IDENTIFICATION AND CONVERSION ON FONT-DATA IN INDIAN LANGUAGES

Anand Arokia Raj, Kishore Prahallad

International Institute of Information Technology, Hyderabad, India.
Language Technologies Institute, Carnegie Mellon University, Pittsburg, USA.
anand@iiit.ac.in, skishore@cs.cmu.edu

ABSTRACT

To build a speech system like TTS (Text-to-Speech) or ASR (Automatic Speech Recognition) or an information extraction system like search engine, it is essential that the text has to be processed in Indian languages context. The text corpus or document database has to be in a single standard format. In this paper we discuss our effort in addressing the issues related to font-data like font encoding identification and font-data conversion in Indian languages.

Index Terms: Font, Font-Type, Font-Data, Font Converter, TF-IDF weights

1. INTRODUCTION

There is a chaos as far as the text in Indian languages in electronic form is concerned. Neither one can exchange the notes in Indian languages as conveniently as English language, nor one can perform search on texts in Indian languages available over web. This is so because the texts are being stored in ASCII (as apposed to Unicode) based font dependent glyph codes. Given that there are 25 official languages in India, the amount of data available in ASCII based font encoding is much larger (more dynamic also) then the text content available in ISCII [1] or Unicode [2] formats.

A large collection of such text corpus assumes major role in building large vocabulary speech recognition, unit selection based speech synthesis systems and search engine for Indian languages. So it becomes unavoidable to process text in ASCII based font encoding and convert it either to Unicode format or to a Roman character based transliteration scheme. Due to availability of more than one font encoding per language, the processing of ASCII based fonts needs to be done in two stages. The first stage involves identification of the underlying encoding and the second stage involves converting it into the required format.

This paper addresses such issues and provides solutions and explains it by organizing into three parts. The first part presents the nature of the Indian language scripts and their different storage formats. The second part presents a new TF-IDF sec 3.1 weights based approach to identify the font-types. The third part explains a generic framework for building font converters for Indian languages using glyph-map tables and glyph assimilation process.

2. NATURE OF INDIAN LANGUAGE SCRIPTS

The scripts in Indian languages have originated from the ancient Brahmi script. The basic units of the writing system are referred to as Aksharas. The properties of Aksharas are as follows: (1) An Akshara is an orthographic representation of a speech sound in an Indian language; (2) Aksharas are syllabic in nature; (3) The typical forms of Akshara are V, CV, CCV and CCCV, thus have a generalized form of C*V. Here C denotes consonant and V denotes vowel.

The shape of an Akshara depends on its composition of consonants and the vowel, and sequence of the consonants. In defining the shape of an Akshara, one of the consonant symbols acts as pivotal symbol (referred to as semi-full form). Depending on the context, an Akshara can have a complex shape with other consonant and vowel symbols being placed on top, below, before, after or sometimes surrounding the pivotal symbol (referred to as half-form).

Thus to render an Akshara, a set of semi-full or half-forms have to be rendered, which are in turn rendered using a set of basic shapes referred to as glyphs. Often a semi-full form or half-form is rendered using two or more glyphs, thus there is no one-to-one correspondence between glyphs of a font and semi-full or half-forms.

2.1. Convergence and Divergence

There are 25 official languages of India, and all of them except (English and Urdu) share a common phonetic base, i.e., they share a common set of speech sounds. While all of these languages share a common phonetic base, some of the languages such as Hindi, Marathi and Nepali also share a common script known as Devanagari. But languages such as Telugu, Kannada and Tamil have their own scripts.

The property that makes these languages separate can be attributed to the phonotactics in each of these languages rather than the scripts and speech sounds. phonotactics is the permissible combinations of phones that can co-occur in a language.

2.2. Digital Storage of Indian Language Script

Another aspect of diversion of electronic content of Indian languages is their format of digital storage. Storage formats like ASCII (American Standard Code for Information Interchange) based fonts, ISCII (Indian Standard code for Information Interchange) and Unicode are often used to store the digital text data in Indian languages. The text is rendered using fonts supporting these formats.

2.3. ASCII Format

ASCII is a character encoding based on the English alphabets. Digital computers and operating systems in the early 90s supported only ASCII based encoding and hence many electronic news papers in Indian languages used ASCII based fonts to store and render scripts of Indian languages. A font encoding stored in ASCII format specifies a correspondence between digital bit patterns and the symbols/glyphs of a written language. ASCII is, strictly, an eight bit code so ranges from 0 to 255.

2.4. Unicode Format

To allow computers to represent any character in any language, the international standard ISO 10646 defines the Universal Character Set (UCS) [2]. UCS contains the characters to practically represent all known languages in the world. ISO 10646 originally defined a 32-bit character set. Each character is assigned a 32 bit code. However, these codes vary only in the least-significant 16 bits.

UTF: A Universal Transformation Format (UTF) is an algorithmic mapping from every Unicode code point (except surrogate code points) to a unique byte sequence [3]. Actual implementations in computer systems represent integers in specific code units of particular size (8 bit, 16 bit, or 32 bit). Encoding forms specify how each integer (code point) for a Unicode character is to be expressed as a sequence of one or more code units. There are many Unicode Transformation Formats for encoding Unicode like UTF-8, UTF-16 and UTF-32. Both UTF-8 and UTF-16 are substantially more compact than UTF-32, when averaging over the world's text in computers. With the advent of Unicode, UTF-8 and their support in operating systems, most of the current electronic documents are being published in Unicode specifically in UTF-8 formats. Some of the news websites which produce Indian language content in Unicode format are: BBC news, Yahoo, MSN and Google.

