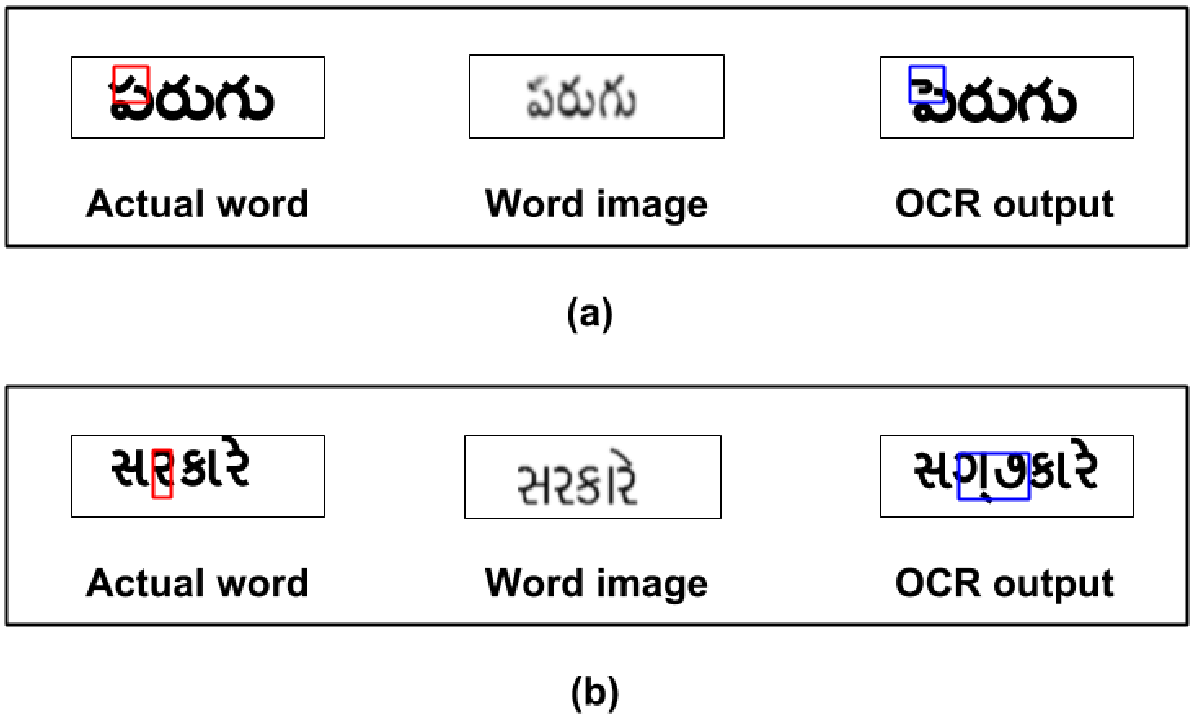


Figure 3.1: Types of OCR errors: Figure (a) shows a word in Telugu language, its image and the mis-recognized OCR output which is another valid word in Telugu. A unicode character (in red box) is recognized as another unicode character (in blue box). Figure (b) shows a Gujarati word, the image of which when recognized, gives an invalid word as output. The unicode character (in red box) is recognized as a group of graphemes (in blue box)

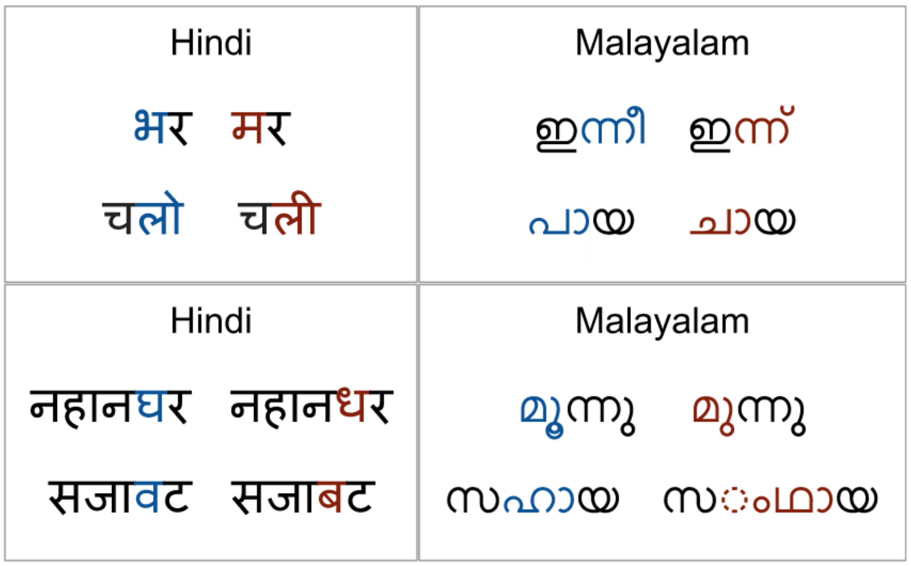

Hindi
भर मर
चलो चली
Malayalam
ഇന്നി ഇന്ന്
പായ ചായ
Hindi
नहानघर नहानधर
सजावट सजाबट
Malayalam

Figure 3.2: Figure shows some real word errors(top box) and non-word errors(bottom box) in Hindi and Malayalam. In Hindi, inflection due to gender causes existence of words with variations in the unicode glyph at the end of the word. Such real word errors can be difficult to detect without applying grammar rules.

3.1.1 Types of OCR Errors

We analyzed the errors occurring on the dataset [5]. We considered substitution, insertion and deletion errors as they give a more fine grained picture of the nature of errors occurring. We calculate the word error rate as well as the character error rates.

Word error rate is the percentage of erroneous words. Character error rate is calculated with respect to unicode characters. Character error rate shows the percentage of unicode characters that have been incorrectly recognized by the OCR. In order to compute the error rates, we have to first align the recognized word sequence with the reference (ground truth) word sequence using dynamic string alignment. Note that the word error rates are higher compared to character error rates. This is natural as even a single character in a long word can create a word error. We also observe that different Indian languages have varying complexity with respect to errors. The word and character error rates in Telugu, Gujarati and Kannada are much higher than other languages like Gurumukhi, Malayalam and Bangla. The complexity of Telugu (explained later in Section 3.2.3) and Kannada scripts with nearly touching glyphs can complicate the recognition process and create errors. The existence of glyphs which are difficult to visually distinguish also serve as a possible reason for this increased error rates in these scripts.

	Telugu	Gujarati	Kannada	Tamil	Hindi	Malayalam	Bangla	Gurumukhi
Test data size (words)	635,669	846,469	420,365	702,864	1,546,148	818,938	454,854	1,541,798
Word error rate	73.77	63.92	63.58	49.21	34.06	31.39	26.17	24.92
Substitution error rate	59.95	53.54	50.38	43.24	29.95	25.65	19.06	17.66
Insertion error rate	12.92	10.11	11.52	2.79	1.91	5.55	6.69	7.05
Deletion error rate	0.89	0.27	1.68	3.18	2.20	0.19	0.42	0.21

Table 3.1: Figure shows OCR word error rates for different Indian languages for an SVM based recognizer

We can observe that among the three types of errors, namely substitution, insertion and deletion errors, the incidence of substitution error is much higher than the other two types of errors. This is possibly because of the existence of visually similar glyphs. But most errors in the OCR output is due to the recognition of *matras* incorrectly. This occurs due to the confusion between *matras* ending with long and short vowel sounds. This issue causes substitution errors. Insertion errors occur mainly due to the mis-recognition of a single/group of unicode characters with other unicode characters whose length exceeds the length of original characters. Another issue occurs when OCR omits the unicode characters such as *anusvara* (represented by a dot above the letters) which is used to indicate the plural form of

	Gujarati	Telugu	Tamil	Kannada	Hindi	Bangla	Gurumukhi	Malayalam
Test data size (Characters)	3,990,243	4,385,500	5,545,503	3,069,767	5,554,255	2,078,260	5,517,503	7,798,388
Character error rate	37.25	28.86	20.65	20.24	16.94	8.96	8.37	5.99
Substitution error rate	18.14	16.58	7.30	9.78	8.50	4.11	3.86	2.77
Insertion error rate	16.43	7.66	4.44	4.72	3.52	3.04	2.74	1.69
Deletion error rate	2.68	4.61	8.91	5.74	4.92	1.81	1.77	1.53

Table 3.2: Figure shows OCR character error rates for different Indian languages for an SVM based recognizer

some words. The omission of this glyph results in deletion error. The issue of similar *matras* and *anusvara* is better explained in Figure 3.3. It is observed that deletion errors are the least occurring of all the three types of errors and good quality image can reduce deletion errors significantly.

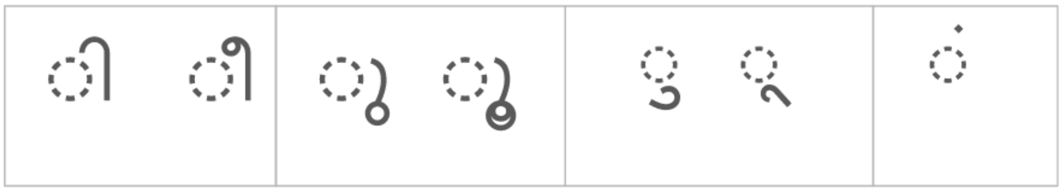


Figure 3.3: Figure shows four boxes which shows unicode characters responsible for substitution and deletion errors in Malayalam and Hindi. The first box shows the unicode character pair of a dependent vowel in Malayalam. The unicodes in the box correspond to long and short vowel sounds. The second box also shows another visually similar unicode pair in Malayalam. The third box shows a similar pair of unicode characters in Hindi. The last box shows *anusvara*, a unicode character used in Hindi, which is often missed by the OCR resulting in deletion error.

In order to understand the state of current OCRs we have obtained statistical results of errors produced by two different OCRs, namely Tesseract [8] and RNN OCR [7] on a subset of the dataset used previously [5]. The character error rate details of RNN OCR is shown in Table 3.3 and word error rate details are shown in Table 3.4. Similar statistics for Tesseract OCR is shown in Table 3.5 and Table 3.6. We observe that the OCR using RNN based classifier performs better in the set of books we have used for recognition. The word level accuracy of RNN based OCR is significantly better in languages like Telugu and Malayalam. In case of Gujarati, both Tesseract and RNN based OCR have struggled to recognize the words correctly. An OCR can perform better if the recognition model is better adapted to the font used for testing. The RNN based OCR has an upper hand in this case.

	Malayalam	Kannada	Telugu	Hindi	Gujarati
Test data size (Characters)	238,213	179,640	372,202	238,055	185,619
Character error rate	1.60	2.59	1.83	8.84	13.40
Substitution error rate	1.03	1.28	1.02	1.76	4.41
Insertion error rate	0.41	0.72	2.36	2.86	7.44
Deletion error rate	0.17	0.59	11.63	4.22	1.54

Table 3.3: Figure shows RNN OCR character error rates for different Indian languages

	Malayalam	Kannada	Telugu	Hindi	Gujarati
Test data size (Words)	25,423	25,741	55,703	70,207	39,321
Word error rate	9.05	11.01	6.89	19.05	24.65
Substitution error rate	8.27	8.84	6.44	13.71	19.03
Insertion error rate	0.74	2.05	0.66	0.66	5.56
Deletion error rate	0.03	0.11	0.49	4.67	0.06

Table 3.4: Figure shows RNN OCR word error rates for different Indian languages

However we observe that the Tesseract OCR produces less deletion errors in languages like Hindi. In other languages, the deletion error rates are comparable. The major errors produced by the Tesseract

OCR is due to substitution and insertion errors. The complexity of Indic scripts and similar looking glyphs have resulted in these errors.

	Malayalam	Kannada	Telugu	Hindi	Gujarati
Test data size (Characters)	238,213	179,640	372,202	238,055	185,619
Character error rate	34.83	9.71	18.84	16.53	15.60
Substitution error rate	19.24	5.16	11.34	5.54	7.41
Insertion error rate	11.91	3.85	6.83	7.27	6.70
Deletion error rate	3.68	0.70	0.66	3.72	1.50

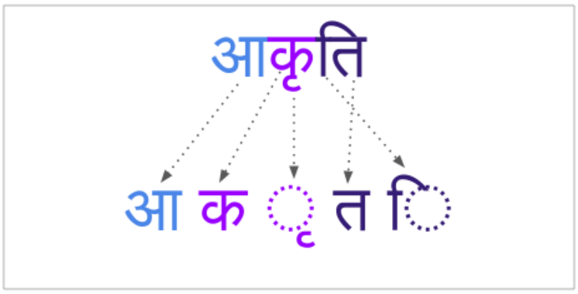
Table 3.5: Character Error Rates of Tesseract OCR

	Malayalam	Kannada	Telugu	Hindi	Gujarati
Test data size (Words)	25,423	25,741	55,703	70,207	39,321
Word error rate	91.37	27.94	47.83	26.44	28.47
Substitution error rate	86.31	25.66	45.22	21.03	23.76
Insertion error rate	5.05	2.06	2.47	2.09	4.13
Deletion error rate	0.01	0.21	0.15	3.33	0.58

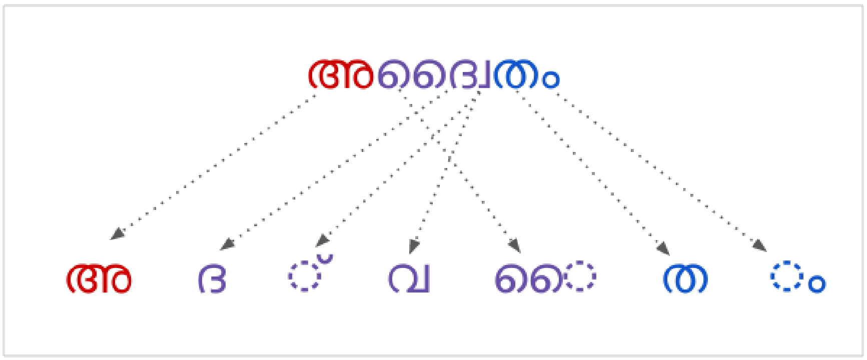
Table 3.6: Word Error Rates of Tesseract OCR

3.1.2 Analysis of Indic OCR Errors

In the previous section, we have seen the common types of errors which can occur in the Indic OCR output. In addition to the errors created due to the mis-recognition of characters/glyphs, there are some errors which occur due to the way segmentation and recognition is done in OCRs. In this section we look at some such errors which are not usually present in Languages like English. These errors occur specifically due to ordering of unicode to form a word. The type of segmentation used in OCR pipeline has a major role to play in this error. In the OCR pipeline, the task of the recognizer is to classify the characters in a word using a pre-trained classifier. Different classifiers like KNN and Multilayer perceptron (MLP) were used initially for this recognition task [35]. Many classifiers using Support Vector Machine (SVM) also became popular later [6]. In all these classifiers, segmentation of connected components was an indispensable and critical task. Unlike English where each character recognized in a word had to be placed next to the previously recognized one, in Indic scripts the ordering of unicodes recognized had to be done to generate meaningful text. This means that the unicodes if simply put in the order of recognition may not give valid words in most Indic scripts. An example of such ordering needed in Hindi and Malayalam is shown in Figure 3.4.



