Query Word Labeling using Supervised Machine Learning: Shared task report by PESIT team

Channa Bankapur
Asst. Prof. of Computer Science
PES University
Bangalore, India
channabankapur@pes.edu

Adithya Abraham Philip
PES Institute of Technology
Bangalore, India
adithyaphilip@gmail.com

Saimadhav A Heblikar
PES Institute of Technology
Bangalore, India
saimadhavheblikar@gmail.com

ABSTRACT

The aim of this task is to identify words as belonging to an Indian language (L) or English (E) from sentences written in Roman script and if the word belongs to Indian language (L), transliterate the same to its Devanagari script equivalent. We have participated in the shared task where the Indian language is Hindi. We have used a supervised machine learning approach to classify words as belonging to either L or E and a state machine based approach to transliterate words.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Search and Retrieval

Keywords

Bilingual classification, transliteration

1. INTRODUCTION

Most Indian languages are written in the Devanagari script. The difficulty in representing Devanagari characters lies in the fact that they don't have one-to-one correspondence with their Roman script equivalents. We are forced to use the Roman script to attempt to phonetically replicate the corresponding word in Devanagari script. The ambiguity in this representation lies primarily in the fact that multiple combinations of letters in Roman script can be used to refer to the same word in Devanagari script. For example, “bharat” and “bharath” both refer to the same word in Devanagari script (meaning “India”). Besides, due to the different ways it is pronounced in different regions, there will be further ambiguity when the same word is to be represented in Roman script. Also due to socio-cultural reasons, there is bound to be a mix of languages even though the script used is the Roman script. The process of labeling is a two class classification problem. The process of representing words from a source language into the target language phonetic equivalent is called transliteration.

We have used training data from FIRE 2013 to train the classifier. Character n-grams have been used for this purpose. The transliterator has been manually built using language specific data available on the internet.

2. HINDI – ENGLISH QUERY WORD TRANSLITERATION:

Our system has been divided into two parts: the classification part and the transliteration part. The classification uses a supervised machine learning approach. First the query string q_1 $q_2 \dots q_n$ are fed to the classifier, where q_1, q_2, … , q_n are query words to be labeled. If the query word is English, there is no transliteration required. If the query word is of the Indian language L, then the word is fed to the transliterator which uses states to transliterate it from Roman script to Devanagari script.

2.1 Classification

The training data for the classifier was obtained from FIRE 2013 shared resources. The training data was processed to make a list of 29,934 Hindi words represented in Roman script. A similar list was made for English words using the Debian wordlist virtual package[4]. This wordlist has 71,971 words. It is to be noted that there is no relationship or correspondence between the two lists.

The two lists were separately processed as follows:

2.1.1 Extracting character n-gram frequency

The feature to train the classifier was chosen as character n-grams. Frequency data was obtained for character n-grams for both the lists for n = 1, 2, 3, 4, 5, and 6 including word boundaries. Including word boundaries means including the space character at the start of and at the end of the word. Space character was used as a delimiter in this case. This data was sanitized to remove overlapping cases at word boundaries between n and n-1 grams. This data was sorted and stored separately. No feature was discarded even though the feature list was huge.

2.1.2 Training the classifier

The processed data from the above step was used to train the classifier. A popular Python based machine learning library called scikit-learn 0.15.0[5] was used. The classification algorithm used was Multinomial Naive Bayes. The other classifier used was LinearSVC, which performed slightly worse as compared to the Multinomial Naive Bayes classifier in terms of accuracy. Performance-wise, there was no noticeable difference between the two, with both the classifiers taking approximately the same time to classify the same data set.

2.1.3 Classifying the words

In this step, the input query string was tokenized on common word boundaries like space, full stop, etc. Words containing characters other than alphabets were ignored. Each word was fed to the classifier built earlier to predict whether it belongs to the Indian language L or English E.

2.2 TRANSLITERATION

2.2.1 *Introduction*

The transliteration of a word consists of three steps: the Splitter, the Convertor, and the Generator.

The Splitter splits the word into separate vowel and consonant blocks, with the idea that each block represents a syllable.