2.5. ISCII Format

In India since 1970s, different committees of the Department of Official Languages and the Department of Electronics (DOE) have been developing different character encodings schemes, which would cater to all the Indian scripts. In 1983, the DOE announced the 7-bit ISCII-83 code , which complied with the ISO 8-bit recommendations [1]. ISCII (Indian Script Code for Information Interchange) is a fixed-length 8-bit encoding. The lower 128(0-127) code points are plain ASCII and the upper 95(160-255) code points are ISCII-specific, which is used for all Indian Script based on Brahmi script. This makes it possible to use an Indian Script along with Latin script in an 8-bit environment. This facilitates 8-bit bi-lingual representation with Indic Script selection code. The ISCII code contains the basic alphabet required by the Indian Scripts. All composite characters are formed by combining these basic characters. Unicode is based on ISCII-1988 and incorporates minor revisions of ISCII-1991, thus conversion between one to another is possible without loss of information.

2.6. Fonts and Glyphs

People interpret the meaning of a sentence by the shapes of the characters contained in it. Reduced to the character level, people consider the information content of a character inseparable from its printed image. Information technology, in contrast, makes a distinction between the concepts of a character's meaning (the information content) and its shape (the presentation image). Information technology uses the term ”character” (or ”coded character”) for the information content; and the term ”glyph” for the presentation image. A conflict exists because people consider ”characters” and ”glyphs” equivalent. Moreover, this conflict has led to misunderstanding and confusion. The technical report in [4] provides and expalins a framework for relating ”characters” and ”glyphs” to resolve the conflict because successful processing and printing of character information on computers requires an understanding of the appropriate use of ”characters” and ”glyphs”. It defines them as follow:

Character: A member of a set of elements used for the organization, control, or representation of data.

Coded Character Set: A set of unambiguous rules that establishes a character set and the relationship between the characters of the set and their coded representation.

Font: A collection of glyph images having the same basic design, e.g., Courier Bold Oblique.

Glyph: A recognizable abstract graphic symbol which is independent of any specific design.

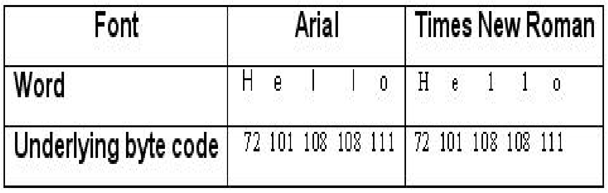
Indian language electronic contents are scripted digitally using fonts. A font is a set of glyphs (images or shapes) representing the characters from a particular character set in a particular typeface. Glyphs do not correspond one-for-one with characters. A font is or may be a discrete commodity with legal restrictions.

2.7. Need for Handling Font-Data

In the case of Indian languages, the text which is available in digital format (on the web) is difficult to use as it is because they are available in numerous encoding (fonts) based formats. Applications developed for Indian languages have to read or process such text. The glyphs are shapes, and when 2 or more glyphs are combined together form a character in the scripts of Indian languages. To view the websites hosting the content in a particular font-type then one requires these fonts to be installed on local machine. As this was the technology existed before the era of Unicode and hence a lot of electronic data in Indian languages were made and available in that form. The sources for these data are News websites (mainly), Universities/Institutes and some other organizations. They are using proprietary fonts to protect their data. Collection of these text corpora, identifying the type of encoding or font and conversion to font-data into a phonetically readable transliteration scheme is essential for building speech recognition and speech synthesis systems.

A character of English language has the same code irrespective of the font being used to display it. However, most Indian language fonts assign different codes to the same character. For example ’a’ has the same numerical code ’97’ irrespective of the hardware or software platform.

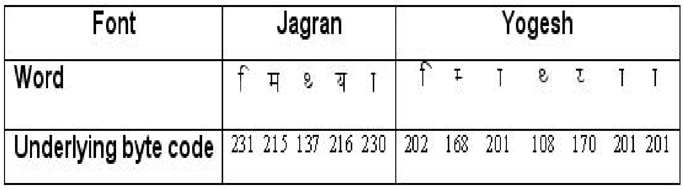
Consider for example the word ”hello” written in the Roman Script and the Devanagari Script.



Font	Arial	Times New Roman
Word	H e l l o	H e l l o
Underlying byte code	72 101 108 108 111	72 101 108 108 111

Fig. 1. Illustration of glyph code mapping for English fonts.

Arial and Times New Roman are used to display (Fig 1) the same word. The underlying codes for the individual characters, however, are the same and according to the ASCII standard.



Font	Jagran	Yogesh
Word	ि म [illegible] [illegible] ा	ि [illegible] ा [illegible] [illegible] ा ा
Underlying byte code	231 215 137 216 230	202 168 201 108 170 201 201

Fig. 2. Illustration of glyph code mapping for Hindi fonts.

The same word displayed (Fig 2) in two different fonts in Devanagari, Yogesh and Jagran. The underlying codes for the individual characters are according to the glyphs they are broken into. Not only the decomposition of glyphs and the codes assigned to them are both different but even the two fonts have different codes for the same characters. This leads to difficulties in processing or exchanging texts in these formats.