(a)



(b)

Figure 3.4: The Figure (a) shows a word in Hindi in which unicode rearrangement is required to render the word correctly. Figure (b) shows a similar case in Malayalam. Each *akshara* unit in a word is shown in single color.

Each *akshara* in the word is shown in unique color. The unicodes corresponding to the *akshara* in the word is shown using arrows. In Figure (a), the dependent vowel sign is placed before the consonant in the Hindi word. In unicode order, it appears after the consonant (check the intersection of arrows). The Malayalam word shown in Figure (b), shows another such example where arranging unicodes is critical to render the word correctly. When we observed the errors created by the SVM based shape classifier in [6], we found that most of the errors which were created had violated the rules of arrangement of unicodes in the language. A few examples of such errors are seen in Figure 3.5.

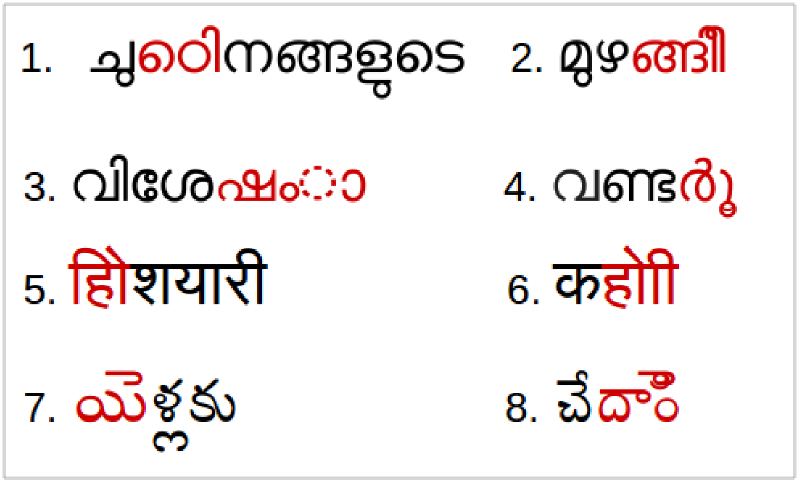


Figure 3.5: The Figure shows some errors which have distorted arrangement of unicodes in the words. The colored part shows the region where error occurred due to incorrect ordering

In word 1 in the Figure 3.5, a Malayalam word is shown in which two different dependent vowel signs are attached to a consonant, making it an unacceptable word. In the word 2, the same vowel sign is repeated twice. The word 3 also shows another Malayalam word wherein an *anuswara* is followed by a dependent vowel sign due to an insertion error. Since the vowel sign has no consonant to attach itself with, it is seen along with a circular dotted symbol. In words 5 and 6, similar errors in language Hindi (Devanagari script) is seen. In word 5, the position of the dependent vowel is changed. Instead of attaching to the second consonant, the dependent vowel is seen with the first consonant itself. In word 6, three different dependent vowel signs are seen after a consonant due to insertion error. Similar errors in Telugu are seen in words 7 and 8. Recognition using Hidden Markov Models (HMM) [36] made things simpler as now we could do away with the symbol/glyph segmentation task. HMM based approaches explicitly tried to address this problem by defining the input as a sequence of feature vectors. The use of Recurrent Neural Network (RNN) [7] based recognizer modules considered the problem as transcribing a set of feature vectors to output labels. When we observed the errors in an RNN based OCR, we found that the errors due to incorrect arrangement of unicodes were completely eliminated. The resulting errors were mainly due to issues like insertion, deletion or mis-recognition of symbols.

3.2 Challenges

Detecting errors can be a simple task if there is a nearly complete dictionary at hand. Many Indian languages are highly inflectional in nature which means that a word can take multiple forms due to gender, size or contextual factors. Inflectional nature of these languages make them morphologically rich. These languages are also agglutinative in nature which causes two or more words to combine to form another valid word. Inflection and agglutination results in massive increase in the number of words in the language. Hence maintaining a nearly complete dictionary is not a solution, but can only be a supplementary strategy. In this section we discuss various issues which pose challenges in post processing.

3.2.1 Closely Related Works

There are challenges offered by Indic scripts which are unique to these languages. A detailed work in this direction have been in [37] which presents statistical analysis of ten major Indian languages. The paper mentions that a study analyzing the language statistics could not be done previously due not the non-availability of a balanced corpus. The authors have used resources like CIIL corpus [38], newspapers and other books to create a balanced corpus. In this work, an estimate of the words in the corpus along with the frequency is made available. The study compares behaviour of morphologically rich language Telugu, which has a large unique word count with languages like Hindi in which the words are repeated often. The paper also makes a significant analysis of the number of words required to cover a certain percentage of the corpus. This gives an idea of the enormity of the words in a language. It is observed that Dravidian languages need more words to cover a significant percentage of the words in the language than Indo-Aryan languages. Sankaran et al. [10] shows a comparison of the word coverage in Indic scripts with English using [11] corpus. This gives a picture of how error detection using methods like using a binary dictionary is effective in English and not much successful in Indic scripts. The paper [10] has used a larger corpus for experiments and the results are supportive of the experimental results observed in [37]. Another interesting observation is regarding how fast the number of new words observed declines in English [10]. However, in Indic languages, especially Telugu and Malayalam, this convergence has not happened with the available corpus. This explains why it is easier to create a dictionary in English with a good coverage using a standard corpus, but is difficult in Indian languages. We have repeated many of the experiments in these works using a larger corpus for most languages, consisting of crawled corpus from web. The details of the corpus is given in Table 3.7.

3.2.2 Exploding Number of Unique Words

In order to get an estimate of the word cover of languages, we have used the word coverage estimate. Word coverage is a measure of the number of unique words needed to cover certain percent of a lan-

Corpus	English	Gujarati	Hindi	Kannada	Malayalam	Marathi	Tamil	Telugu
Total Words	6,026,940	4,328,769	18,180,174	3,889,172	5,999,978	4,188,321	10,949,052	3,193,283
Unique Words	233,158 (3.87%)	288,406 (6.66%)	288,535 (1.59%)	607,051 (15.61%)	1,016,856 (16.94%)	289,949 (6.92%)	1,022,403 (9.34%)	432,554 (13.54%)

Table 3.7: Table shows the details of the data set used to create corpus. The data is obtained by crawling various online news sites.

guage. We have recreated the word cover statistics from a similar work [37]. The Table 3.8 shows the word coverage statistics of major Indian languages and English.

Corpus %	English	Gujarati	Hindi	Kannada	Malayalam	Marathi	Tamil	Telugu
10	3	9	4	28	45	23	58	66
20	10	48	10	203	283	103	248	272
30	35	158	29	783	944	300	691	793
40	107	426	88	2253	2594	723	1657	1905
50	362	1027	226	5613	6456	1572	3812	4303
60	1037	2389	535	13415	15422	3420	8963	9689
70	2643	5674	1221	33478	37870	7991	22624	22856
80	7103	14955	2957	94830	101738	20834	64921	60660

Table 3.8: The Table shows the word cover statistics of different Indic scripts and English.

To avoid any bias due to the differences in the size of the corpus available for each language, we have restricted the total number of words in each language to 3 Million. We can observe that while English requires around 7K words to cover 80% of the corpus, highly inflectional languages like Malayalam requires 100K words. It is interesting to observe that Dravidian languages require more than 60K words to cover 80% of the corpus while Indo-Aryan languages need much lesser words. It is interesting to observe that the unique number of words in a book of nearly 46K words was only around 5200 in Hindi. On the other hand, in Malayalam, a book of 33K words had 16K unique words in it.

We also estimate the average number of unique words per 1000 words in the language as the corpus size increases. A similar experiment was performed on three languages in [10]. We created the statistics in 8 different languages. The Figure 3.6 shows that while the number of unique words for languages like English, Hindi, Gujarati and Marathi becomes close to zero really fast, for languages like Telugu,Tamil, Kannada and Malayalam, the convergence requires a larger corpus. It is observed that Indo-Dravidian languages have large unique word occurrences compared to Indo-Aryan languages.

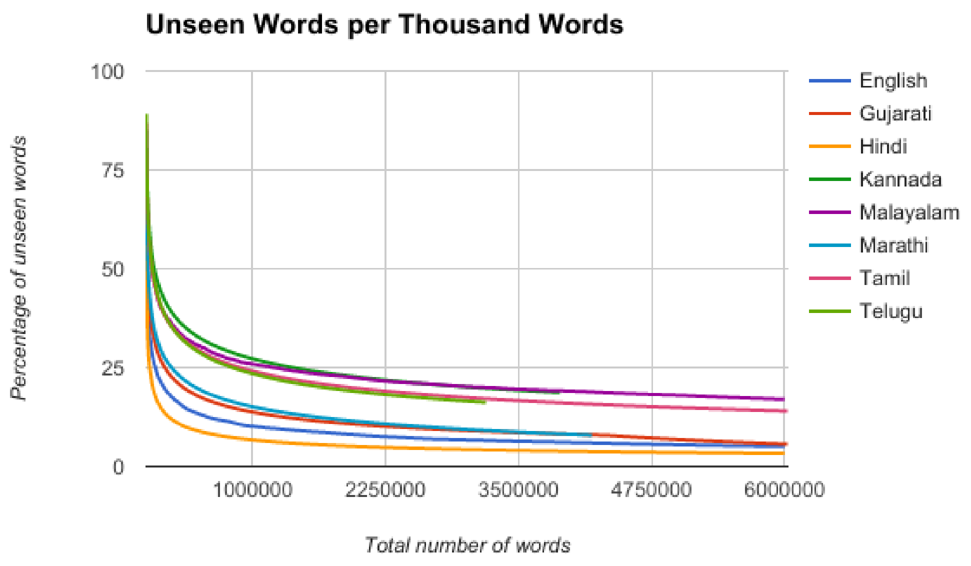


Figure 3.6: Unique word coverage between Malayalam, Telugu and English. For a language to get saturated, we would require a large dataset.(better seen in colour)

3.2.3 Overlapping of Glyphs

One of the major issues in Indic scripts is that of the complexity of scripts causing overlapping of glyphs during segmentation. Perfect glyph segmentation is a challenge since most Indic scripts have dependent vowels or other modified consonants which are connected to a consonant. This leads to issues in perfect glyph or character segmentation because a single character touching another character is segmented as a single character, thereby hindering the correct recognition process. Except for a few languages like Malayalam and Tamil, the complexity of scripts affects the OCR accuracy and post-processing activities. Generally in OCR systems, each non-touching independent symbol, referred to as glyph is the basic recognition unit. The segmentation of each word involves identification of these glyphs. Glyphs are segmented using connected components. This is depicted in Figure 3.7. In English,

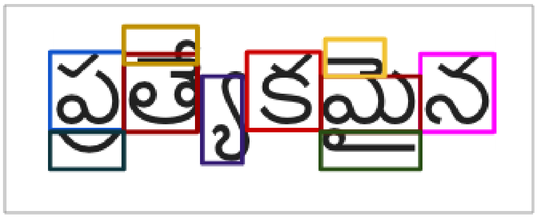


Figure 3.7: Figure shows the segmentation of each disjoint glyph in a word in Telugu. It is seen that segmentation of each glyph is a difficult task because of the nature of the script.

this level of segmentation is relatively easier because of the simple nature of the script. A major issue in Indic scripts is that the glyph level segmentation is overlapping due to the complexity of scripts (note the overlapping boxes). Most Indic scripts have dependent vowels or other modified consonants which are connected to a consonant. Hence perfect glyph segmentation is a challenge. This causes recognition issues leading to errors in the output such as a glyph getting mis-recognized as another glyph or cluster of graphemes. Post-processors are useful in this context wherein, a good Language Model and knowledge of error patterns can handle the issues created by similar shaped glyphs. Another issue is the combination of two or more unicode characters forming a compound character. One such instance in Malayalam is shown in Figure 3.8.