The Convertor maps each of the blocks generated by the convertor to a set of possible Hindi equivalents, based on a set of rules.

The Generator generates all possible Hindi transliterations of the word, based on output from the Convertor, and selects one transliteration, based on a dictionary search, followed by picking the first word which found a match in case of multiple matches.

2.2.2 *The Splitter*

In most words, a set of continuous consonants or vowels represents a single syllable. e.g. in the words “dekha”, “hamare”, ”zindagi”.

By splitting a word into such blocks, we are required to define conversion rules only for single syllables, which help simplify the problem to a great degree.

This would result in the word “dekha” being split into blocks “d”-“e”-“kh”-“a”, each of which would be processed independently by the convertor.

There are certain exceptions to the one-syllable-per-block rule, which are discussed in, and handled by, the Convertor.

2.2.3 *The Convertor*

The Convertor takes input from the Splitter in the form of individual consonant or vowel blocks, and generates a list of possible Hindi equivalents of the input, as output.

The output of the Convertor is represented in an intermediary form consisting of groups of Roman script letters, representing specific symbols in Devanagari script of Hindi, delimited by “#” characters, such as “K” to represent the first Hindi consonant (pronounced “ka”) and “DU” to represent a dot to be placed under the succeeding consonant. The control character “!” is used to indicate that the preceding and succeeding consonants are not to be joined as half letters, which is the default behavior (in case “!” is absent”). This intermediary form allows for easier programming and debugging rather than directly using Unicode characters. The mapping has been given in the appendix.

There are two distinct stages in the Convertor, followed by exception handling.

In the first stage, the Convertor accepts a single consonant or vowel block, and the position of the block in the word (“beginning”, “middle” or “end”) as input. Using a set of pre-defined letter groupings as rules, and based on the position of the block in the word, it attempts to map each block to a set of similar-sounding matches in Hindi. For example, the block “a” present at the beginning of a word would be mapped to the first vowel of Hindi (which sounds like “aa”), while “a” present at the end of a word would be mapped to the “sign” (“matra”) of the first vowel in Hindi. The block “a” in the middle of a word, say “hamare”, would be mapped to two possibilities – the sign (“matra”) of the first vowel in Hindi, or “nothing”, where “nothing” (represented by “!”) means the preceding and succeeding syllables are not to be joined as half letters. “H#!#M” would sound like “hum” while “H#M” would be half of the “H” consonant joined with “M”. When more than one possible syllable equivalent exists for a block, the output from the Convertor is ordered with the most probable syllable equivalent first. This order is implemented in the part of the Convertor where we map the block to the syllable equivalent. For example, for the “a” block in the middle of the word “hamare”, we add the “!” to the output list before the sign (“matra”) of the first vowel of the Hindi language, as we believe “!” is more likely than the sign (“matra”) of the first Hindi vowel in this case. These rules, which include the order of syllables in the output list, have been decided based on human intuition and individual letter frequency charts[6]. These rules have been elaborately upon in the appendix.

If the block cannot be mapped in the first stage, the algorithm proceeds to the second stage. In this stage the word is compressed in a manner similar to Compressed Word Format (CWF)[1] by removing continuous sequences of the same vowel or consonant and replacing each sequence with one character. This would mean that “aao” or “aaaoo” after failing the first stage would be reduced to “ao” in the second stage.

In the case of consonant blocks, the second stage is also the stage where, based on position, “special” letters like “h” or “n” are used to determine whether certain letters require extra stress (in case of “h”), or require a dot on top (in case of “n”). These parts of the block once processed are removed from the block before further processing.

After the second stage, the unmapped parts of the block are sent to the first stage again. If it still cannot be mapped, it is classified as an exception and passed on to the exception handler as defined below.

The exception handler is meant to handle exceptions to the one-syllable-per-block rule, which can occur in two broadly classified groups – clearly enunciated words and ambiguously enunciated words.
Clearly enunciated words refer to Hindi words written in Roman script which clearly represent the enunciation of the corresponding Hindi word. The word “karvega” is an example of a word in this group while “krvega” is an example of the same Hindi word in a less clearly enunciated form and therefore not in this group.