Three major reasons which cause this problem are, (i) There is no standard which defines the number of glyphs per language hence it differs between fonts of a specific language itself. (ii) Also there is no standard which defines the mapping of a glyph to a number (code value) in a language. (iii) There is no standard procedure to align the glyphs while rendering. The common glyph alignment order followed is first left glyph, then pivotal character and then top or right or bottom glyph. Some font based scripting and rendering is violating this order also.

2.8. A Phonetic Transliteration Scheme for Storage of Indian Language Scripts

To handle diversified storage formats of scripts of Indian languages such as ASCII based fonts, ISCII (Indian Standard code for Information Interchange) and Unicode etc, it is useful and becomes necessary to use a meta-storage format.

A transliteration scheme [5] [6] maps the Aksharas of Indian languages onto English alphabets and it could serve as metastorage format for text-data. Since Aksharas in Indian languages are orthographic represent of speech sound, and they have a common phonetic base, it is suggested to have a phonetic transliteration scheme such as IT3. Thus when the font-data is converted into IT3, it essentially turns the whole effort into font-to-Akshara conversion.

3. IDENTIFICATION OF FONT-TYPE

The widespread and increasing availability of textual data in electronic form in various font encoded form in Indian languages increases the importance of using automatic methods to analyze the content of the textual documents. The identification and classification of the text or text documents based on their content to a specific encoding type (specially font) are becoming imperative. Previous works [7] [8] [9] were done to identify the language and later to identify the encodings also. These works are based on statistical language modeling technique.

In this work we propose to use vector space model and Term Frequency - Inverse Document Frequency (TF-IDF) based approach for identification of Font-Type. To perform TF-IDF approach, it is essential to define what a ”term” is and what a ”document” is.

- Term: It refers to a unit of glyph. In this work we have experimented with different units such as single glyph g_i (uniglyph), two consecutive glyphs $g_{i-1}g_i$ (biglyph), three consecutive glyphs $g_{i-1}g_ig_{i+1}$ (triglyph).

- Document: Document refers the ’font-data (words and sentences) in a specific font-type’.

3.1. TF-IDF Weights

The TF-IDF weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document.

The term frequency in the given document is simply the number of times a given term appears in that document. This count is usually normalized to prevent a bias towards longer documents (which may have a higher term frequency regardless of the actual importance of that term in the document) to give a measure of the importance of the term t_i within the particular document.

$$tf_i = \frac{n_i}{\sum_k n_k} \tag{1}$$

with n_i being the number of occurrences of the considered term, and the denominator is the number of occurrences of all terms.

The document frequency is the number of documents where the considered term has occurred at least once.

$$|\{d : d \ni t_i\}| \tag{2}$$

The inverse document frequency is a measure of the general importance of the term (it is the logarithm of the number of all documents divided by the number of documents containing the term).

$$idf_i = log \frac{|D|}{|\{d : d \ni t_i\}|} \tag{3}$$

with $|D|$ total number of documents in the corpus

The effect of ’log’ in this formula is smoothing one.

$|\{d : d \ni t_i\}|$: Number of documents where the term ti appears (that is $n_i \neq 0$) Then

$$tfidf = tf.idf \tag{4}$$

A high weight in TF-IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms.

3.2. Modeling and Identification

Data Preparation: For training we need sufficiently enough data of that particular type. Thus we have collected and used around 0.12 million unique words per font-type. This data has been collected and prepared manually for 10 different fonts of 4 languages.

Modeling: Generating a statistical model for each font-type using these TF-IDF weights is known as modeling the data. For modeling we considered three different types of terms such as uniglyph, biglyph, triglyph (refer to Section 3).

The procedure for building the models is: First we have taken all the provided data at once. Also note that we have considered three different kinds of terms for building models. (i) First step is to calculate the term frequency Eqn (1) for the term like the number of times that term has occurred divided by the total number of terms in that specific type of data. So it will be stored in a matrix format of N $*$ 256 for uniglyph, N $*$ 256 $*$ 256 for biglyph and N $*$ 256 $*$ 256 $*$ 256 for triglyph model depending upon the term. Where ’N’ denotes the number of different font types and 256 (0 to 255) is the maximum number value for a single glyph. (ii) Second step is to calculate document frequency Eqn (2) like in how many different data type that specific term has occurred. (iii) Third step is to calculate inverse document frequency Eqn (3) like all data types divided by the document frequency. Logarithm of inverse document frequency is taken for smoothing purpose. (iv) Fourth step is to compute TF-IDF which is calculated like term frequency * inverse document frequency Eqn (4). Finally that matrix will be updated with these values. The common terms get zero values and other terms get non-zero values depending upon their term frequency values. From those values the models for each data type is generated.

Identification: The steps involved in identification of encoding type are as follows:

- Extract terms from the input word or sentence
- Create a query vector using these terms
- Compute the distance between query and all the models of font-type using TF-IDF weights
- The input word is said of be originated from the model of font-type which gives a maximum TF-IDF value
- Get the TF-IDF weight of each term from the models of all font-types

3.3. Performance Analysis

Test Data: It is typically observed that TF-IDF weights are more senstitive to the length of query. The accuracy increases with the increase in the length of test data. Thus two types of test data were prepared for testing. One is set of unique words and the other one is set of sentences.