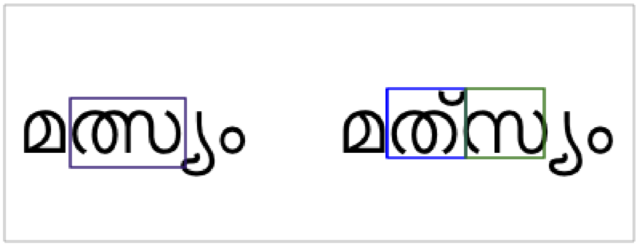
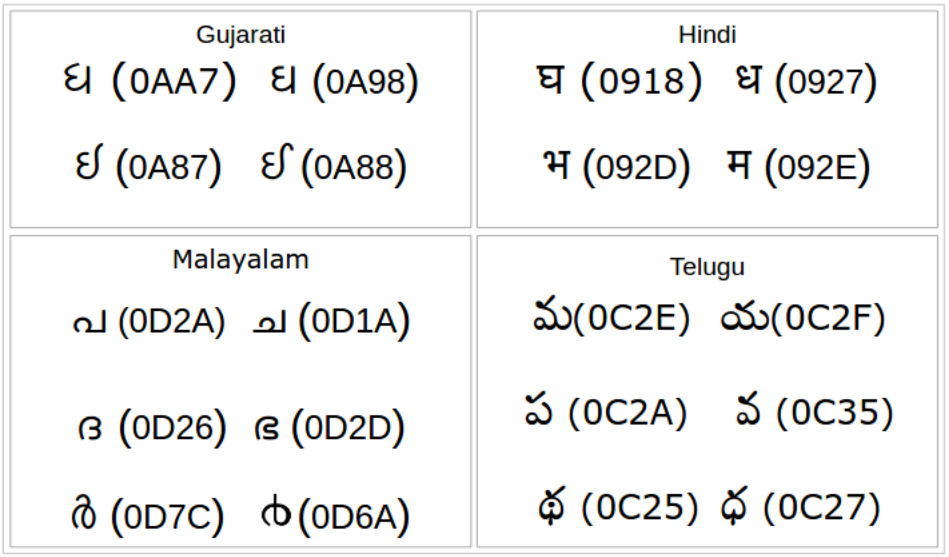


Figure 3.8: Figure shows different ways in which same word is written. Here compounding of characters causes three unicode characters to combine into a single connected component

Another major factor contributing to the difficulty is the visual similarity of various unicode characters in the languages. This is shown in figure 3.9. Some of these characters are difficult to distinguish



Gujarati		Hindi	
ધ (0AA7)	ઘ (0A98)	घ (0918)	ध (0927)
ઇ (0A87)	ઈ (0A88)	भ (092D)	म (092E)
Malayalam		**Telugu**	
പ (0D2A)	ച (0D1A)	మ(0C2E)	య(0C2F)
ദ (0D26)	ഭ (0D2D)	ప (0C2A)	వ (0C35)
ർ (0D7C)	൪(0D6A)	థ (0C25)	ధ (0C27)

Figure 3.9: Figure shows visually similar characters in Gujarati, Malayalam, Hindi and Telugu scripts along with their unicode values.

even for humans,without knowledge of the language.

3.2.4 Words at one Hamming Distance

The chances of mis-recognizing a word is more when there exist many words at a Hamming distance of one from the intended word. For example, the words {'bot', 'cat', 'sat', 'bag', 'pat', 'rat', 'bar', 'ban'} are all valid words at a Hamming distance of one from the word 'bat'. To identify the word in the image, the full context information may be necessary. For instance if the article is about machines and intelligence, 'bot' may make more sense. On the other hand, if it is a sports related article, then 'bat' makes more sense. There can be instances where the context information can also be of not much help in finding the word. This is because in Indian languages words can display variations due to inflection with only a single unicode glyph change. An example of this behavior is shown in the Figure 3.10 for some words in Malayalam and Hindi. In Hindi, most verbs display this inflection based on gender and singularity or plurality of the subject. In order to analyze the severity of the issue, [10] has used a

അതാണ് അതാണു അതാണീ അതാണോ അതാണൊ

चुनती चुनते चुनता

Figure 3.10: Figure shows the words that can be converted to another valid character with a Hamming distance of 1. The first row shows words in Malayalam and second that in Hindi.

plot of words which exists at particular Hamming distance. We have repeated the experiment in more languages in Figure 3.11. We can see that for languages like Gujarati, Hindi and Tamil, there exists many words at a single Hamming distance. This increases the incidence of real word errors. Malayalam has the least number of words at Hamming distance 1.

3.2.5 Lack of mature grammatical tools

To work at the word level, a suitable method would be to identify and decouple the agglutinated words. Also, separating the inflectional affixes can simplify any attempt made to detect errors. The performance of morphological analyzers and *sandhi* splitters developed for most Indian languages do not meet the level of accuracy required to aid in error detection [39]. This is because any incorrect identification and decoupling of agglutinated words can result in recognition of correct words as errors by the post-processor, which can negate its usefulness. Any post-processing system which should handle real word errors will require information about the parts of speech of a sentence. The parts of speech taggers can help identify the correctness of the inflectional word to a great extent. Also tools for semantic

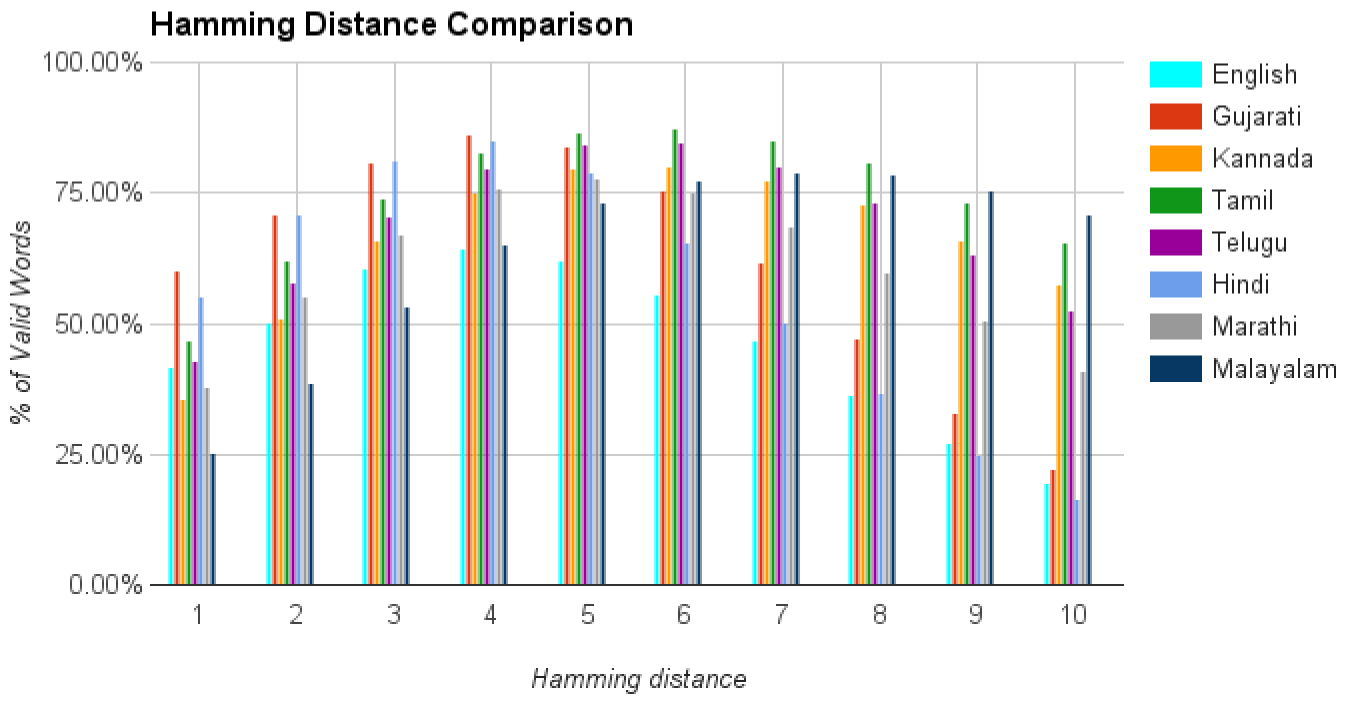


Figure 3.11: Plot showing the percentage of words converting to other valid words in various languages.

understanding, pragmatic modeling and discourse structure modeling if available would enhance the performance of the post-processing module, by incorporating the language information at a greater level than by mere language models created from words or sentences.

3.3 Summary

The errors in the output of Indic OCRs occur not only due to mis-recognition of glyphs/symbols but also due to the arrangement of the unicodes after recognition to form a word. Recognition systems which did not depend on sub-word level segmentation played a laudable job in removing these errors. The errors which occur in Indic OCR output are largely due to the almost touching glyphs and similar looking glyphs. To correct these error words is also a challenging task. The existence of huge vocabulary, especially in Dravidian languages like Telugu and Malayalam makes the error detection and correction using simple Dictionary method inefficient. The existence of many words at a Hamming distance of one from a word makes error correction process in languages like Hindi even more challenging. There are many words from which we have to choose the correct word for replacement of the error word. Also only if we have highly efficient grammatical tools can we come up with methods to deal with agglutinated words which is one of the primary causes of exploding number of unique words. At present we do not have efficient *sandhi splitters* which can help in splitting these words. These issues make post-processing in Indic scripts a challenging task which requires alternate methods to solve the problem.

Chapter 4

An Empirical Study of Effectiveness of Post-processing in Indic Scripts

4.1 Introduction

Errors in OCR system are largely unavoidable and occur due to issues like poor-quality images, complex font etc. A post-processing system can help in improving the accuracy by using the information about the patterns and constraints in the word and sentence formation to identify the errors and correct them. Indic OCRs have trailed behind in achieving accuracy comparable to English OCRs [4, 8], which have claimed accuracy close to 99%. The errors persisting in the recognized text, after the shape classifier has done the recognition task, are handled by a post-processing module. This module uses the language information to improve the accuracy of text recognized further. First we explore the effectiveness of a statistical language model or SLM based error correction technique, for post-processing. We have used character (unicode) level bigram and trigram language models to find the errors in the OCR output. Since *aksharas* form a more meaningful sub-unit of a word, we evaluate the performance of SLM based method using *akshara* level ngram Model. It is often assumed that a simple dictionary method, if employed can increase the accuracy significantly. We perform experiments to understand the efficacy of dictionary based method for error correction. We analyze if the dictionary based method is able to correct the errors in Indic OCR output as well as it perform in English.

Both SLM and dictionary based methods can only detect error words which are not valid words in the language, known as non-word errors. For example, consider the sentence “Take a break”. If the OCR recognizes this sentence as “Tale a break”, both the above mentioned approaches are bound to fail as Tale is a valid word in English. These class of errors known as real word errors require context information and can be corrected by using a word level statistical language model [17]. Non-word errors form a major portion of errors in the OCR output and detecting and correcting them should itself improve the OCR error rate significantly. Hence we restrict our work to only non-word errors at present.

The SLM based method relies on the use of a language model, which computes the probability of a sequence of characters or words. We have conducted the experiments using two different language models, namely unicode level and *akshara* level. We have not used word level language model which is a popular approach in English [17]. The character based language models better model languages with a rich morphology. To use a word level language model, a huge corpus containing text from different domains is essential. Since we do not have such a corpus, we restrict our work to character and *akshara* level language model.

The success of dictionary based method for error detection depends on the number of words included in the dictionary. If a word correctly recognized by the OCR is not present in the dictionary, then the dictionary based method would label it as an error (when the word is actually correct). These false positives will affect the recognition accuracy of the OCR as these words would be replaced by any word which is closer to the detected 'error word' from the dictionary. Therefore, it is essential that the dictionary we use should cover a large percentage of words in the language. This itself is another issue as the number of words required to cover a language like Telugu and Malayalam is much larger than those required in English [10]. Another issue is availability of balanced corpus in Indic languages. Since a large corpus like BNC corpus [11] in English, which is collected from a wide range of sources is not available in Indian languages for direct use, we depend on crawled corpus from the web. This corpus mainly involves the news sites and other such sources. This restricts our dictionary words, making it less diverse. The availability of corpus which covers a wide range of topics can solve this issue. Another bottleneck occurs due to the properties like inflection and agglutination. Inflection causes a word to exist in different forms to express a grammatical function such as tense, gender, mood, number, person etc. Usually these words show variation at the end of a word. A bigger challenge is when many words join together to form new words (agglutination). This creates too many words to be included in the dictionary. Since we do not have highly efficient NLP tools to split the agglutinated word into its constituent words, this issue is left unsolved at present. The advantages of using a dictionary lies mainly in the fact that it does not require any complex post-processing model, but an unabridged dictionary containing the words in the respective language. Another factor to consider is updating the dictionary continuously as and when new words are added in the language. Though this method fails to detect correctness of some words like proper nouns (whose addition to the dictionary is a difficult task), it is a straight forward and effective method, especially in English.

In this Chapter, our aim is to understand if traditional post-processing approaches which work well in English would perform the same in Indic OCR outputs. We perform experiments to assert the importance of considering *akshara* as the basic unit of a word in Indic scripts, instead of unicode which can represent alphabets in English. We also understand the different errors occurring at various Hamming distances from the original word and the effect of these error detection and correction techniques on these errors.

4.2 Methodology

We have conducted experiments using SLM and dictionary, to analyze the performance of post-processing in Indic scripts with English OCR output. In our experiments, we have not included shape classifier accuracy information such as the probability of recognized characters and next probable character information. We also do not take into account the character confusion information, which depends on the OCR system used for recognition. Assuming that the OCR output text is available, we use ngram probabilities to detect the errors and find possible replacement words for correcting the errors detected.

