Named Entity Recognition for Indian Languages

Animesh Nayan, B. Ravi Kiran Rao, Pawandeep Singh,
Sudip Sanyal and Ratna Sanyal
Indian Institute of Information Technology
Allahabad, India
e-mail@domain

Abstract

Abstract Stub This paper talks about a new approach to recognize named entities for Indian languages. Phonetic matching technique is used to match the strings of different languages on the basis of their similar sounding property. We have tested our system with a comparable corpus of English and Hindi language data. This approach is language independent and requires only a set of rules appropriate for a language.

1 Introduction

Named Entity Recognition (NER) is a subtask of machine translation and information retrieval. Named entities are words which belong to certain categories like persons, places, organizations, numerical quantities, expressions of times etc. A large number of techniques have been developed to recognize named entities for different languages. Some of them are Rule based and others are Statistical techniques. The rule based approach uses the morphological and contextual evidence (Kim and Woodland, 2000) of a natural language and consequently determines the named entities. This eventually leads to formation of some language specific rules for identifying named entities. The statistical techniques use large annotated data to train a model (Malouf, 2002) (like Hidden Markov Model) and subsequently examine it with the test data. Both the methods mentioned above require the efforts of a language expert. An appropriately large set of annotated data is yet to be made available for the Indian Languages. Consequently, the application of the statistical technique for Indian Languages is not very feasible.

This paper deals with a new technique to recognize named entities of different languages. Our approach does not use the previously mentioned techniques. Instead, we use an approach that not only reduces the burden of collecting and annotating data, but is language independent as well. We use this method to build a multilingual named entity list that can be used by the named entity recognizer. Our method recognizes and finds the actual representation of the named entities in the target language from an untagged corpus. Our idea was to match the two representations of the same named entity in two different languages using a phonetic matching algorithm. This comes from the property of named entities that they sound similar when written in native script or any other script. However this cross-lingual matching is not a trivial task. First of all, the two strings to be matched have to be represented in a common script. So we face two choices here. Either we should convert the two strings into some common intermediate representation (ex. Phonemic representation) or transliterate the name written in Indian language to English and then look for phonetic equivalence. Our engine has been tested for Hindi. After making transliteration rules for Hindi, we used a variation of the Editex algorithm to match the transliterated string with entries in English named entity database to find a match. Here it is worthwhile to mention that certain class of name entities which are not similar sounding (mostly phrases) cannot be extracted through this cross-lingual matching. E.g. “United Nations”, “Government of India” etc. Abbreviations which are spelled character by charac-

Proceedings of the IJCNLP-08 Workshop on NER for South and South East Asian Languages, pages 97–104,
Hyderabad, India, January 2008. ©2008 Asian Federation of Natural Language Processing

ter in both the languages can however be extracted. E.g. BBC (बीबीसी), LTTE (एलटीटीई) etc.

In the next section we have given the system architecture. The logical flow and overall description of the system are discussed here. Our own set of transliteration rules in Hindi are given in the third section. In the fourth section we define our baseline task. Our system has been tested with a parallel corpus which consisted of both English and Hindi language data. The results obtained using our system is described in the fifth section together with an analysis. Conclusions are presented in the last section together with directions for future improvements.

2 System Architecture: Logical Flow and overall description of the System

The system architecture is shown in Figure 1. It consists of the following modules:

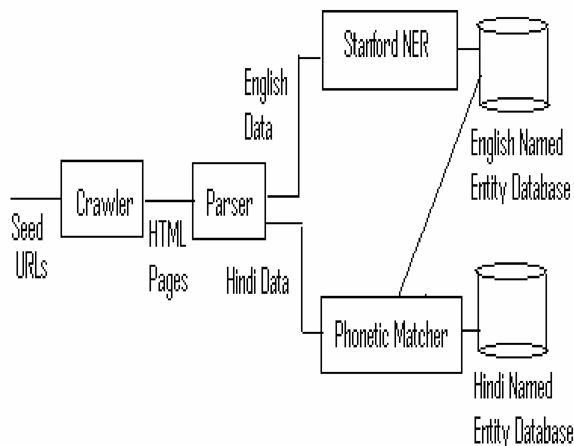

Figure 1: System Architecture

2.1 Crawler

The crawler is a web-bot or spider which browses the web in an automated manner. It starts with a list of Uniform Resource Locators (URL) that it is to visit, called the seeds. As the crawler visits these URL's it collects all the hyperlinks and adds them to a queue. URL's from the queue are crawled further. Since the crawler collects the data from web, the data collection is fully automated. The crawler gathers data for both English and other Indian languages. The data collected for English is used to populate the English named entity database which is significantly accurate. We have used the freely available Stanford Named Entity Recognizer (Finkel, Grenager, and Manning, 2005) in our engine. The data collected for Indian languages will be used to build a database of named entities for the given language.

2.2 Parser

The crawler saves the content in an html form onto the system. The parser parses these html files. Additionally the parser can also parse the PDF as well as RTF files. The output of the parser is passed to the corresponding modules for the two different languages.

2.3 Phonetic Matcher

Phonetic matching is the task of matching two representations of the same name. A name may have more than one representation in its native script itself. If the name is represented in a script other than its native script, there may be large number of potential variants for its representation. Phonetic matching is a fuzzy string matching technique in which we match strings on the basis of their similar sounding property and not identity. Most common phonetic matching techniques are Soundex and Editex. These techniques are used to match two representations of the same name in English. We survey the techniques in the following subsections.

2.3.1 Soundex

Soundex algorithm was designed by Odell and Russell in 1918 to find spelling variation of names. It represents classes of sounds which can be lumped together. The classes for the algorithm are shown in Appendix A. These classes are placed for phonetic matching according to the following algorithm:

1. Replace all but the first letter of the string by its phonetic code.
2. Eliminate any adjacent representation of codes.
3. Eliminate all occurrences of code 0 i.e. eliminate all vowels.
4. Return the first four characters of the resulting string.
5. Examples: Dickson = d25, Dikson = d25.

Two names match if they have the same soundex representation. This method does not account

for vowels and hence is not accurate for cross-lingual matching.

2.3.2 Editex

The Editex algorithm was designed by Zobel and Dart (Zobel and Dart,1996). It is an enhancement of the Levenshtein (Levenshtein, 1966) edit distance algorithm. The Levenshtein algorithm measures the edit distance between two strings where edit distance is defined as the minimum number of basic operations required to match one string to the other where the basic operations are insertion, deletion and substitution. Insertion and deletion costs are 1 and substitution cost is given by a function subst_cost (Xi, Yj) which returns 0 if the two characters Xi and Yj are same and 1, if they are different. The score dist [m, n] is returned as the edit distance between two strings. A score of zero implies a perfect match.

The algorithm has O (mn) time and space complexity where m and n are the lengths of the two strings respectively. The pseudo code for the Levenshtein edit distance algorithm is described in Appendix B. Editex groups similar sounding phonemes into equivalence classes. The substitution cost is determined by a function S (Xi, Yj) that returns 0 if the two characters Xi and Yj are same, 1 if they lie in the same equivalence class and 2 otherwise. The insertion and substitution costs are determined by a function D (Xi-1, Xi) which is almost same as S (Xi, Yj) except for the difference that it compares letters of the same string and it returns 1 if Xi-1 is 'h' or 'w' and Xi-1 is not equal to Xi. The editex equivalence classes and the editex pseudo-code are given in Appendix C.

Editex performs fairly better than Soundex and Leveinshtein edit distance algorithms. However further enhancements in Editex are also possible. "Tapering" is one enhancement in which we weigh mismatches at the beginning of the string with higher score than mismatches towards the end (Zobel and Dart, 1996). Other enhancements are those in which input strings are mapped to their phonemic representation, called phonometric methods (Zobel and Dart, 1996).