Testing Criteria: While testing we are identifying the closest matching models for the given inputs. And we are evaluating the identification accuracy in (%) as given below.

$$Accuracy = \frac{Correct}{Total} \tag{5}$$

Where $Correct$: number of correctly identified test-inputs and $Total$: total number of test-inputs.

Testing: The accuracy of a font encoding identifier depends on various factors: 1) The number of encodings from

Table 1. Performance of Font-Type Identification using Uniglyph

Font Name	Identification for Sentences	Identification for Words
Amarujala (Hindi)	100%	100%
Jagran (Hindi)	100%	100%
Webdunia (Hindi)	100%	0.1%
SHREE-TEL (Telugu)	100%	7.3%
Eenadu (Telugu)	0%	0.2%
Vaarttha (Telugu)	100%	29.1%
Elango Panchali (Tamil)	100%	93%
Amudham (Tamil)	100%	100%
SHREE-TAM (Tamil)	100%	3.7%
English-Text	0%	0%

which the identifier has to select one, 2) The inherent confusion of one font encoding with another and 3) The type of unit used in modeling.

For a given 'X' number of different inputs we identified the closest models and calculated the accuracy. It is done (repeatedly) for various (uniglyph, biglyph and triglyph) categories. The results are tabulated in Table 1, 2 and 3.

Identification Results: The testing is done for 1000 unique sentences and words per font-type and evaluation results are tabulated below. We have added English data as also one of the testing data set, and is referred to as English-Text. Table 1 shows the performance results for uniglyph (current glyph) based models. We can observe that the uniglyph as a term fails to capture enough distinction among font types. Table 2 shows the performance results for biglyph ("current and next" glyph) based models and Table 3 shows the performance of triglyph ("previous", "current" and "next"). From Table 1, 2 and 3, it is clear that triglyph seems to be an appropriate unit for a term in the identification of font-type. It can also be seen that the performance at word and sentence level is nearly 100% with triglyph.

4. CONVERSION OF FONT-DATA

By font conversion we mean here the conversion of glyph to grapheme (akshara). So we want to make clear about glyph and grapheme. A character or grapheme is a unit of text, whereas a glyph is a graphical unit. In graphonomics, the term glyph is used for a non-character, i.e: either a sub-character or multi-character pattern. In typography, a grapheme is the fundamental unit in written language. Graphemes include letters, Chinese characters, Japanese characters, numerals, punctuation marks, and other glyphs.

Font-data conversion can be defined as converting the font encoded data into required phonetic transliteration scheme like IT3 based data. Previous works [10] [11] [12] tried the same problem in different manner for few languages. As we already have the notion that the characters are split up into glyphs in font-data so the solution would be merging up the glyphs to get back to the valid character.

Table 2. Performance of Font-Type Identification using Biglyph.

Font Name	Identification for Sentences	Identification for Words
Amarujala (Hindi)	100%	100%
Jagran (Hindi)	100%	100%
Webdunia (Hindi)	100%	100%
SHREE-TEL (Telugu)	100%	100%
Eenadu (Telugu)	100%	100%
Vaarttha (Telugu)	100%	100%
Elango Panchali (Tamil)	100%	100%
Amudham (Tamil)	100%	100%
SHREE-TAM (Tamil)	100%	100%
English-Text	100%	96.3%

Table 3. Performance of Font-Type Identification using Triglyph.

Font Name	Identification for Sentences	Identification for Words
Amarujala (Hindi)	100%	100%
Jagran (Hindi)	100%	100%
Webdunia (Hindi)	100%	100%
SHREE-TEL (Telugu)	100%	100%
Eenadu (Telugu)	100%	100%
Vaarttha (Telugu)	100%	100%
Elango Panchali (Tamil)	100%	100%
Amudham (Tamil)	100%	100%
SHREE-TAM (Tamil)	100%	100%
English-Text	100%	100%

Problem Statement: The problem of conversion of font-data could be stated as follows: Given a sequence of glyphs the objective is to combine them in a fashion so that they form a valid character.

A generic framework has been designed for building font converters for Indian languages based on this idea using glyph assimilation rules. The font conversion process has two phases, in the first phase we are building the Glyph-Map table for each font-type and in the second phase defining and modifying the glyph assimilation rules for a specific language.

4.1. Exploiting Shape and Place Information

As we have seen already in Sec. 2, the natural shape of a vowel or consonant gets changed when they become half or maatra. These symbols get attached in different places (top, bottom, left and right) around a pivotal consonant while forming an akshara. The novelty of our approach lies in exploiting positional information of a glyph in Akshara to perform the conversion. We exploit the positional information of a glyph in our approach for building the glyph-map table for a font-type. It is quite possible for a native speaker of the language by looking at the shape of the glyph to say whether that glyph occurs in center, top, left, right, bottom position of an Akshara. Thus we followed a simple number system to indicate the position information of a glyph.

- Glyphs which could be in pivotal (center) position get a code of 1.
- Glyphs which could be in left position of pivotal symbol get a code of 2.
- Glyphs which could be in right position of pivotal symbol get a code of 3.
- Glyphs which could be in top position of pivotal symbol get a code of 4.
- Glyphs which could be in bottom position of pivotal symbol get a code of 5.

Following figures depict a few examples in Hindi and Telugu about the position number assignment to glyphs. Figure (Fig 3) shows the position number assignment for glyphs in Hindi. In the top portion of the figure it shows the word considered here and right below it distinguishes different positions pictorially.