4.2.1 SLM based Post-Processing

4.2.1.1 Statistical Language Models: An Overview

The works related to statistical language modeling task has been in progress since the beginning of the 20th century when Markov tried to model letter sequences in works of Russian literature [40]. Linguist G K Zipf [41, 42] studied statistical properties of text and formulated an empirical law using mathematical statistics. Zipf's law states that given a large sample of words used, the frequency of any word is inversely proportional to its rank in the frequency table. However, it was Shannons work [19] that inspired later research in this area. Shannon used a prediction machine that involved n-grams to investigate the information content of English text. He evaluated n-gram models performance by comparing their cross-entropy on text with the true entropy estimated using predictions made by human subjects. For many years, statistical language models have been used primarily for automatic speech recognition. Since the first significant language model was proposed [43], statistical language modeling has become a integral component of speech recognition, machine translation, spelling correction, and so forth. There are other applications of natural language processing tasks such as natural language generation and summarization. A statistical language model is a probability distribution over all possible sentences or other linguistic units in a language [43]. It can also be viewed as a statistical model for generating text. The task of language modeling, in general, is to answer the question: how likely can we observe the ith word in a sequence assuming that we know the preceding i-1 words? In most applications of language modeling, such as speech recognition and information retrieval, the probability of a sentence is decomposed into a product of ngram probabilities of constituent tokens. Consider a sequence S of k words,

$$S = w_1, w_2, ..., w_k$$

An ngram language model considers the word sequence S to be a Markov process with probability

$$P_n(S) = \prod_{i=1}^{k} P(w_i | w_{i-1}, w_{i-2}, ..., w_{i-n+1})$$

where n implies that it is n^{th} order of the Markov process. A bigram model is estimated by using the information about the co-occurrence of pairs of words i.e, $n = 2$. When $n = 1$, we get a unigram model which estimates the probabilities of individual words. In information retrieval, the role of word order is less clear and unigram models have been used extensively. Unigrams are powerful, especially in cases where occurrence of words like “Dissappointed”, “Interesting” etc. gives more insight into the sentiment of the text. For applications such as speech recognition, natural language generation, machine translation etc., word order is important and higher-order (usually trigram) models are used. To establish the word ngram language model, probability estimates are typically derived from frequencies of ngram patterns in the training data. It is highly likely that many possible word combinations would not appear in the actual data used for estimation, even if the size of the data is huge and the value of n is small. As a consequence, for rare or unseen events the likelihood estimates that are directly based on counts can cause issues. This is often referred to as the data sparseness problem. A popular method used for issues like data sparseness is smoothing. Many smoothing techniques are available today like Additive smoothing, Good Turing Estimate, Kneser-Ney smoothing etc [40]. Evaluation of language models has typically been done using a measure called “perplexity” [40] which is directly related to entropy. Entropy measures the average uncertainty of a single random variable. From a language perspective, it is the information that is produced on an average, for each letter of text in the language. When a model which has enough information about the text it has modeled, the uncertainty or entropy will be less. Hence, lower the entropy, the better the model.

4.2.1.2 Language Model Creation

As we have seen, the goal of using language modeling here is to learn a probability distribution over a sequence of tokens in a word. Tokens used here are characters and *aksharas* in a word [40]. In SLM based error correction, we have used the probability of bigrams and trigrams in words for error detection and correction. Bigrams provide the conditional probability of a token given the preceding token. Bigram probability is equal to the probability of their bigram, or the co-occurrence of the two tokens $P(W_{n-1}, W_n)$ divided by the probability of the preceding token. This is shown in the equation below.

$$P(W_n|W_{n-1}) = \frac{P(W_{n-1}, W_n)}{P(W_{n-1})}$$

where W_n is the n^{th} token and W_{n-1} is the token preceding it, i.e $(n-1)^{th}$ token. Similarly, a trigram uses the probability of a character, given the previous two characters in a word. We have created the language model using SRILM [44]. We have combined the corpus created from the 5K books [5] and crawled corpus [45] to create the language model. Smoothing is done to take into account those words which have not appeared in the corpora, in which case a probability of zero will be assigned to them. We have used the Good Turing Estimate for applying smoothing in the language model.

4.2.1.3 Error Detection using SLM

In this approach, we use a unicode level language model which gives the bigram and trigram probabilities of unicodes in a word. This is used as a look-up table. We then find the average of bigram and trigram probabilities of unicodes in the input word using the look-up table. If this value is less than a threshold, we declare the word as error and correct word otherwise. The basis of this method is the assumption that all the non-word errors have a probability (calculated from its constituent ngrams) much less than that of the correct words. For each word which is fed into the error detection module, we split the word into its constituent unicode characters and obtain the bigram and trigram probabilities of its characters. If a bigram or trigram is not found in the bigram or trigram list, it is given a very low probability value, indicative of the presence of an error. The computation of the word probability is as follows. Consider a word ”bags”. We begin by splitting it into ”b-a-g-s”. We then append a start-of-word ($< s >$) and end-of-word ($< /s >$) marker at the beginning and end of word. Now we identify bigrams in the word, which in our case are $\{< s >$b, ba, ag, gs, s$< /s > \}$. We find the product of these bigrams which gives bigram probability of the word. The trigrams in this case are $\{< s >$ba, bag, ags, gs$< /s > \}$. We also find the trigram probability of the word. We compute the average of these two probabilities, which is then used to decide if the word is error or not. This is done by comparing the probability value with a threshold previously estimated from a list of correct words in the language.

4.2.1.4 Error Correction

The assumption behind this approach is that the error in a word exists at the lowest probability ngram. Now, to replace this ngram we find a list of ngrams which are at least distance from the ngram to be replaced. From the candidates for replacement, we choose that ngram which maximizes the probability of the word. Repeat the above steps on the result for a fixed number of times (for correcting multiple errors) or till average of bigram and trigram probability obtained is above a threshold. To replace an ngram, we search both these bigram and trigram list because a deletion error or an insertion error can be taken care of by looking in both the lists. There are two possible issues we face in this method. Even with many ngram replacements we may not get a word probability which is satisfactory. In this case we stop the replacement after a fixed number of iterations. The second issue is that different ngram replacements can give us different words which are all valid words in the language. In this case we choose the word whose probability is the highest.

4.2.2 Dictionary based Post-Processing

4.2.2.1 Dictionary Creation

For error detection using Dictionary, we check if the word is present in the dictionary or not. The word will be labeled as a correct word only if the word is present in the dictionary. Figure 4.1 shows the error detection stage using a dictionary. The Figure 4.1 also shows correction stage wherein candidates for replacement are generated. Even if the word is a valid word, if it is not present in the dictionary, it will be labeled as an error word. This is critical in case of dictionary method because a dictionary with insufficient word coverage will create a lot of false positives (correct words recognized as errors). These words when passed to the next stage will result in these words being replaced by other words closer to

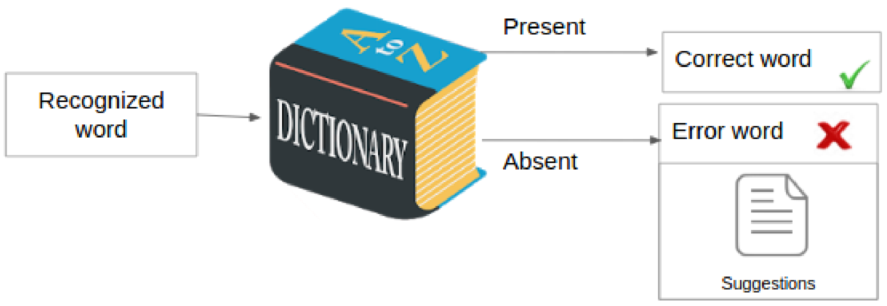


Figure 4.1: Error Detection using a simple dictionary method. The words output by the shape classifier is input to a dictionary. The word is labelled as error if it is not present in the dictionary. For each error word top n candidate words are retrieved for replacement.

these words. This causes the word error rate of the OCR pipeline to increase instead of decreasing after using a post-processor. The solution to avoid such issues would be to make the dictionary as huge as possible. Also we need to continuously add new words emerging in the language to the dictionary.

Generally, the success of dictionary method depends on the size of the dictionary. In our case, to create a large dictionary in English is a fairly easy task as there are huge corpus like Google dataset [25] etc. In case of Indian languages we have the limitation of corpus availability. The only large corpus we can depend on is the crawled corpus which is abundantly available [45]. However, the variation of data you observe in books cannot be found in the crawled data. Hence, we have used the words present in 5K book corpus [5] to create a dictionary in all the five languages. The words in the test book may have overlap with words in the dictionary, but is not guaranteed. Let us call this dictionary 1. The details of the number of words in the dictionary 1 for each language is given in the Table 4.1. We have also created another dictionary (dictionary 2) which contains all the words in dictionary 1 along with the correct words in the book used for testing.

4.2.2.2 Error Detection

For error detection, we check if the word is present in the dictionary or not. The word will be labeled as a correct word only if the word is present in the dictionary. Even if the word is a valid word, if it is not present in the dictionary, it will be labeled as an error word. This is critical in case of dictionary method because a dictionary with insufficient word coverage will create a lot of false positives (correct words recognized as errors). These words when passed to the next stage will result in these words being replaced. This causes the word error rate of the OCR pipeline to increase rather than decrease after using a post-processor. The issue in Indian languages is that the dictionary cannot cover all the words. We cannot even guarantee that the dictionary will cover most of the common words in the language. The

Language	English	Hindi	Gurumukhi	Telugu	Malayalam
Words in dictionary1	38,727	92,620	90,844	258,299	331,007
Words in dictionary2	40,410	93,530	91,297	264,831	336,013

Table 4.1: Details of the vocabulary size used to build the dictionary1 and dictionary2.

reason being that the words in languages like Malayalam, Telugu etc. are agglutinative. Though their *sandhi* split words may exist, their agglutinative combinations are difficult to cover. And we cannot add all the agglutinated words to the dictionary as the list is nearly endless because of the enormous combinations of words generated.

4.2.2.3 Error Correction

When a word is detected as error in the previous stage, the closest word from the dictionary is used to replace it. As a first step, a list of top 'n' candidate words are retrieved from the dictionary. There are many ways to find the closest matching word in the dictionary. A popular method is using edit distance (Levenshtein distance) based distance metric[46]. In this method, the distance is the number of deletions, insertions, or substitutions required to transform the source word into the target word. Another popular metric used to find the closest matching words is Gestalt algorithm [47] which is used in spell checkers.

We have conducted the experiments in four different Indian languages namely Hindi and Gurumukhi (Indo-Aryan languages) and Telugu and Malayalam (Dravidian languages) [48]. We also compare the results of performance in these languages with English. To analyze the errors corrected, we divided the errors based on their distance from the actual word into five classes namely, errors at distance 1 to 4 and above 4. The classification of errors based on Hamming distance from the actual word is shown in Figure 4.2. The errors at lower Hamming distances from the correct word should be easier to correct than the ones which are at larger distances. The errors produced depend on the shape classifier used and the quality of images used for recognition. We can find that in the Figure above, English has a significant portion of errors which are above 4 distance from the actual word. This will affect the error correction process in English. These types of errors are due to faulty images or font issues which makes comparison of different OCR errors difficult. Hence, we have considered only those errors which are at a distance less than or equal to Hamming distance 3 from the actual word for error correction.

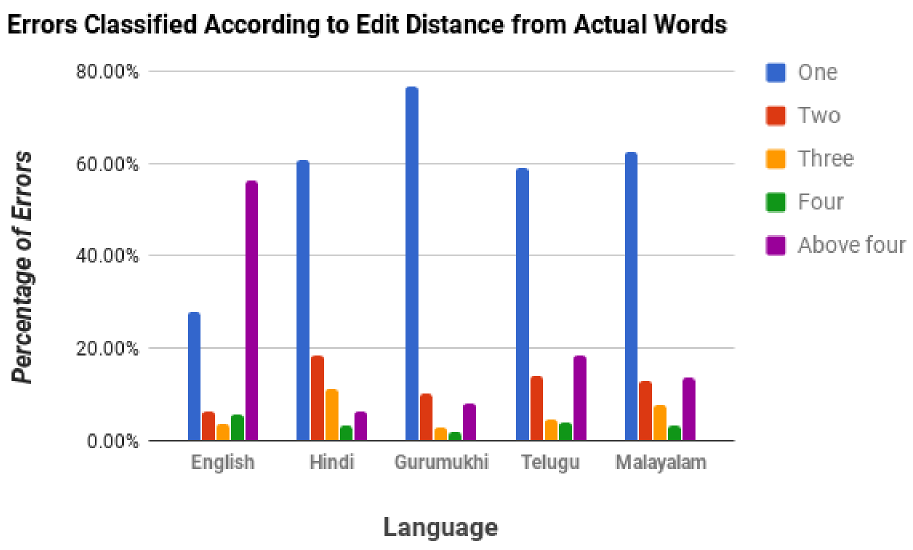


Figure 4.2: Figure shows errors classified according to its Hamming distance (1 to 4 and above 4) from the actual word in different language OCR outputs.