3 Transliteration rules

To perform phonetic matching of two different representations of a named entity, we need both of them in a common script. We choose to transliterate the named entity in Indian language to English. The transliteration rules for a language must be written for the same. We have written our own set of transliteration rules for Hindi. These can be described briefly as under

The entity to be transliterated is scanned character by character from left to right. Each character of Hindi is mapped to an equivalent character/set of character in English according to a mapping function. The character set generated by the function is appended into a string as per the rules. E.g. का = क् + अ is a single character representation in Unicode ('क') and maps to 'Ka'.

1. Start with an empty string. When a consonant or singleton vowel (not as 'matra') is encountered append the set of characters returned by mapping function.
2. When a consonant is followed by a vowel the preceding 'a' should be removed and the character set for the vowel should be appended. E.g. के consists of two characters क + े. Once we encounter क we append 'ka' and when े is encountered next we remove the 'a' and append the mapping for े i.e. 'e'. This rule applies in general to all the vowels.
3. If the transliterated string has 'a' as its last character while it doesn't have the vowel ा as last character of Hindi string, remove this occurrence of 'a'. The last vowel in Hindi is very important as two altogether different words may have the only difference in the last vowel. E.g. "कमल" and "कमला" are proper nouns having different genders. Their English representations are "Kamal" and "Kamla" respectively.

The transliteration always performs a one to one mapping of a character in Hindi to a set of characters in English. However the English representation may have different character sets for the same Hindi character in different names. E.g. "कमल" is "Kamal" while "क्रिकेट" is "**Cricket**". 'क' is often represented by 'K' for Hindi names, by 'C' for

English names and by ‘Q’ for Urdu names. The Editex algorithm groups these letters in the same equivalence class.

4 Baseline Task

At the core of our method lies the phonetic matching algorithm. We have modified the Editex algorithm as mentioned in Appendix C. Editex can be modified to take into account that there can be more than three (0, 1, 2) levels of acceptability for substitutions due to the inherent properties of particular languages. For example, say “ckq” is one equivalence class in Editex. ‘c’ and ‘k’ have a substitution cost of 1. We may reduce this substitution cost to 0.5 for a language in which it is highly probable that the same character maps to ‘c’ and ‘k’ in the English representation of its names. Thus the equivalence classes and the substitution costs in Editex can be modified for cross-lingual phonetic matching. There can also be further language specific enhancements. The following algorithm along with some language specific enhancements was implemented for Hindi.

4.1 Abbreviation Check

Abbreviations form an important class of named entities. So, we first check whether the Hindi string is an abbreviation in which the English characters are spelled individually. For each English alphabet we have some unique Hindi representation. The function performs accurately most of the time and extracts such named entities. If we are able to find out that the string is an abbreviation, the corresponding English representation can be returned by the function itself, hence there is no need of further matching. If the string is not an abbreviation, we proceed to the actual matching algorithm.

4.2 4.2. First letter matching

The first letters of the two strings must either be the same or should belong to the same equivalence class. The equivalence classes for first character matching are:

"ckq", "wbv", "iy”,"jz", "aeiou"

The English named entity database must be indexed according to the first letter of the named entity so that we only search for matches in those indexes which fall into the same equivalence class. This is very important for the computational efficiency of the engine as it reduces the search space.

4.3 Preprocessing

Often the phonetic inconsistencies in English lead to low matching score for two representation of the same name. To take this into account, before matching the two strings the named entity retrieved from English Named entity database is preprocessed to form a new string. We have used the famous “Mark Twain’s plan for the improvement of English spelling” (http://grammar.ccc.commnet.edu/grammar/twain.htm) added with some more rules. This way we tackle the problem of more than one possible character sets for some vowels since only one of them can be chosen during transliteration. We also tackle some other problems like silent-alphabets and repeated alphabets so that the probability of generating high matching score increases. The following set of rules for preprocessing was used.