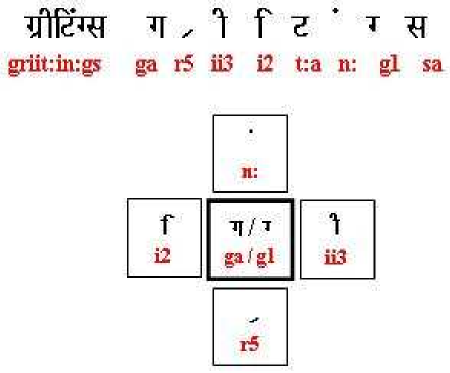


Fig. 3. Assigning position numbers for Hindi glyphs.

The second figure (Fig 4) shows the position number assignment for glyphs of Telugu language.

Independent vowels are assigned '0' or nothing as position number. All the dependent vowels known as "Maatras" occur at left or right or top or bottom side of a pivotal character (vowel). So they get attached a positional numbers like 2 (left), 3 (right), 4 (top) and 5 (bottom) always. This is common across all Indian languages. Most of the consonants will be either full or half and occur at center. So accordingly they are assigned 0 or 1 as positional number. Some of the half consonants also occur rarely at the left, right, top and bottom of a full character. They are all assigned a positional number like 2 (left), 3 (right), 4 (top) and 5 (bottom) according to their place.

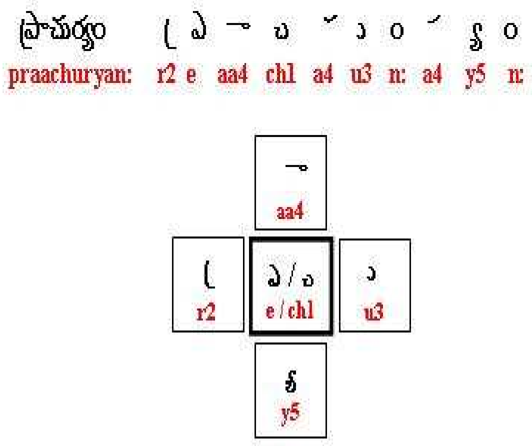


Fig. 4. Assigning position numbers for Telugu glyphs.

4.2. Building Glyph-Map Table

Glyph-Map table is a map table which gives a mapping between the glyph code (0 to 255) to a phonetic notation. This table gives us the basic mapping between the glyph coding of the font-type to a notation of a transliteration scheme. As a novelty in our approach we have attached some number along with the phonetic notation to indicate the place of occurrence of that glyph in that akshara. The way the position numbers are assigned is '0' or nothing for a full character (ex: e, ka), '1' for a half consonants (ex: k1, p1), '2' for glyphs occur at left hand side of a pivotal character (ex: i2, r2), '3' for glyphs occur at right hand side of a pivotal character (ex: au3, y3), '4' for glyphs occur at top of a pivotal character (ex: ai4, r4) and '5' for glyphs occur at bottom of a pivotal character (ex: u5, t5).

The position numbers along with the phonetic notation help us to form well distinguished glyph assimilation rules, because when similar glyphs get attached in different positions form different characters. For each and every font in a language the glyph-map table has to be prepared like that. That means the glyph-map table converts different font encoding to a meta data which is in a phonetic notation. It is observed that when we do such glyph-mapping for three different fonts of a language we have covered almost 96% of possible glyphs in a language.

4.3. Glyph Assimilation Process

Glyph Assimilation is defined as the process of merging two or more glyphs and forming a single valid character. In this way the split up parts of a character get merged to reveal the original character again. This happens in many levels like consonant assimilation, maatra assimilation, vowel assimilation and consonant clustering and in an ordered way.

The identification of different levels and ordering them are the another important thing we have done here. Broadly they can be classified into four levels. They are language preprocessing, consonant assimilation, vowel assimilation and schwa deletion. Under language preprocessing level, language specific modifications like halant and nukta modifications are carried out first. Because here afterwards there should not be any more symbols but only valid character glyphs are allowed. The next step is to reconstruct the pivotal consonant in an akshara if there is. This can be done under three sublevels like consonant assimilation, consonant and vowel assimilation and consonants clustering. Then we can form the vowels from the left out glyphs. Finally we have to do the schwa deletion because when a consonant and maatra merge up the inherent vowel (schwa) has to be removed.

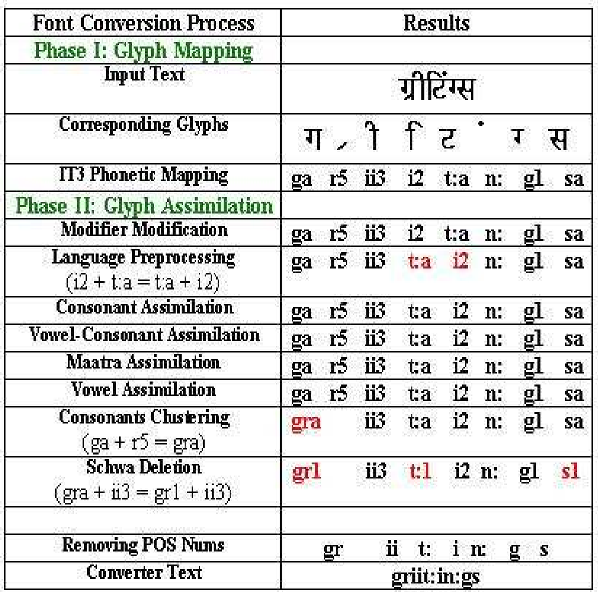
This assimilation process has been observed across many Indian languages and found that they follow certain order. The order is: (i) Modifier Modification, (ii) Language Preprocessing, (iii) Consonant Assimilation, (iv) Consonant-Vowel Assimilation, (v) Consonants Clustering, (vi) Maatra Assimilation, (vii) Vowel-Maatra Assimilation and (viii) Schwa Deletion. The concept is, revealing out the pivotal consonant in an akshara first then the vowels.