4.3 Results and Analysis

4.3.1 Results using SLM

The results of error detection using unicode level SLM is shown in Figure 4.3. The Figure shows errors at varying distance from the actual word in different colors. It is observed that this method does a significant role in detecting errors, especially in languages like English and Gurumukhi. We repeated the experiment, this time using *aksharas* instead of unicode level SLM. The result of error detection using *aksharas* is shown in Figure 4.4. The error detection accuracy for errors at various distances are shown in different colors. A comparison of error detection performance using *akshara* and unicode is shown in Figure 4.5. It is clear that *akshara* level SLM do a significantly better job in detecting the errors. This is because insertion or deletion of even a small glyph in the word can alter the aksharas formed. When a valid word is split, the *aksharas* generated also will be valid. On the other hand, splitting an error word causes formation of invalid *aksharas* which are less likely to be listed in the unigram list of *aksharas*. Formation of such *aksharas* are indicative of presence of error in the word. This information is not available in unicodes; hence unicode performance is not as good as that of *akshara* split words. It is also observed that in Malayalam, more than 75% of the errors at Hamming distance 1 could be detected using *aksharas* while around 30% only could be detected using unicodes. Telugu also shows a significant improvement in error detection results when we switched to *aksharas*. The error correction using SLM at unicode level is shown in Figure 4.6. The error correction is not significant in any of the languages when unicode level language model is used. The result of error correction using *aksharas*

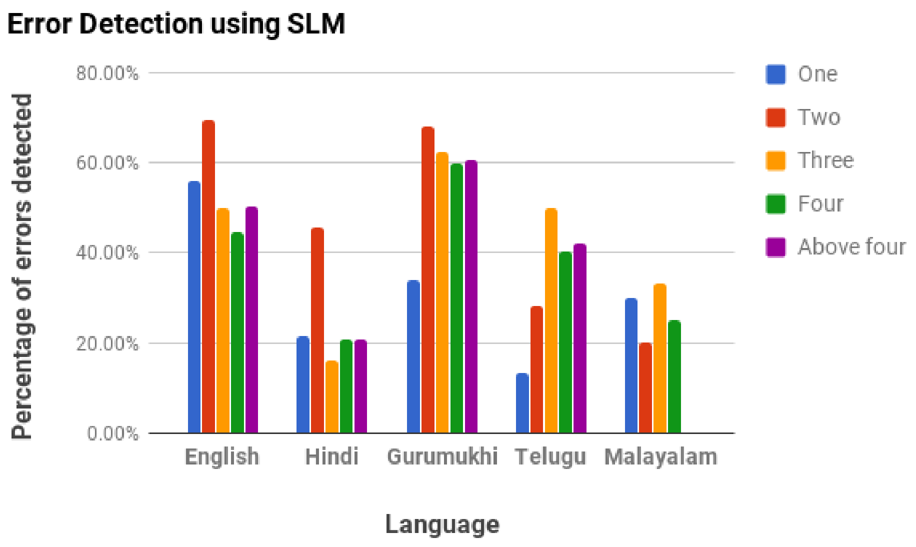


Figure 4.3: Error Detection using SLM at unicode level, for different errors at varying distance from the actual word (shown in different colors).

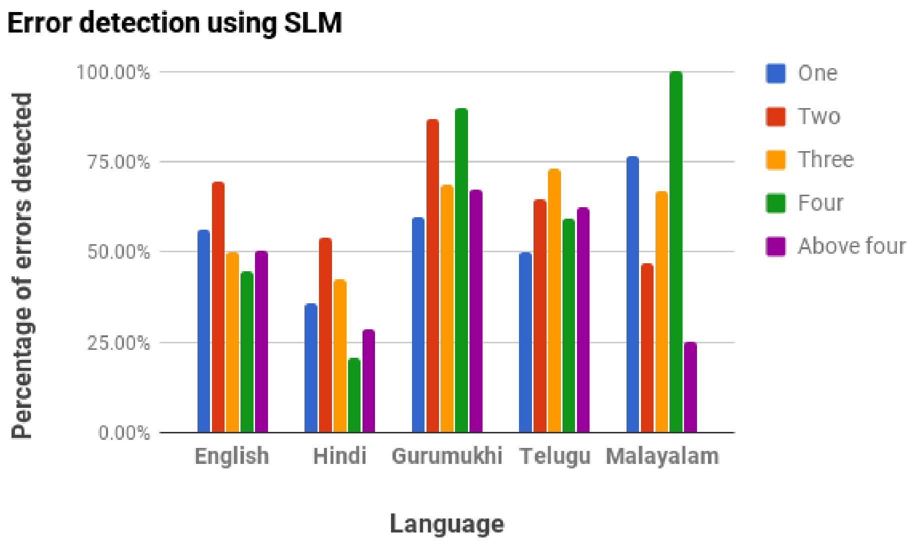


Figure 4.4: Error Detection using SLM (*akshara* level for Indian languages and unicode for English) for different errors (shown in different colors) at varying distance from the actual word.

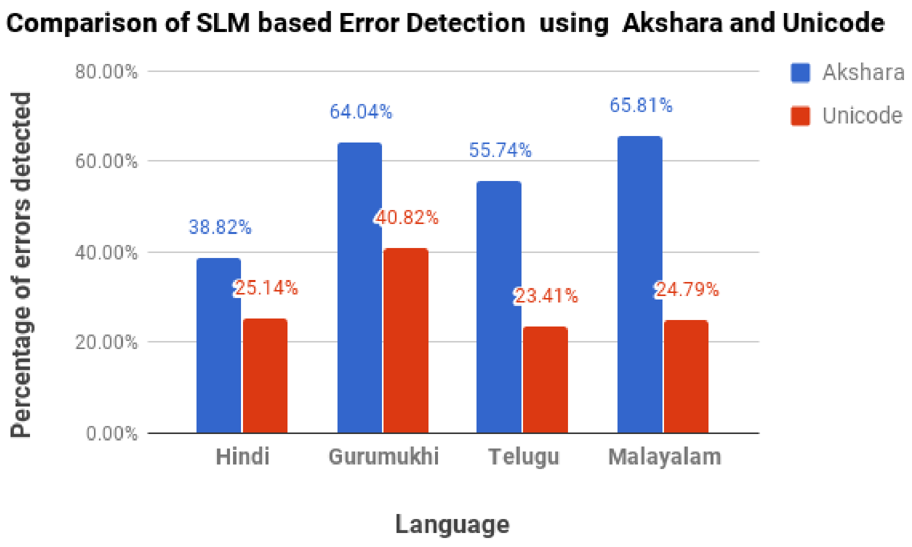


Figure 4.5: Figure shows comparison of Error Detection using *akshara* (blue color) and unicode level (red color) SLM for Indian languages.

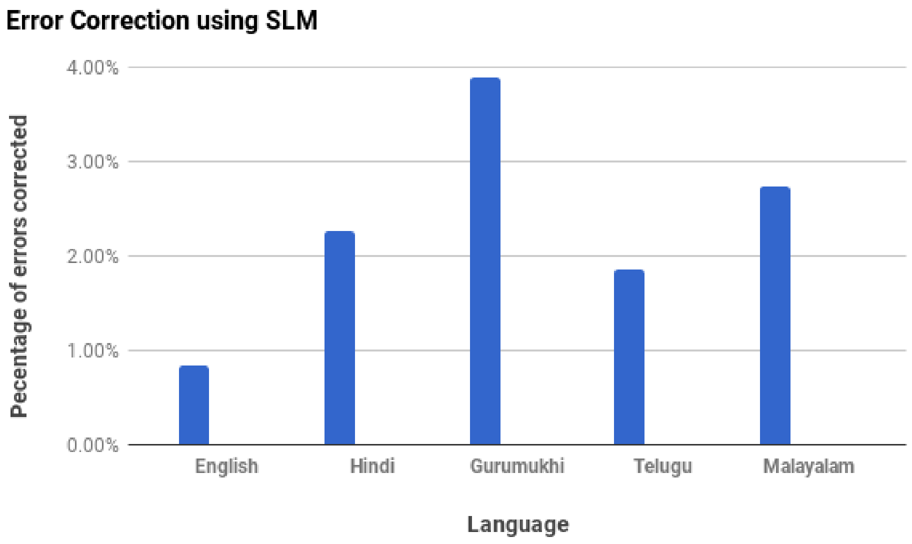


Figure 4.6: Error Detection using SLM at unicode level, for different errors (shown in different colors) at varying distance from the actual word. The errors beyond distance 1 are not corrected using SLM.

is shown in Figure 4.7. The results using *aksharas* are better than those using unicode, both for error

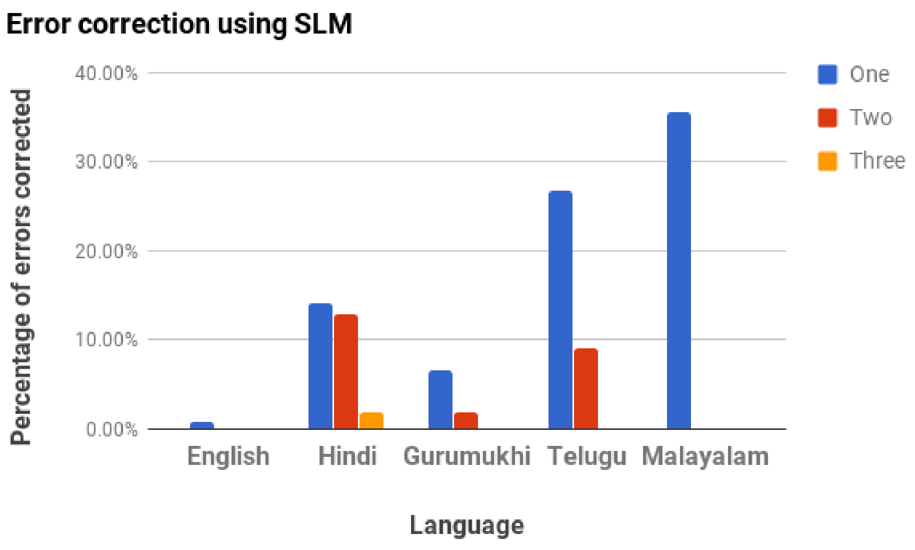


Figure 4.7: Error Correction using SLM (*akshara* level for Indian languages and unicode for English) for different errors (shown in different colors) at varying distance from the actual word. Very few errors beyond distance 1 are not corrected using SLM.

detection and correction. However, error correction using unicode and *akshara* do not yield promising results in any language. The use of ngrams for error correction can create multiple candidate words. Since we have to choose only one word for replacement, we have chosen the word with the highest probability. This can create a situation wherein a correct replacement which does not have the highest probability among the candidate words being ignored by the system.

4.3.2 Results and Analysis of Dictionary method

In the error detection experiment performed using dictionary, in Hindi 57% of errors were detected and in Gurumukhi 66% of the errors were detected. The highest error detection is observed in Malayalam and Telugu, 78% and 70% respectively. In English, only 44% of the errors could be detected. When we observed the errors in English, many errors which occurred were real word errors, due to incorrect recognition of punctuation etc. In Hindi, when *matras* were recognized incorrectly, inflection caused many incorrectly recognized words to be valid words. The results of experiments of error correction using dictionary method is shown in Figure 4.8, in which we have retrieved the top 3 candidates for error correction from the dictionary. In order to observe the performance of this method when all correct alternatives are available in the dictionary, we have done the experiment using dictionary 2. When using dictionary 1, we can see that in English, 56% of errors could be corrected. However, after using dictionary 2, the percentage of error words corrected is 61%. When we compare this with other

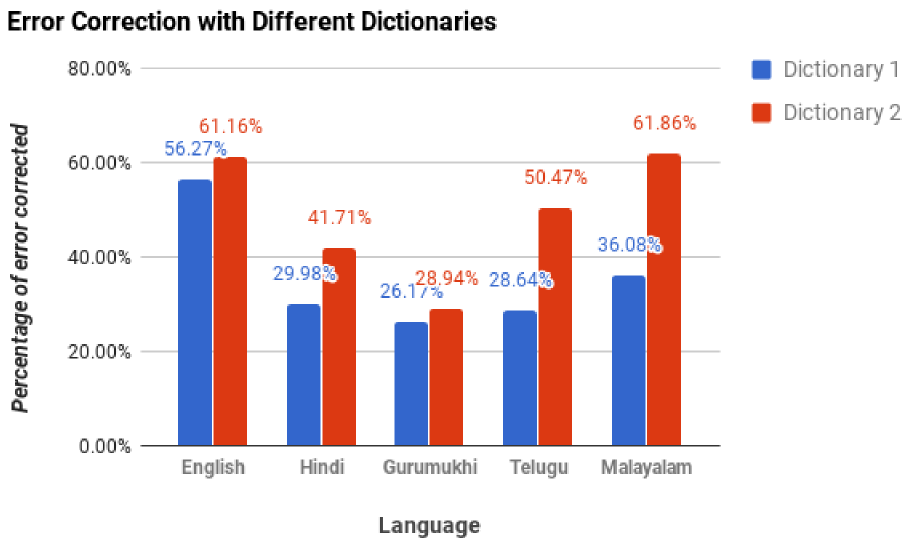


Figure 4.8: Figure shows the results using 2 different Dictionaries, Dictionary which has all correct words corresponding to the error words included (red) and one in which it is not explicitly included (blue). Gestalt score is used to find the candidate words.

languages, we can see that in Malayalam, the correction accuracy increased from 36% to 62%. This is a significant increase. A similar behavior is observed in Telugu, from 29% to 50%. Though Hindi and Gurumukhi also have their error correction rate improved, it is not comparable to the increase we see in Malayalam and Telugu. This shows that the dictionary 1 covers many common words in the languages in English, Hindi and Gurumukhi. Whereas in Telugu and Malayalam, many words were added which were not present in original dictionary. The error detection in inflectional languages is easy if we are able to create a good dictionary. An alternate method we can use is to split the words which are agglutinated so that the words before agglutination, if present in the dictionary can validate the word. This requires improved language processing tools in the language. The results using Levenshtein distance as the distance metric are shown in Figure 4.9 which is comparable to the results obtained using Gestalt score.