1. Change all occurrences of “oo” to “u”. (both character sets are for the vowel ◌ू)
2. Change all occurrences of “ee” to “i” (both character sets are for the vowel ◌ी)
3. Change all occurrences of “f” to ph”
4. Change all occurrences of “au” to “o”
5. If a word starts with "x", replace the "x" with a "z". Change all the remaining "x"s to "ks"s.
6. If a "c" is directly followed by an "e" or "i", change the "c" to an "s"
7. If a "c" is directly followed by a "k", remove the "c". Keep applying this rule as necessary (Example: "cck" becomes "k".)
8. If a word starts with "sch", change the "sch" to a "sk".
9. If a "ch" is directly followed by an "r", change the "ch" to a "k".
10. After applying the above rules, change all "c"s that are not directly followed by an "h", to a "k". (This includes all "c"s that are last letter of a word)
11. If a word starts with "kn" change "kn" to“n”
12. Change all double consonants of the same letter to a single consonant. A consonant is any letter that is not one of "a, e, i, o, u." (Example: "apple" becomes "aple"). Keep

applying this rule as necessary (Example: "zzz" becomes "z".)

4.4 Editex Score

Now the transliterated string and the preprocessed string are compared to generate an editex score. The equivalence classes we used were similar to as proposed in the original editex algorithm except for some language specific changes for Hindi. Length of the two strings has to be considered while deciding the threshold score for a match otherwise there can be greater number of mismatches for small strings. So we normalize editex score as
d = [1- {editex(X, Y) / (length(X) + length(Y)}]

The decided threshold for match was 0.86. A score above threshold guarantees equivalence of the two representations. The results are shown in Table-1.

Hindi NE	English NE	Transliteration Output	Editex Score
हिन्दी	Hindi	Hindi	1.0
फ़लस्तीनी	Philistini	Phalastini	0.9
बांगलादेश	Bangladesh	Bangladesh	1.0
झारखण्ड	Jharkhand	Jharakhand	0.894
पश्चिम	Pashchim	Pashchim	1.0
बंगाल	Bengal	Bangal	0.916
भारत	Bharat	Bharat	1.0
क्रिकेट	Cricket	Kriket	0.923
ग्रेग	Greg	Greg	1.0
चैपल	Chappel	Chaipal	0.857
महेंद्र	Mahendra	Mahendr	0.933
राहुल	Rahul	Rahul	1.0
द्रविड	Dravid	Dravid	1.0
छत्तीसगढ	Chattisgarh	Chattisagadh	0.866

Table-1: Hindi named entities with transliteration output and normalized Editex scores

5 Results and Analysis

We have tested our system with a parallel corpus which consisted of both English and Hindi language data. Further we used the web crawler to populate our NE list of both the languages thus embedding the concept of comparable corpus. The results **for English** obtained using parallel corpus are:

Precision: 81.40% and Recall: 81.39%

This corpus carried named entities from the domain of travel, tourism and culture. Further for classifying the results for Hindi we used the definition of named entities as given by Chinchor (Chinchor, 1997) as for entity names organizations (OE), person names (PE) and location names (LE). The results for numeric expressions (monetary values and percentages) and temporal expressions (dates and times) were not considered for results because it is a trivial task to build grammar rules for such entities which appear quite regularly.

We have focused on OE, PE and LE named entities for Hindi so that we can analyze the performance on new and hitherto undiscovered entities which come into existence with the passage of time. This premise provides the real basis for challenging the performance of any NER technique for Indian Languages.