The first figure (Fig 5) shows how the font conversion is happening in Hindi with an example word. First phase gives the output in meta notation and the second phase re-orders, modifies and assimilates the glyphs based on rules. Finally the positional numbers get removed to reveal the converted word in a phonetic transliteration form.

The second figure (Fig 5) shows how the font conversion is happening in Telugu with an example word. First phase gives the output in meta notation and the second phase re-orders, modifies and assimilates the glyphs based on rules. Finally the positional numbers get removed to reveal the converted word in a phonetic transliteration form.

4.4. Rules for Glyph Assimilation

Glyph assimilation rules are defined by observing how the characters are being rendered by the rendering engine. Each rule takes a combination of two or more glyphs in a certain order and produces a valid character. Such way we have defined a set of rules for each level and for every language separately. Since they are written in the phonetic transliteration



Font Conversion Process	Results
Phase I: Glyph Mapping	
Input Text	ग्रीटिंग्स
Corresponding Glyphs	ग ्र ी ि ट ं ग् स
IT3 Phonetic Mapping	ga r5 ii3 i2 t:a n: g1 sa
Phase II: Glyph Assimilation	
Modifier Modification	ga r5 ii3 i2 t:a n: g1 sa
Language Preprocessing (i2 + t:a = t:a + i2)	ga r5 ii3 t:a i2 n: g1 sa
Consonant Assimilation	ga r5 ii3 t:a i2 n: g1 sa
Vowel-Consonant Assimilation	ga r5 ii3 t:a i2 n: g1 sa
Maatra Assimilation	ga r5 ii3 t:a i2 n: g1 sa
Vowel Assimilation	ga r5 ii3 t:a i2 n: g1 sa
Consonants Clustering (ga + r5 = gra)	gra ii3 t:a i2 n: g1 sa
Schwa Deletion (gra + ii3 = gr1 + ii3)	gr1 ii3 t:1 i2 n: g1 s1
Removing POS Nums	gr ii t: i n: g s
Converter Text	griit:in:gs

Fig. 5. Glyph Assimilation Process for Hindi.

scheme (IT3) it is easily to understand by anybody. There may be some common rules across many languages and some specific rules for a language also. The different rules under each and every category are explained with some examples below. These rules can be modified or redefined whenever it is required.

(i) *Modifier Modification* is the process where the characters get modified because of the language modifiers like virama and nukta. Ex:

(a) ta + halant = t1 (Fig 7)

(b) d'a + nuk = d-a (Hindi) (Fig 8)

(c) n: + halant = r1 (Telugu) (Fig 9)

(ii) *Language Preprocessing* steps deal with some language specific processing like Ex:

(a) aa3 + i3 = ri (Tamil) (Fig 10)

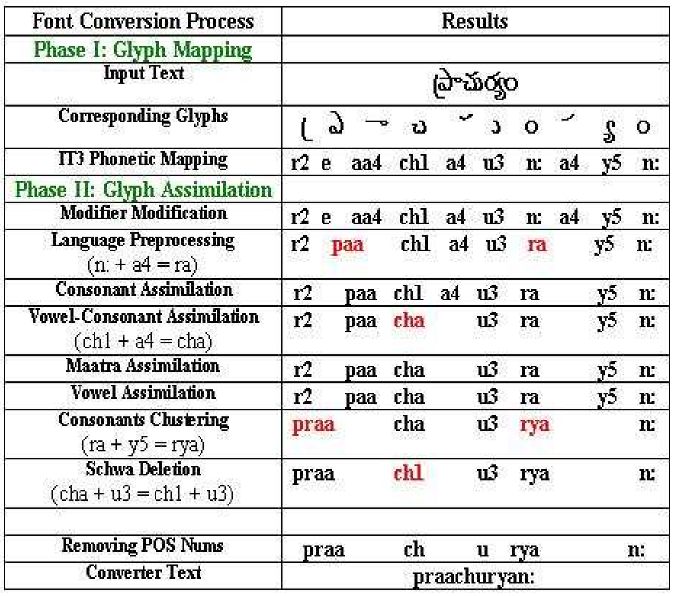
(b) r4 (REF) moves in front of the previous first full consonant (Hindi) (Fig 11)

(c) r3 moves in front of the previous first full consonant (Kannada) (Fig 12)

(iii) *Consonant Assimilation* is known as getting merged two or more consonant glyphs and forms a valid single consonant like Ex:

(a) d1 + h5 + a4 = dha (Telugu) (Fig 13)

(v) *Consonant-Vowel Assimilation* is known as getting merged two or more consonant and vowel glyphs and forms a valid single consonant like Ex:



Font Conversion Process	Results
Phase I: Glyph Mapping	
Input Text	ప్రాచుర్యం
Corresponding Glyphs	[illegible]
IT3 Phonetic Mapping	r2 e aa4 ch1 a4 u3 n: a4 y5 n:
Phase II: Glyph Assimilation	
Modifier Modification	r2 e aa4 ch1 a4 u3 n: a4 y5 n:
Language Preprocessing (n: + a4 = ra)	r2 paa ch1 a4 u3 ra y5 n:
Consonant Assimilation	r2 paa ch1 a4 u3 ra y5 n:
Vowel-Consonant Assimilation (ch1 + a4 = cha)	r2 paa cha u3 ra y5 n:
Maatra Assimilation	r2 paa cha u3 ra y5 n:
Vowel Assimilation	r2 paa cha u3 ra y5 n:
Consonants Clustering (ra + y5 = rya)	praa cha u3 rya n:
Schwa Deletion (cha + u3 = ch1 + u3)	praa ch1 u3 rya n:
Removing POS Nums	praa ch u rya n:
Converter Text	praachuryan:

Fig. 6. Glyph Assimilation Process for Telugu.

(a) va + uu3 = maa (Telugu) (Fig 14)

(b) kshh1 + aa3 = kshha (Gujarati) (Fig 15)

(c) y1 + u3 = ya (Kannada) (Fig 16)

(iv) *Maatra Assimilation* is known as getting merged two or more maatra glyphs and forms a valid single maatra like Ex:

(a) aa3 + e4 = o3 (Hindi) (Fig 17)

(b) e4 + ai5 = ai3 (Telugu) (Fig 18)

(c) e2 + e2 = ai3 (Malayalam) (Fig 19)

(vi) *Vowel-Maatra Assimilation* is known as getting merged two or more vowel and maatra glyphs and forms a valid single vowel like Ex:

(a) a + aa3 = aa (Hindi) (Fig 20)

(b) e2 + e = ai (Malayalam) (Fig 21)

(vii) *Consonant Clustering* in known as merging the half consonant which usually occurs at the bottom of a full consonant to that full consonant like Ex:

(a) ma + b5 = mba (Bengali) (Fig 22)

(b) r2 + tii = trii (Telugu) (Fig 23)

(c) ma + y3 = mya (Malayalam) (Fig 24)

(viii) The *Schwa Deletion* is deleting the inherent vowel ’a’ from a full consonant in necessary places like Ex:

(a) ka + o3 = ko

5. PERFORMANCE ANALYSIS

Data Preparation: In the phase of training, ’K’ different sets of words were prepared for each iteration. For testing 500 unique words were prepared.

Training: At first we build a simple converter with minimal rules. Then we pass the first set of words and get the output. Then we will ask the native speaker (or a linguist) to evaluate the output. He/she will provide the evaluation besides the correction for the wrongly converted words. Based on that we will define new rules or modify the existing rules. Then we will pass the next set of words and we will collect the feedback and modify the rules. This process will be continued for many iterations until we reach less or 0 conversion errors. Then the whole process is repeated for a new font of that language. At least we need to do this training for three different fonts of a language. At end of this training process we will be having the converter for that language.

Testing: We pass a set of 500 words from a new font of that language to the already built font converter. Again we ask the linguist to evaluate the output. We considered this is what the performance accuracy of that font converter.

Evaluation: We have taken 500 unique words per font-type and generated the conversion output. The evaluations results (table 4) show that the font converter performs consistently even for a new font-type. So it is only sufficient to provide the Glyph-Map table for a new font-type to get a good conversion results. In the case of Telugu, the number of different glyphs and their possible combinations are huge than other languages. Also it is common that the pivotal character glyph comes first and other supporting glyphs come next in the script. Whereas in Telugu some times the supporting glyphs come before the pivotal glyph which creates ambiguity in forming assimilation rules. Hence the converter performed lower than other converters. The conversion results are tabulated below.

6. RAPID FASHION OF BUILDING FONT CONVERTERS

The actual aim of this research is to find a method to build font converters rapidly for Indian languages. We have achieved this by following glyph assimilation process to build a single font converter per language. We want to make it clear that we developed converter per language and not per font. These font converters are basically working based on some set of glyph assimilation rules. And these rules are written on a meta notation known as positional phonetic notations. So the converters are independent of font but the glyph mapping tables are depended on fonts. If the user wants to convert a new font data then all he wants to do is prepare the glyph mapping table and pass that table and input text. For building the glyph map table for a native speaker hardly it takes one hour.

Table 4. Font conversion Performance.

Language	Font Name	Training/ Testing	Performance
Hindi	Amarujala	Training	99.2%
	Jagran	Training	99.4%
	Naidunia	Training	99.8%
	Webdunia	Training	99.4%
	Chanakya	Testing	99.8%
Marathi	Shree Pudhari	Training	100%
	Shree Dev	Training	99.8%
	TTYogesh	Training	99.6%
	Shusha	Testing	99.6%
Telugu	Eenadu	Training	93%
	Vaarttha	Training	92%
	Hemalatha	Training	93%
	TeluguFont	Testing	94%
Tamil	Elango Valluvan	Training	100%
	Shree Tam	Training	99.6%
	Elango Panchali	Training	99.8%
	Tboomis	Testing	100%
Kannada	Shree Kan	Training	99.8%
	TTNandi	Training	99.4%
	BRH Kannada	Training	99.6%
	BRH Vijay	Testing	99.6%
Malayalam	Revathi	Training	100%
	Karthika	Training	99.4%
	Thoolika	Training	99.8%
	Shree Mal	Testing	99.6%
Gujarati	Krishna	Training	99.6%
	Krishnaweb	Training	99.4%
	Gopika	Training	99.2%
	Divaya	Testing	99.4%
Punjabi	DrChatrikWeb	Training	99.8%
	Satluj	Training	100%
			99.9%
Bengali	ShreeBan	Training	97.5%
	hPrPfPO1	Training	98%
	Aajkaal	Training	96.5%
Oriya	Dharitri	Training	95%
	Sambad	Training	97%
	AkrutiOri2	Training	96%