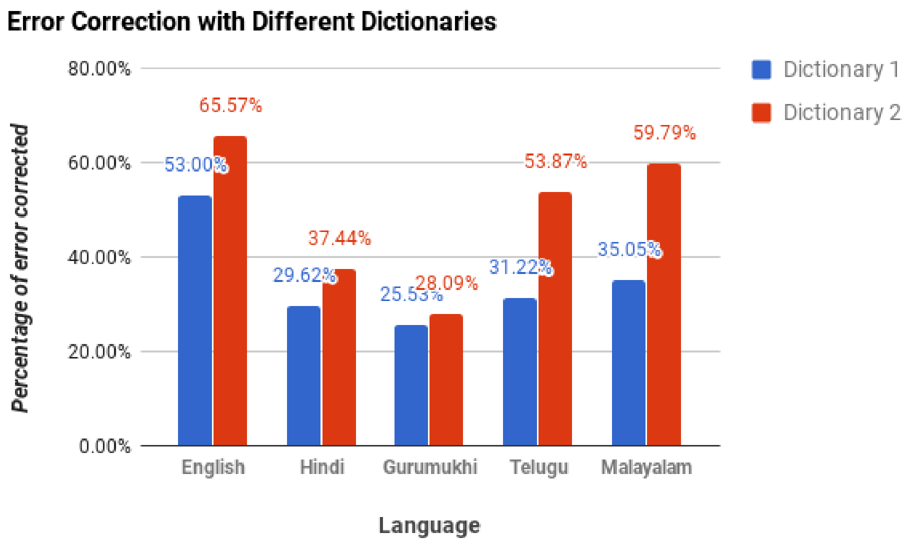


Figure 4.9: Figure shows the results using 2 different Dictionaries, Dictionary which has all correct words corresponding to the error words included (red) and one in which it is not explicitly included (blue). Levenshtein distance is used to find the candidate words.

4.4 Summary

In Indic language OCRs, traditional methods used for error detection and correction such as dictionary and character ngrams alone cannot solve the problem. A major bottleneck is the availability of a balanced corpus to create an unabridged dictionary and word level language model. The dictionary creation is particularly a difficult task for Dravidian languages such as Telugu and Malayalam due to the exploding number of unique words. We also need grammatical tools like morphological analyzers, POS taggers etc. to tackle the problem effectively. Also when compared to unicode, *aksharas* are more meaningful choice as the basic unit of a word in Indian languages. *Akshara* level language models contain more information when compared to unicode level language models.

Chapter 5

Error Detection In Indic OCR using RNN and GMM

In this thesis, we propose an error detection technique for Indian languages using a combination of Recurrent Neural Network (RNN) and a Gaussian Mixture Model (GMM). In Indian languages, words are composed of *aksharas* which are similar to syllables in English. We divide words into their constituent *aksharas* and use their bigram and trigram probabilities to build features for training the classifiers. RNN learns the pattern of word formation using *aksharas* in correct and incorrect words from their bigram and trigram probabilities. We use the GMM model to check the misclassification of right words as errors by the RNN. The error detection approach essentially requires the learning of patterns which can distinguish a word as error or not. This was the motivation behind using RNN which offers good trainability. Our method can be used on any language without requiring knowledge of the intricacies of its grammar, provided we have a fairly large and clean corpus. Unavailability of a large corpus prompted us to use a web crawler to take advantage of the huge digital content available online.

5.1 Methodology

5.1.1 Basic OCR Model and Error Detection Procedure

One of the primary issues in Indian languages is the unavailability of a huge corpus like the British National Corpus [11], Brown Corpus etc. which are available for English. A huge corpus is essential to create a good language model for any language. The unique word coverage of the existing corpora like [38] are not sufficient for our application. To include words across domains, a common approach [37] is to use a corpus made by crawling popular news sites and other websites in popular Indian languages. The crawled data contains noise due to unwise use for Zero-Width Joiners (ZWJ) and Zero-Width Non-Joiners (ZWNJ) which are used for proper rendering of the unicode symbols. Also many unicode characters which do not belong to the concerned language may also be present in the crawled data. The unicode range for the language is used to filter out the undesirable characters from the corpus. Further, simple cleaning techniques like eliminating words with occurrence of successive vowels are also done. Splitting of words into *aksharas* can be done using a simple regular expression. *Akshara* is formed using zero or more consonants followed by a vowel. When a word is split into its constituent syllables, if the syllables formed does not belong to the set of syllables already created from the corpus of large words, it is likely that an error has occurred. The words in a particular language has a set of commonly used syllables. This set is not finite, yet if a large corpus is used, we get a fair share of the commonly used syllables. The presence of errors in a word often results in the formation of syllables which are generally not found in the language. However, with the increasing influence and incorporation of words over time from other languages, especially English, the number of syllables in the language is also increasing. For example, many English words like stall, bag, office etc. are widely transliterated to Indian languages, introducing new syllables. Also, in error words, even if the constituent syllables are valid, its bigram or trigram combinations may have less probability. In our experiments, we first split the words into their constituent syllables to compute their bigram and trigram probabilities in the corpus. Table 5.1 shows the number of unique words in the crawled corpus, which is used to create syllables in each language, along with the number of unique syllables, bigram and trigram counts.

SRILM toolkit [44] is used to compute the bigram and trigram probabilities of syllables and smoothing is done using Good-Turing discounting to estimate the probabilities of unseen objects [49]. Probability computations are done for ngrams using nth-order Markov chain assumption and log probabilities are used in computations, since the probability values are very small. The probability of unseen syllable ngrams are taken care of using the smoothing technique as shown below.

$$p_0 = \frac{N_1}{N} \qquad p_r = \frac{(r+1)S(N_{r+1})}{NS(N_r)}$$

where p_0 is the probability for an unseen syllable ngram, p_r is the probability for an ngram encountered r times, N is the total number of ngrams, N_i is the count of ngrams occuring i times and S is a smoothing function. The Simple Good-Turing (SGT) method uses a simple linear smoothing function and also specifies a threshold for switching from Good-Turing estimate to Maximum Likelihood Estimate (MLE) for higher frequencies as Good-Turing estimate is accurate only for lower frequencies. We create a lookup table (LT) of these syllables along with their bigram and trigram probabilities for creating features of the words.

Language	Unique Words	Unique Syllables	Bigram Count	Trigram Count
Hindi	891,960	15,805	313,989	407,534
Malayalam	398,887	7,257	124,033	176,087
Gujarati	643,986	7,889	172,581	271,075
Telugu	1,305,852	10,762	254,960	441,806

Table 5.1: Statistics of Unique Words and Syllables in Different Indian Languages.

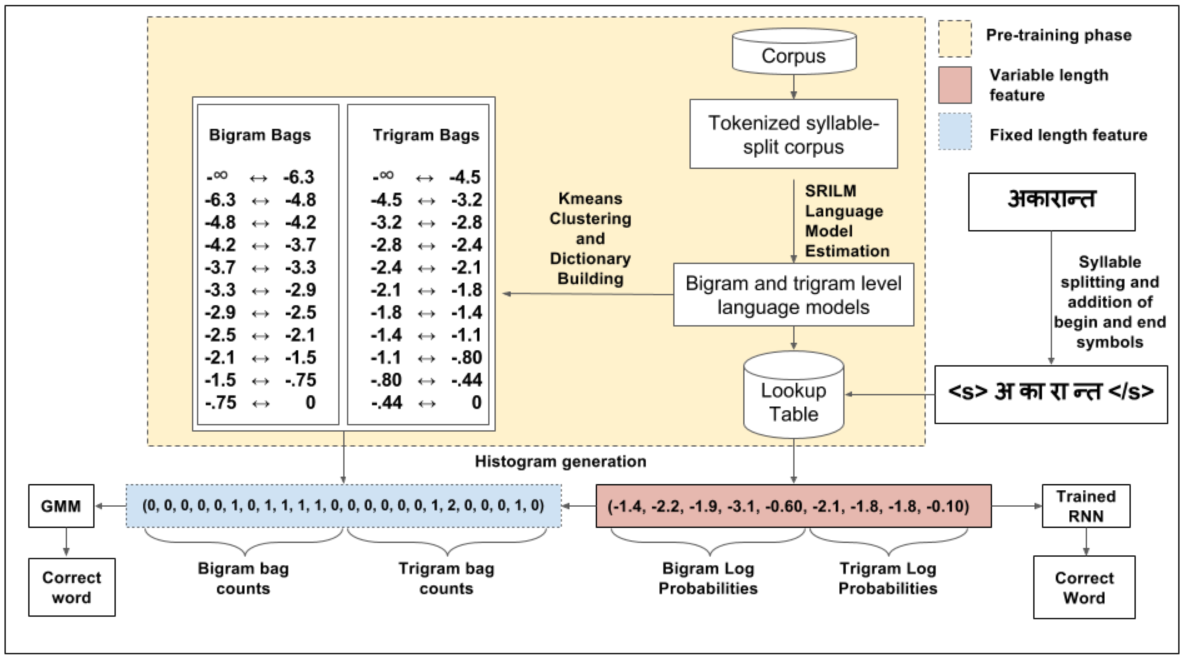


Figure 5.1: The image shows how feature is created from a word for RNN and GMM training. After adding markers to the beginning and end of *akshara* split words in a huge corpus, its language model is generated. The bigram and trigram models are clustered separately. We then perform Dictionary Building to find the cluster centroids and create bags of syllables, which is stored as a lookup table. The GMM model takes as input, the fixed length histogram of the syllable split words whereas the RNN uses the raw bigram and trigram syllable probability. Each model then makes a prediction of the label of the input word. A word is declared error only if both the models label it as an error.

5.1.2 Structure of the Solution

We use two methods for detecting errors in the OCR output; one using generative model and the other using a neural network. In generative approach, we use a Gaussian Mixture Model to create models for correct words and error words in the OCR output. In the second approach, we use BLSTM [50] deep learning neural network for classification. In order to create features for training, we have used the bigram and trigram probability of syllables in the corpus, obtained from the lookup table LT. We split each word in the huge corpus into its constituent syllables and add special characters to mark the beginning and end of the word. This is important because in Indian languages, only a specific set of syllables can occur at the beginning of any word. Certain unicode character combinations which occur in the erroneous words, may not be present in the list of syllables created from the huge corpus. We assign a very low probability value to bigrams and trigrams containing these character groups.

5.1.3 Gaussian Mixture Model for Error Detection

In this method we first cluster the probabilities of all syllable bigrams in the corpus, using K-means clustering to create bags of syllable bigrams. We have found 10 to be optimum number of clusters giving good results by testing on validation data. Each bag has a minimum probability and a maximum probability bigram. We then use this bag of syllable bigrams to create a histogram of each syllable split word. The same procedure is done for probabilities of syllable trigrams. If there are J bags for syllable bigrams and N bags for syllable trigrams, the size of the feature is $J + N + 2$. Here the 2 is for the additional bags for unseen syllable bigrams and trigrams. The steps to creating feature vector for a word are as follows:

1. Create a zero vector of dimension equal to J.
2. For each syllable bigram in the word identify the bag j in which the bigram probability lies.
3. Increment by one, the count of the the j^{th} component in the feature vector.
4. In case of new syllables, we increment the count of the bag reserved for unseen bigrams.
5. Repeat the procedure for trigrams using zero vector of size N.
6. Concatenate the above feature vectors to get the final feature vector.

These two histograms of bigram and trigram probabilities are used to create a Gaussian Mixture Model. In the model we used validation dataset to find the optimum number of components such that issues of over fitting of data (with too many components) are taken care of. The procedure is done for obtaining the models for correct words as well as erroneous words. For each word in the testing data, we find the model which best fits the histogram of the word. The word is declared error if it fits the error model and right otherwise. In GMM model, we use the information in the language model to predict the label of the words. When the GMM is given an unseen word whose syllable bigram and trigram probabilities are comparable to the trained valid word probability, it can use the language model information to correctly predict the label of the word. We preferred to use GMM over other generative methods because of the flexibility it offered in selecting the number of mixture components and its ability to cluster multi dimensional data of unknown distribution better.

5.1.4 Error Detection using RNN

Recurrent neural network (RNN) is a class of neural networks with the capability of persisting the information from previous states. The loops or connections in the nodes of the recurrent neural network enable it to use an "internal memory" to remember and process past information[51]. In our problem of error detection in OCR output, we use a Long Short Term Memory (LSTM) network. The LSTMs have been used in a wide range of problems including text recognition in images and generating language models. Bidirectional RNNs are based on the idea that the output at particular time may not only be dependent on the previous elements in the sequence, but also on the future elements. We prefer the use of LSTM for error word detection over other classifiers like support vector machines. A neural network can learn the error model in the erroneous words during training. Apart from the advantage provided by the use of networks for better learning, it also provides flexibility of using arbitrary number of sequences as input. The number of unicodes or *aksharas* in words are not fixed, leading to different number of bigrams and trigrams in different length words. We need not use padding or other methods to create fixed length feature while using a LSTM. While GMM uses bags of *akshara* level ngram probabilities, RNN uses the raw values of probabilities for training. For each bigram and trigram in the word, the bigram probabilities, followed by trigram probabilities form the feature vector for the word. The size of the feature for each word having n syllables is $2n - 3$, the sum of the number of bigrams and trigrams in the word. Figure 5.1 illustrates the feature creation and prediction in GMM and RNN.

5.2 Experiments

In order to create error words for training, we used the OCR outputs of Hindi, Gujarati, Malayalam and Telugu OCRs [6]. We used 5K document images from each language and used the OCR output collected from the respectieve OCRs. Recursive Text Alignment Tool (RETA) [52] is used to align the OCR output with the annotated ground truth text and extract the misrecognized words. We have ignored numbers, punctuations, special characters etc. which are not identified correctly by the OCR.

5.2.1 Corpus Creation

We have used a corpora consisting of seven popular Indian languages in our experiments. The corpus is collected by crawling various online newspapers and Wikipedia dump. For some languages, we have also relied on the crawled and maintained corpus available for download [45]. The details of the corpus used are shown in the Table 5.2 below. The crawled corpus needed extensive cleaning as they contained noise and unicode characters from other languages. For cleaning task, we maintained a list of *aksharas* which were manually separated into valid and invalid set. These were obtained by splitting words in the crawled corpus into *aksharas*. Any word which contained invalid *aksharas* were removed. We also removed words which contained unicode characters not belonging to the language. Most languages have variations in the word suffixes and sentence structure depending on different regions which use the same language. These dialects may not be covered when using the crawled corpus as it belongs to a more formal context. Due to the seeming intractability of this issue, we have relied on the crawled corpus obtained. In order to analyze the errors in OCR output, we have used OCR outputs from [6]. We have also analyzed the errors produced by two other OCR systems namely Tesseract and an RNN based OCR.