Exceptions to the rule in the clearly enunciated group primarily occur in vowel blocks. Words such as “aao” will be considered as a single vowel block but should be represented as two syllables “aa” and “o”. In the case of “aao”, the block would already have been reduced to “ao” (CWF) by the second stage, and if no existing rules map the block “ao”, the Convertor would assume that each letter in the block represents a different syllable. It would map them by recursively calling itself, passing “a” and “o” as two separate blocks, both at the “beginning” position.

Hindi words improperly/ambiguously enunciated in Roman Script include words like “krvega”, which is an ambiguously enunciated equivalent of “karvega” and is harder to handle, as the consonant block “krv” represents as a single block what we should have ideally interpreted as three blocks – “k”, “a” and “rv” However, such constructs are popular among users of social networking sites and must be taken care of. In its current state, the Convertor guesses that all letters in an unmapped consonant block represent individual half letters and maps each individual letter to the corresponding half-consonant (except the last letter, which is assumed to be a "full" consonant).

The output from the Convertor is an ordered list of syllables in Hindi which sound like the given block at the given position in the word, with the most probable syllable equivalent first. This output is used by the Generator.

2.2.4 The Generator

A set of words is generated, by listing all possible combinations from the set of possible syllable mappings produced for each block by the Convertor. These words are then passed through a dictionary filter to remove non-existent combinations. In the case of exactly one word from the list of words being present in the dictionary, it is given as output. In the case of more than one word finding dictionary matches, the first match among them, based on how the Convertor orders its output (most probable first) is chosen and given as output. In case of no matches, the dictionary is not considered and the most first word from the entire unfiltered list is chosen, once again based on the ordering from the Convertor.

After the word is chosen, it is mapped from the intermediary form used in the Convertor to its Hindi Devanagari Script equivalent.

3. RESULTS

The token level precision, recall and F-measure for Hindi were found to be 0.81, 0.813 and 0.812 respectively, and for English was found to be 0.767, 0.798 and 0.782. The token level language labeling accuracy was 0.656, while EQMF[7], and ETPM[7] were found to be 0.001 and 0.5737 respectively. EQMF without transliteration – With both named entities and MIX tags, without named entities, without MIX tags, and without both named entities and MIX tags were 0.158, 0.258, 0.158, and 0.258 respectively.

4. DRAWBACKS

4.1 Classification

4.1.1 Undefined behavior for certain queries

The classifier was found to exhibit undefined behavior for certain words which are "valid" in both languages (Hindi and English in our case). Examples of such words are "do", "hai", "to" etc. The classifier's prediction varied on each run for such queries. One possible reason is that such a word is equally probable of belonging to both the classes, in which case the prediction may be randomly decided.

4.1.2 Classifying named entities

The system was unable to detect named entities as it was not trained for this task. This prevented it from classifying them. Named entities in queries were not treated as such. They were treated as normal words belonging to either Indian language L or English E.

4.2 Transliteration

4.2.1 Lack of a Machine Learning System

No machine learning system was used, and hence all rules were written based on manual evaluation of data such as frequency charts[6]. This prevents the use of training data to improve the system and makes it prone to human error in laying down the rules. One example is when the rules for an "a" vowel block in the middle of a word was set – we initially gave the sign("matra") of the first vowel of Hindi higher preference as compared to the "!" ("nothing") character (the former was ordered first in the output list of the Convertor, and the latter second). We later found, by trial and error, that "!" should have been given higher priority in the output order of the Convertor, and we had to manually change the rule to make "!" come before the sign("matra") of the first vowel in Hindi in the output list of the Convertor.

5. SCOPE FOR IMPROVEMENT

5.1 Word Sense Disambiguation

The drawback described in 4.1.1 can be approached by having in place a system which can identify the sense of the word. This could possibly be done by looking at the words around the sentence or having the system learn common patterns of language use in a bilingual space. An example of such a pattern is L-L-E-L-E. Then based on such data of patterns, we may be able to identify such words.