7. CONCLUSION

This paper explained the nature and difficulties associated with font-data processing in Indian languages. We have discussed the new TF-IDF weights based approach for font identification. We have explained a framework to build font converters for font-data conversion from glyph-to-grapheme using glyph assimilation process. We have also studied the performance of the font identification for various cases. We have also demonstrated the effectiveness of our approach for font-data conversion to be as high as 99% on 10 Indian languages and for 37 different font-types.

8. REFERENCES

[1] ISCII, "ISCII - Indian Standard Code for Information Interchange," .

[2] Unicode Consortium, "Unicode - Universal Code Standard," .

[3] UTF, "UTF - Unicode Transformation Form," .

[4] ISO/IEC JTC1/SC18/WG8, "Document processing and related communication - document description and processing languages," 1997.

[5] G. Madhavi, M. Balakrishnan, N. Balakrishnan, and R. Reddy, "Om: One tool for many (indian) languages," *Journal of Zhejiang University Science*, 2005.

[6] P. Lavanya, P. Kishore, and G. Madhavi, "A simple approach for building transliteration editors for indian languages," *Journal of Zhejiang University Science*, 2005.

[7] K. Beesley, "Language identifier: A computer program for automatic natural-language identification on on-line text," 1988.

[8] William B., Cavnar, and John M. Trenkle, "N-gram-based text categorization," in *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, US, 1994, pp. 161–175.

[9] Anil Kumar Singh, "Study of some distance measures for language and encoding identification," in *Proceedings of the Workshop on Linguistic Distances*, Sydney, Australia, July 2006, pp. 63–72, Association for Computational Linguistics.

[10] Bharati Akshar, Nisha Sangal, Vineet Chaitanya, Amba P. Kulkarni, and Rajeev Sangal, "Generating converters between fonts semi-automatically," in *SAARC conference on Multi-lingual and Multi-media Information Technology*, 1998.

[11] Garg Himanshu, “Overcoming the font and script barriers among indian languages,” 2005.

[12] S. Khudanpur and C. Schafer, “Devanagari converters,” .

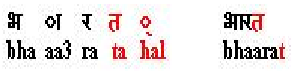


Fig. 7. Halant based Schwa Deletion.

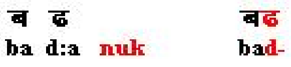


Fig. 8. Nukta based Modification.

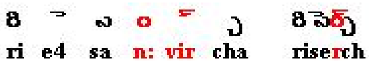


Fig. 9. Halant based Modification.

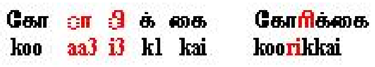


Fig. 10. Tamil r,ri,rii Modifications.

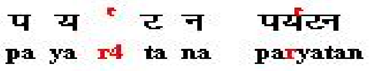


Fig. 11. Hindi REF movement.

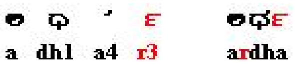


Fig. 12. Kannada arkkaa votthu movement.

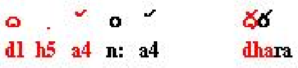


Fig. 13. Telugu da + ha assimilate to dha.

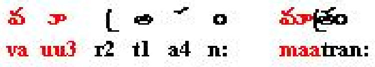


Fig. 14. Telugu consonant + maatra assimilation.

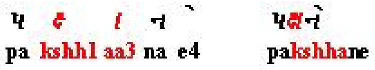


Fig. 15. Gujarati consonant + maatra assimilation.

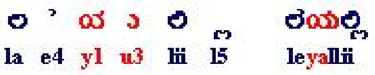


Fig. 16. Kannada consonant + maatra assimilation.

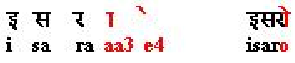


Fig. 17. Hindi maatra assimilation.

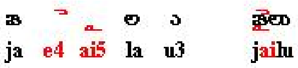


Fig. 18. Telugu maatra assimilation.

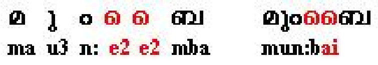


Fig. 19. Malayalam maatra assimilation.

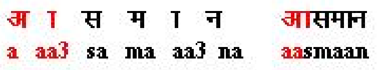


Fig. 20. Hindi vowel assimilation.

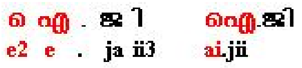


Fig. 21. Malayalam vowel assimilation.

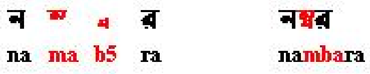


Fig. 22. Bengali b5 clustering.

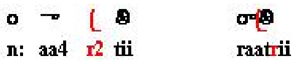


Fig. 23. Telugu r2 clustering.

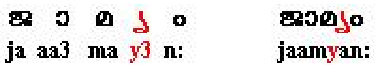


Fig. 24. Malayalam y3 clustering.