Corpus	English	Gujarati	Hindi	Kannada	Malayalam	Marathi	Tamil	Telugu
Total Words	6,026,940	4,328,769	18,180,174	3,889,172	5,999,978	4,188,321	10,949,052	3,193,283
Unique Words	233,158 (3.87%)	288,406 (6.66%)	288,535 (1.59%)	607,051 (15.61%)	1,016,856 (16.94%)	289,949 (6.92%)	1,022,403 (9.34%)	432,554 (13.54%)

Table 5.2: Details of the data set used for corpus and language model creation in various languages

5.2.2 Data and Evaluation Metrics

The details of the data used for training and testing using RNN and GMM is shown in table 5.3. We used a train-val-test split ratio of 64-16-20 in the experiments. In order to evaluate the error detection accuracy, we use True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) values. TP gives the percentage of errors correctly detected and TN gives the percentage of correct words rightly detected by the post processor. The FN value shows the undetected errors and its value goes up when there are more real word errors. When the right words are labeled as errors, the FP value increases. This can occur when the correct word pattern is not recognized by the post processor A good error detection system should give significant TP without generating much FP. This means that while all/most of the error words are detected correctly, the percentage of correct words labeled as errors should be kept minimum if not zero. We have used Precision, Recall and F-measure to compare the results of various approaches. Our aim is to maintain a high precision value because large number of correct words recognized as errors make the error detection module insignificant in post processing.

Language	Words for Training		Words for Testing	
	Errors	Correct	Errors	Correct
Hindi	81,632	89,196	20,308	22,299
Malayalam	966,16	137,171	24,155	34,293
Gujarati	150,825	171,730	37,706	42,932
Telugu	149,501	174,113	37,376	43,529

Table 5.3: Details of Training and Testing Corpus Size

5.2.3 Results of using RNN and GMM Methods

The results of error detection experiments using RNN is shown in table 5.4. While Malayalam, Gujarati and Telugu has comparable values of True Postives, a many errors went undetected in Hindi. Analyzing the results, we identify the presence of many valid words as errors. We behavior is seconded by the presence of large number of words at a particular Hamming distance [10] in Hindi. Therefore, when a character is mis-recognized by the OCR, there is a good chance that another valid word is formed, which is difficult to detect. Other False Positives include words which are not inherently found in the language such as names of people, places etc. The table 5.5 shows the results of both RNN and GMM

Language	TP	TN	FP	FN
Hindi	72.30	90.90	9.10	27.70
Malayalam	87.56	94.23	5.77	12.44
Gujarati	83.47	93.70	6.30	16.53
Telugu	80.34	95.69	4.31	19.66

Table 5.4: True Positive, False Positive, True Negative and False Negativer percentage for Languages

methods. While comparing the F-measure values of both the approaches, we can see that RNN based

Language	RNN			GMM		
	Precision	Recall	F-score	Precision	Recall	F-score
Hindi	89.30	77.22	82.82	85.46	77.44	81.25
Malayalam	93.82	87.56	90.58	88.47	84.70	86.54
Gujarati	92.98	83.47	87.97	92.05	80.28	85.77
Telugu	94.91	80.34	87.01	92.44	79.55	85.51

Table 5.5: Comparing Precision, Recall and F-Score values for RNN and GMM approaches. (The values are shown in percentage)

approach performs better than GMM. This can be attributed to the effective learning capability of neural networks. It is also observed that the Recall of Hindi is almost same in both the approaches. The effectiveness of both the approaches can be combined to build a powerful post processor.

5.2.4 Combining RNN and GMM Approaches

One of the important concerns in OCR post processing is the misclassification of correct words identified correctly by the OCR. The cost of mis-classifying a correct word in the OCR output as 'error' by the post processor is much higher than the cost of not identifying an error. This implies that we should be more concerned about increasing the precision. A good post processor should try to minimize the occurrence of False Positives while also trying to maximize the True Positives.

As observed in [24] relying on one method can fix some obvious errors but it can also increase the rate of hallucination of correct words as errors. We combine both our approaches to create a more reliable classifier wherein a word is declared as an error only if both the models label it as an error. The table 5.6 shows how a word is given a label from the labels of RNN and GMM approach.

RNN output	GMM output	Combined Approach Output
Error	Error	Error
Error	Right	Right
Right	Error	Right
Right	Right	Right

Table 5.6: Rules for labeling a word by combining the models

The results of the combined approach for different languages are shown in figure 5.2, 5.3, 5.4 and 5.5 for languages Malayalam, Hindi, Gujarati and Telugu respectively.

5.2.5 Observations

The error detection in OCR output using RNN and GMM gives us good detection accuracies. The primary reason for this is the exploitation of the potential of a neural network and complementing its predictions using a generative method. Also the use of *akshara* as the basic recognition unit of a word helps in learning the morphology of a word and patterns in word formation, enabling better prediction of labels of unseen words as errors or correct words. The method fails to detect correct words like person names or place names which are not related to the region where the language is used. Also detection of errors in punctuation and digits is a troublesome task. Overall, the approach succeeds in providing a fair solution for detecting non word errors in the OCR output. We evaluate the error detection accuracy on the output of RNN OCR and the results are shown in table 5.7. Some qualitative results of the combined approach are shown in figure 5.6.

We observed that long words like the Malayalam word, which are actually correct are identified correctly by the RNN. The GMM does well at picking up transliterated words from languages like English as shown in the Hindi example. It can be seen that combining both the models helps to reduce mis-classification of correct words.

Language	TP	TN	FP	FN	Precision	Recall	F-Score
Malayalam	75.57	85.38	14.62	24.43	83.79	75.57	79.47
Gujarati	67.39	94.83	5.17	32.61	92.87	67.39	78.11
Hindi	64.83	87.98	12.02	35.17	84.36	64.83	73.32

Table 5.7: Results of error detection on a RNN based OCR using the pre-trained error detection model. We have used a combination approach discussed in previous session to achieve this result. The values are in percentage.

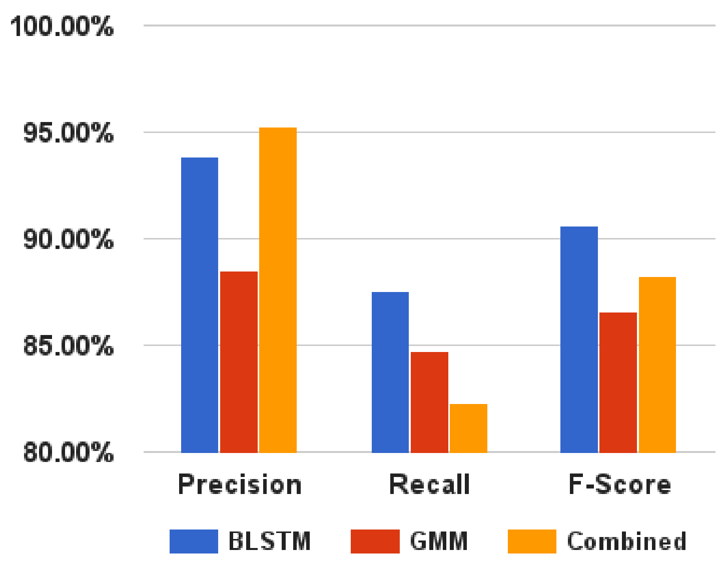


Figure 5.2: The bar graph shows the precision recall and F-score using RNN, GMM and the combined approach in Malayalam. The precision in the combined approach exceeds both the individual approaches.

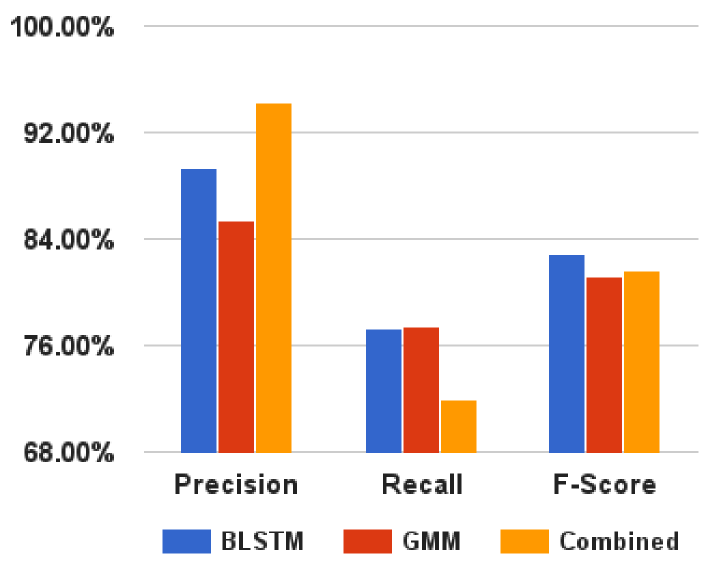


Figure 5.3: The bar graph shows the precision recall and F-score using RNN, GMM and the combined approach in Hindi. The precision in the combined approach exceeds both the individual approaches.

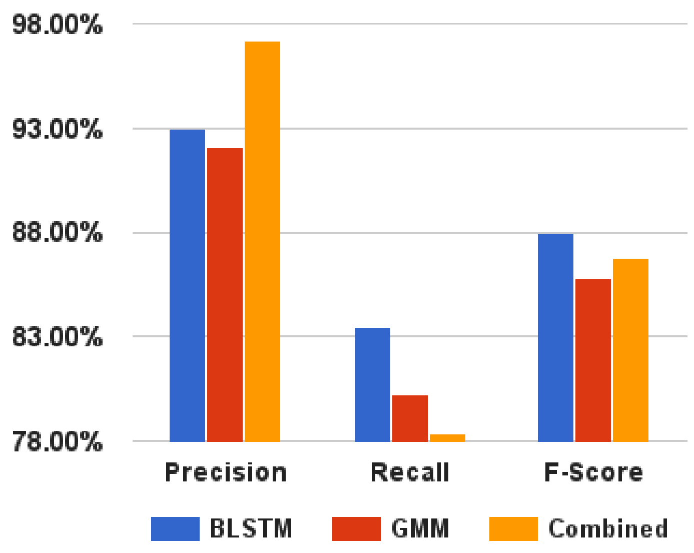


Figure 5.4: The bar graph shows the precision recall and F-score using RNN, GMM and the combined approach in Gujarati.The precision in the combined approach exceeds both the individual approaches.

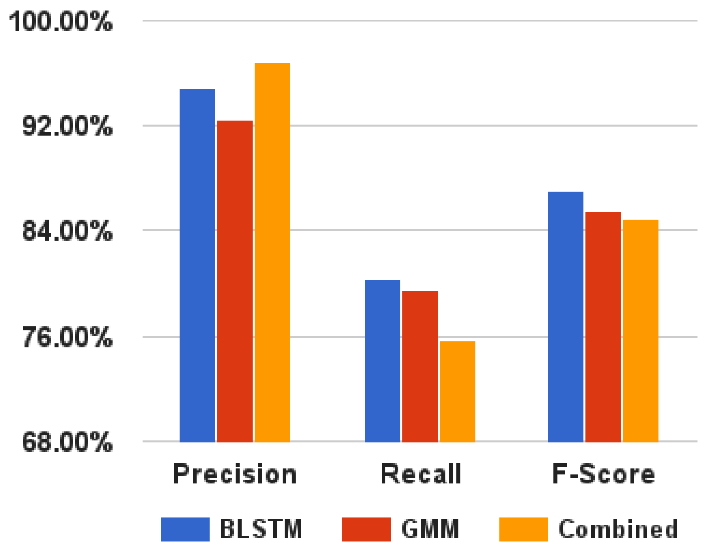
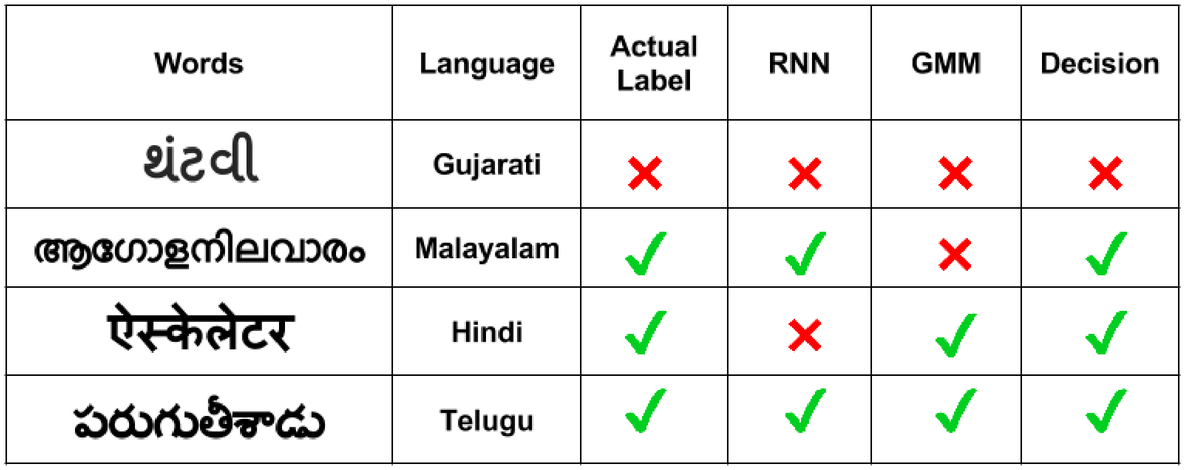


Figure 5.5: The bar graph shows the precision recall and F-score using RNN, GMM and the combined approach in Telugu.The precision in the combined approach exceeds both the individual approaches.