The testing on the corpus of around 1000 sentences revealed the following results **for Hindi**:

- Precision for all named entities (PE+OE+LE): 80.2%
- Recall for PE (person entity names): 47.4%
- Recall for OE (organization entity names): 42.9%
- Recall for LE (location entity names): 74.6%

It is important to observe here that the engine shows good recall for location entity names (LE) which were more abundant in the corpus. Besides this, the corpus had a heterogeneous mix of named entities with tourism-related information not only from India but also from the continents of South America and Antarctica. A good recall percentage for Hindi location entity names is encouraging as the named entities related to South America and Antarctica did not have phonetic similarity with

the native entities available from tourism information from India. This gives good credence to the phonetic matching approach used above. Causes for the comparatively lower recall percentage among person entity names and organization entity names are under further investigation.

6 Conclusions

We have used the phonetic matching technique to match the strings of different languages on the basis of their similar sounding property. As the Phonetic Matcher module is tested for more data, more generic rules can be made to improve its accuracy. The Engine should be improved so that it may recognize phrasal named entities and abbreviations. The engine will work for any language if the phonetic matching rules are written for that language. We can also develop a crawler which will be focused upon a certain domain of interest. Focused crawlers are very important for generating resources for natural language processing. A focused crawler application is an intelligent agent that crawls the web for content related to a specific domain. This kind of crawler could be used in the future for purposes of data collection for a particular domain.

7 Acknowledgements

The authors gratefully acknowledge financial assistance from TDIL, MCIT (Govt. of India).

References

Chinchor, N. 1997. *MUC-7 Named entity task definition*. In Proceedings of the 7th Message Understanding Conference (MUC-7)

Finkel, Jenny Rose, Grenager, Trond and Manning, Christopher. 2005. *Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling*. Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370.

Kim, J. and Woodland, P.C. 2000a. *Rule Based Named Entity Recognition*. Technical Report CUED/ FIN-FENG/TR.385, Cambridge University Engineering Department, 2000.

Malouf, Robert. 2002 *Markov models for language-independent named entity recognition*. In Proceedings of CoNLL-2002 Taipei, Taiwan, pages 591-599.

Levenshtein, V.I. 1966. *Binary codes capable of correcting deletions, insertions, and reversals*. Soviet Physics Doklady 10: 707–710.

Zobel, Justin and Dart, Philip. 1996. *Phonetic string matching: Lessons from information retrieval*. In Proceedings of the Eighteenth ACM SIGIR International Conference on Research and Development in Information Retrieval, Zurich, Switzerland, August 1996, pp. 166-173.

Appendix A: Soundex classes

Code	Letters	Code	Letters
0	aeiouyhw	4	l
1	bpfv	5	mn
2	cgjkqsxz	6	R
3	dt		

Appendix B: Pseudo code for Leveinshtein edit distance:

```
Input: Two strings, X and Y
Output: The minimum edit distance between X and Y

m ← length(X)
n ← length(Y)

for i =0 to m do
dist[i, 0] ← i

for j = 0 to n do
dist[0, j] ← j

for i = 1 to m do
for j = 1 to n do

dist[i, j] =
min{
    dist[i-1, j]+inser_cost,
   dist[i-1, j-1]
   + subst_cost[Xi, Yj],
   dist[i, j-1] + delet_cost
}
end
```

Appendix C: Editex Equivalence Classes:

aeiouy	bp	ckq	dt	lr	mn
gj	fpy	sxz	csz		

Pseudo code for Editex Algorithm

```
Input: Two strings, X and Y
Output: The editex distance
between X and Y
 m = length(X)
 n = length(Y)

editex_dist[0, 0] = 0

for i = 1 to m do

editex_dist[i, 0]
  = editex_dist[i-1, 0]
  + D(Xi-1, Xi)

for j = 0 to n do

editex_dist[0, j]
  = editex_dist[0, j-1]
  + D(Yj-1, Yj)

for i = 1 to m do
for j = 1 to n do

editex_dist[i, j] =
  min { editex_dist[i-1, j]
  + D(Xi-1, Xi),
  editex_dist[i-1, j-1]
  + S(X, Yj),
  editex_dist[i, j-1]
  + D(Yj-1, Yj)

end
```