5.2 Introduction of a Machine Learning System in Transliteration

Allow an unsupervised learning system to be used in the Convertor to map the blocks generated by the Splitter to their syllable equivalents. This would eliminate one of the biggest weaknesses of our model – having the rules which map blocks to syllables being set by humans based on nothing but human intuition assisted by Hindi letter frequency charts.[6] For example, we would leave it to the machine learning system to decide what an "a" block in the middle of a word or an "ee" or "ch" block at the end of a word should be mapped to, and not specify any rules ourselves.

6. ACKNOWLEDGEMENTS

We would like to thank the Indian Statistical Institute for organizing FIRE - 2014, and giving us the opportunity to work on this transliteration system. We would also like to thank Microsoft Research India team for their participation in the same, whose data and evaluation helped us better understand our strengths and weaknesses.

7. REFERENCES

[1] Srinivasan C. Janarthanam, Sethuramalingam S, and Udhyakumar Nallasamy. Named entity transliteration for cross-language information retrieval using compressed word format mapping algorithm. In iNEWS '08: Proceeding of the 2nd ACM workshop on improving non english web searching, pages 33–38, New York, NY, USA, 2008. ACM.

[2] Parth Gupta, Kalika Bali, Rafael E. Banchs, Monojit Choudhury, and Paolo Rosso. 2014. Query expansion for mixed-script information retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval* (SIGIR '14). ACM

[3] P. J. Antony and K. P. Soman, Machine Transliteration for Indian Languages: A Literature Survey. In *International Journal of Scientific & Engineering Research*, Volume 2, Issue 12, December-2011

[4] Debian wordlist virtual package - https://packages.debian.org/sid/wordlist

[5] Scikit-learn version 0.15.0 http://scikit-learn.org/stable/

[6] http://www.sttmedia.com/characterfrequency-hindi

[7] R.S. Roy, M. Choudhury, P. Majumder, K. Agarwal. Overview and Datasets of FIRE 2013 Track on Transliterated Search. FIRE @ ISM. 2013

APPENDIX

A. INTERMEDIARY FORM MAPPING

DT	◌ं	DU	◌़
iii	◌ी	THH	थ
SHH	ष	EEE	ऑ
DD	ढ	UU	ऊ
II	ई	oo	◌ौ
eee	◌ै	AA	आ
PP	फ	EE	ऐ
III	ई	TT	ठ
D	ड	E	ए
G	ग	A	अ
JJ	झ	B	ब
C	च	TH	त
L	ल	M	म
N	न	O	ओ
H	ह	I	इ
J	ज	K	क
U	उ	T	ट
W	व	V	व
BB	भ	P	प
S	स	uu	◌ू
DD2	ध	R	र
Y	य	e	◌े
SH	श	KK	ख
a	◌ा	ii	◌ी
o	◌ो	OO	औ
i	ि◌	u	◌ु
ee	◌ै	CH	छ
D2	द	HALF	◌्
GG	घ		

B. SAMPLE SET OF RULES

For Single Character Consonant Block occurring anywhere (not all rules included, due to limited space):

```
If ‘b’:
        "B"
        "BB"/*adds “B” and “BB” in that order*/
Else if ‘c’ or 'k':
        "K"
Else if 'd':
        "D2"
        "D"
        "DD2"
Else if 'f':
        "PP"
Else if 'g':
        "G"
        "GG"
Else if 'h':
        "H"
Else if 'j':
        "J"
        "JJ"
Else if 'l':
        "L"
Else if 'm':
        "M"
Else if 'p':
        "P"
Else if 'r':
        "R"
Else if 's':
        "S"
Else if 't':
        "TH"
        "T"
        "TT"
Else if ‘v’ or 'w':
        "W"
Else if 'y':
        "Y"
Else if 'z':
        "J#DU"
```

For Double Character Vowel Blocks occurring at beginning of word (not all rules included, due to limited space):

```
If “aa”:
        “AA”
Else if “ai”:
        “EE”
Else if “au”:
        “OO”
Else if “ee”:
        “II”
Else if “ea”:
        “I”
        “II”
Else if “eu”:
        “y#uu”
        “y#u”
```