Words	Language	Actual Label	RNN	GMM	Decision
થંટવી	Gujarati	✗	✗	✗	✗
ആഗോളനിലവാരം	Malayalam	✓	✓	✗	✓
ऐस्केलेटर	Hindi	✓	✗	✓	✓
పరుగుతీశాడు	Telugu	✓	✓	✓	✓

Figure 5.6: Figure shows some of the test cases and the labels assigned to them by each model. Cross mark and tick mark indicates that the label is error and correct respectively. Decision column shows the prediction made by combined method.

5.3 Discussions

We have employed four techniques for error detection in shape recognizer output in this thesis. In Chapter 4 we have seen the use of dictionary for error detection. We also explored error detection using Statistical Language Models. In this chapter we have used two different error detection methods, using a Recurrent Neural Network which can learn sequences and a Gaussian Mixture Model based clustering method. In two methods employed for error detection, namely, RNN based method and SLM based method, we have used the bigram and trigram probability to decide if the word is an error word or not. In RNN based method, we used a machine learning model which uses bigram and trigram probabilities of *aksharas* as features to predict the presence of error. In SLM based model, we have used the average of bigrams and trigrams as a threshold to decide if the word is an error. We compare both the methods for error detection as both the methods relie on ngrams for error detection. In Figure 5.8, we walk through some qualitative results of the experiments done using both the methods.

To compare the two methods, we performed an experiment on a Hindi book in which we found that the RNN based method was able to achieve an f-score of 34% while SLM based method could achieve only 5%. It is observed that both the methods struggle to identify the correctness of transliterated words. This is an expected behavior since both the methods are based on the *akshara* ngrams and since the transliterated words (from English) do contain ngrams which are not commonly found in the language. An instance of this case is shown in the first word in Figure 5.8. We also observed that in Malayalam, RNN classifies most words with *chillu letters* as error words, mostly because the frequency of occurrence of these characters are less. In the third word shown in Figure, the agglutination of words resulted in formation of *akshara* which has less probability of occurrence in natural text. The SLM

No	Word	Language	Actual Label	RNN	SLM
1	സ്റേററുമെന്റുകളിൽ	Malayalam	Correct	Error	Error
2	അവൾ	Malayalam	Correct	Error	Correct
3	മുറിയിൽക്കയറിയിരുന്നു	Malayalam	Correct	Correct	Error
4	ഓരോരുരതരും	Malayalam	Error	Correct	Error
5	പൃവൻപഴം	Malayalam	Error	Error	Correct
6	हो, है	Hindi	Correct	Error	Correct
7	खाऊँगी	Hindi	Correct	Error	Correct
8	सतीश	Hindi	Correct	Correct	Error
9	लनय	Hindi	Error	Correct	Error
10	मोजन	Hindi	Error	Error	Correct
11	तुन्हारे	Hindi	Error	Correct	Error
12	కోవిలలో	Telugu	Correct	Correct	Error
13	అనుకుంటూ	Telugu	Correct	Correct	Error
14	రాజులుయు	Telugu	Error	Correct	Error

Table 5.8: The Table shows qualitative results comparing the SLM based method and the RNN based method for error detection in languages Malayalam, Hindi and Telugu. For each method, the prediction made for the word is seen. A word is predicted 'correct' or 'error'

identifies this correct word as error. Another observation is that the RNN also classifies words with single *akshara* as error. It may be because the training of RNN involved unique words. The system fails to recognize the correctness of such words which occur multiple times in natural scenarios. Some instances of such words are seen in the sixth row of the Figure. The SLM on the other hand classifies these words correctly. In most Indian languages, the independent vowels occur at the beginning of the word. However, in certain cases, independent vowels can occur at the middle of a word. Since these cases are very rare, the RNN could not learn this behaviour and classified the word in seventh row as error. The SLM could predict the correct class of this word. On the whole, we see that if the errors produces *aksharas* which are not seen previously, the SLM marks them as error easily. The RNN learns the patterns of word formation and use this to classify errors.

Chapter 6

Conclusion

In this thesis, we first analyzed the issues and challenges posed by Indian languages in the OCR pipeline. This is discussed in Chapter 3. The major issues were related to the large number of unique words existing in these languages. The complexity of scripts with touching glyphs was identified as another issue. We relied on *akshara* as the basic unit of a word as against unicode, which is popularly used. We compared the issues in Indic scripts with English, in which language, the OCR performance is laudable. In order to compare the languages, we relied on a crawled corpus obtained from various online resources like newspapers, Wikipedia etc. This step was necessary due to the lack of availability of a huge corpus in Indian languages. We compared the dictionary and Statistical Language Model based error detection and correction schemes in English with Indian languages. We found that dictionary is not an effective method for error detection in Indic scripts, especially Malayalam and Telugu, unlike in English. This is because creating a good dictionary was itself a difficult task, without which the whole method would fail. For error correction, we use a dictionary. We identify the closest words to a given word as replacement of that word. We make use of Levenshtein distance selecting the closest words from the dictionary. A human can then select the most appropriate word from the given options. *Akshara* level SLMs performed better than unicode level SLM because *akshara* formed a meaningful unit of a word and formation of invalid *aksharas* pointed the presence of errors.

Further, we discussed and implemented two kinds of error detection methods in Chapter 5. The first method is using a Gaussian mixture model based clustering technique where the probabilities of bigrams and trigrams of *akshara*s in correct words and error words were used as features. The second method was using a Recurrent Neural Network which is good at sequence learning tasks. Both methods provide reasonably good accuracy, detecting almost 80% of the errors in the OCR output. The RNN based method gives better performance than the clustering based method. We use a combination of both methods to reduce the occurrence of False Positives while also trying to maximize the True Positives.

6.1 Future Work

In the future we would like to try new features to train the neural network. Currently we only use the probabilities of *akshara* split bigrams and trigrams as features. We can make use of any development in the NLP capabilities to conquer issues related to inflection and agglutination. For instance we can use POS taggers to tag proper nouns which can then be treated as a special case in error detection and correction. We can also use word level ngram features to predict the real word errors in the OCR output. Also in the area of error correction, we can use word level ngrams to get more information about the most suitable word for replacement. This when used with the closest words from dictionary, selected using least Levenshtein distance can help in error correction. We can also extend this work to other language OCRs.

Related Publications

- Vinitha V S, C V Jawahar, ”*Error Detection in Indic* OCR*s*”, in Document Analysis Systems, 2016
- Vinitha V S, C V Jawahar, ”*An Emperical Study of Effectiveness of Post-processing in Indic Scripts*”, in MOCR, 2017 (*submitted*)

Bibliography

[1] “Census of India.” x, 8

[2] M. Cheriet, N. Kharma, C. L. Liu, and C. Y. Suen, *Character Recognition Systems*. John Wiley & Sons, 2007. 1

[3] “Tesseract Optical Character Recognition Engine .” 1

[4] “Abbyy finereader,” 1, 26

[5] A. Kumar and C. Jawahar, “Content-level annotation of large collection of printed document images,” in *ICDAR*, 2007. 2, 14, 15, 29, 31

[6] D. Arya, T. Patnaik, S. Chaudhury, C. V. Jawahar, B.B.Chaudhuri, A.G.Ramakrishna, C. Bhagvati, and G. S. Lehal, “Experiences of integration and performance testing of multilingual ocr for printed indian scripts,” in *ICDAR*, 2011. 3, 18, 19, 43

[7] N. Sankaran and C. Jawahar, “Recognition of printed devanagari text using blstm neural network,” in *ICPR*, 2012. 3, 15, 19

[8] R. Smith, “An overview of the tesseract ocr engine,” *ICDAR*, 2007. 3, 15, 26

[9] N. Sankaran, *Word recognition in Indic scripts*. MS thesis, IIIT Hyderabad, 2014. 4

[10] N. Sankaran and C. V. Jawahar, “Error detection in highly inflectional languages,” in *ICDAR*, 2013. 4, 7, 20, 21, 24, 27, 44

[11] “British National Corpus (BNC).” 5, 20, 27, 39

[12] J. Bentley, “Programming pearls: little languages,” *Communications of the ACM*, 1986. 5

[13] M. D. McIlroy, “Development of a spelling list,” *IEEE Transactions on Communications*, 1982. 5

[14] K. Kukich, “Techniques for automatically correcting words in text,” *ACM Comput. Surv.*, 1992. 5, 12

[15] Y. Bassil and M. Alwani, “Ocr post-processing error correction algorithm using google online spelling suggestion,” *arXiv:1204.0191*, 2012. 5

[16] A. Carlson and I. Fette, "Memory-based context-sensitive spelling correction at web scale," in *Machine Learning and Applications*, 2007. 6

[17] X. Tong and D. A. Evans, "A statistical approach to automatic ocr error correction in context," in *Proceedings of the fourth workshop on very large corpora*, 1996. 6, 26

[18] J.-C. Amengual and E. Vidal, "Efficient error-correcting viterbi parsing," *Pattern Analysis and Machine Intelligence*, 1998. 6

[19] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, 1948. 6, 28

[20] E. Brill and R. C. Moore, "An improved error model for noisy channel spelling correction," Association for Computational Linguistics, 2000. 6

[21] A. Wilcox-OHearn, G. Hirst, and A. Budanitsky, "Real-word spelling correction with trigrams: A reconsideration of the Mays, Damerau, and Mercer model," *Computational Linguistics and Intelligent Text Processing*, 2008. 6

[22] T. L. Packer, "Performing information extraction to improve ocr error detection in semi-structured historical documents," in *Historical Document Imaging and Processing*, 2011. 6

[23] R. Golding and Y. Schabes, "Combining trigram-based and feature-based methods for context-sensitive spelling correction," in *ACL*, 1996. 6

[24] R. Smith, "Limits on the Application of Frequency-Based Language Models to OCR," in *ICDAR*, 2011. 6, 45

[25] Y. Bassil and M. Alwani, "OCR context-sensitive error correction based on google web 1T 5-gram data set," *American Journal of Scientific Research*, 2012. 6, 31

[26] G. Lehal, C. Singh, and R. Lehal, "A shape based post processor for Gurmukhi OCR," in *ICDAR*, 2001. 6

[27] U. Pal, P. K. Kundu, and B. B. Chaudhuri, "OCR error correction of an inflectional indian language using morphological parsing," *Journal Of Information Science and Engineering*, vol. 16, 2000. 6

[28] K. Mohan and C. V. Jawahar, "A post-processing scheme for malayalam using statistical sub-character language models," in *DAS*, 2010. 6

[29] B. Chaudhuri and U. Pal, "OCR error detection and correction of an inflectional indian language script," in *Pattern Recognition*, 1996. 7

[30] M. Needleman, "The unicode standard," *Serials review*, 2000. 9, 10

[31] R. Ishida, “An introduction to indic scripts,” in *Proceedings of the 22nd International Unicode Conference*, 2002. 10

[32] K. Prahallad, V. Keri, S. Rajendran, and A. W. Black, “The IIIT-H Indic speech databases,” in *INTERSPEECH*, 2012. 10

[33] Y.-S. Hwang, B.-R. Park, H.-C. Rim, and S.-W. Lee, “A contextual post-processing model for Korean OCR using synthesized statistical information,” in *International Conference on Multimodal Interface*, 1999. 11

[34] A. Ganapathiraju, J. Hamaker, J. Picone, M. Ordowski, and G. R. Doddington, “Syllable-based large vocabulary continuous speech recognition,” *Speech and Audio Processing*, 2001. 11

[35] U. Pal and B. Chaudhuri, “Indian script character recognition: a survey,” *Pattern Recognition*, 2004. 18

[36] P. S. Natarajan, E. MacRostie, and M. Decerbo, “The bbn byblos hindi ocr system,” in *Electronic Imaging*, 2005. 19

[37] A. Bharati, P. Rao, R. Sangal, and S. M. Bendre, “Basic Statistical Analysis of Corpus and Cross Comparision,” in *ICON*, 2002. 20, 21, 39

[38] “Central Institute Of Indian Languages (CIIL) Corpus.” 20, 39

[39] P. Kuncham, K. Nelakuditi, S. Nallani, and R. Mamidi, “Statistical sandhi splitter for agglutinative languages,” in *Intelligent Text Processing and Computational Linguistics*, 2015. 24

[40] C. D. Manning, H. Schütze, *et al.*, *Foundations of statistical natural language processing*. MIT Press, 1999. 28, 29

[41] G. K. Zipf, “Relative frequency as a determinant of phonetic change,” *Harvard studies in classical philology*, 1929. 28

[42] G. K. Zipf, “Selected studies of the principle of relative frequency in language,” *Harvard University Press*, 1932. 28

[43] R. Rosenfeld, “Two decades of statistical language modeling: Where do we go from here?,” *Proceedings of the IEEE*, 2000. 28

[44] A. Stolcke *et al.*, “Srilm-an extensible language modeling toolkit.,” in *INTERSPEECH*, 2002. 29, 40

[45] D. Goldhahn, T. Eckart, and U. Quasthoff, “Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages.,” in *LREC*, 2012. 29, 31, 43

[46] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” in *Soviet physics doklady*, 1966. 32

[47] J. W. Ratcliff and D. E. Metzener, “Pattern-matching-the gestalt approach,” *Dr Dobbs Journal*, 1988. 32

[48] A. Zograf, *Languages of South Asia: a guide*. Routledge Kegan & Paul, 1982. 32

[49] W. Gale and G. Sampson, “Good-turing smoothing without tears,” *Journal of Quantitative Linguistics*, 1995. 40

[50] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, “A novel connectionist system for unconstrained handwriting recognition,” *Pattern Analysis and Machine Intelligence*, 2009. 41

[51] L. Medsker and L. Jain, “Recurrent neural networks,” *Design and Applications*, 2001. 42

[52] I. Z. Yalniz and R. Manmatha, “A fast alignment scheme for automatic ocr evaluation of books,” in *ICDAR*, 2011. 43