ICON - 2008

6th International Conference on Natural Language Processing

Details Of The Selected Paper

Title	Sequence labeling approach for English to Tamil Transliteration using Memory-based Learning
Topic	Machine Translation
Abstract	Machine transliteration is an automatic method that converts words/characters in one alphabetical system to corresponding phonetically equivalent words/characters in another alphabetical system. Machine Transliteration has been used extensively to assist machine translation, data mining, cross language information retrieval and more recently in popular web portals, SMS and chat systems. In this paper, we propose a method where transliteration problem is modeled as a sequence labeling problem and proceed to solve this using Memory-based learning. We have applied this technique for transliterating English to Tamil and achieved exact Tamil transliterations for 84.16% of English names. We get an accuracy of 93.33% when we choose from the first five ranked transliterations.
Authors	Vijaya MS Amrita Vishwa Vidyapeetham Shivapratap G Amrita Vishwa Vidyapeetham Dhanalakshmi V Amrita Vishwa Vidyapeetham Ajith VP Amrita Vishwa Vidyapeetham Soman KP Amrita Vishwa Vidyapeetham
Contact	g_shivapratap@ettimadai.amrita.edu
Download	

Close

Sequence labeling approach for English to Tamil Transliteration using Memory-based Learning

Abstract

Machine transliteration is an automatic method that converts words/characters in one alphabetical system to corresponding phonetically equivalent words/characters in another alphabetical system. Machine Transliteration has been used extensively to assist machine translation, data mining, cross language information retrieval and more recently in popular web portals, SMS and chat systems. In this paper, we propose a method where transliteration problem is modeled as a sequence labeling problem and proceed to solve this using Memory-based learning. We have applied this technique for transliterating English to Tamil and achieved exact Tamil transliterations for 84.16% of English names. We get an accuracy of 93.33% when we choose from the first five ranked transliterations.

1 Introduction

As multilingual documents increase rapidly on the Internet, there is an increasing need to bridge the language barrier. One of the most frequent problems dealt by translators is translating named entities (proper nouns) such as person, location, organization names and technical terms. Translation of proper nouns and technical terms is generally recognized as a significant problem in many multi-lingual text and speech processing applications. Hence the transliteration model must aim to preserve the phonetic structure of words as closely as possible. Proper noun processing plays an important role in query translation during Cross-language Information Retrieval, where the query is specified in a language different from that of the retrieved documents.

Machine transliteration is the process of mapping source language phonemes or graphemes into target language approximations. The reverse process is called back transliteration. A source language word can have more than one valid transliteration in the target language. For example, the name "Rama" in English can have the following transliterations in Tamil as ராமா or ரமா. Therefore it becomes necessary to generate all possible transliterations in target language for a given source language name or word.

Various methodologies have been developed for machine transliteration. Arababi (1994) developed a hybrid neural network and knowledge-based system to generate multiple English spellings for Arabic person names. Knight and Graehl (1998) developed a statistical model for back transliteration to transliterate Japanese katakana into English. Al-Onaizan and Knight (2002), Virga and Khudanpur (2003) and Oh and Choi(2000) adopted a source-channel approach incorporating phonetics as an intermediate representation. Li (et al. 2004) introduced direct orthographic mapping through joint source channel for English – Chinese transliteration. Gao (et al. 2004) used Maximum Entropy to English-Chinese transliteration. Kang and Choi (2000) used decision trees model for English-Korean transliteration.

The recent work for sequence alignment on both local classifier-based modeling of complex learning problems (McCallum et al. 2000; Punyakanok and Roth, 2001), as well as global discriminative approaches based on CRFs (Lafferty et al. 2001), SVM (Taskar et al. 2005), and the Perceptron algorithm (Collins 2002) are being adopted for transliteration.

In this paper we present a transliteration model using sequence labeling approach and memory based learning for English to Tamil transliteration.

2 Transliteration as sequence labeling

Transliteration is a process that takes a character string in source language as input and generates a character string in the target language as output. The process can be seen conceptually as two levels of decoding: segmentation of the source string into transliteration units and relating the source language transliteration units with units in the target language by resolving different combinations of alignments and unit mappings. Thus the transliteration problem can be reformulated as a sequence labeling problem from one language alphabet to another.

Consider an English (source language) name X which can be segmented as $x_1x_2..x_i..x_n$ where each x_i is treated as an alphabet in the observation sequence. Let the equivalent Tamil (target language) name be Y. Let Y be segmented as $y_1y_2..y_i..y_n$ where each y_i is treated as a label in the label sequence. Each x_i is now aligned with its phonetically equivalent y_i.

x_1 ———————— y_1
x_2 ———————— y_2
. ————————- .
x_n ————————y_n

This is a very important step in our transliteration process. Proper alignment of phonetically equivalent segments is required to generate an efficient transliteration model.

The valid target language alphabet (yi) for a source language alphabet (xi) in the given source language input word depends on the following factors

- source language alphabet (x_i)
- context alphabets (x_{i-2}, x_{i-1}, x_{i+1}, x_{i+2}) surrounding source language alphabet (x_i)
- target language alphabet (y_i)
- context alphabets (y_{i-2}, y_{i-1}, y_{i+1}, y_{i+2}) surrounding target language alphabet (y_i)

This context information is used to train the model using memory based learning. The trained transliteration model then predicts a valid target language word (label sequence) for new source language word (observation sequence). Thus the transliteration problem in this context becomes a multi-class classification problem.

2.1 Challenges in English-Tamil Transliteration

Following are the challenges in English to Tamil transliteration:

- Complexity arises while transliterating vowels in English, since these vowels may correspond to long vowels or short vowels in Tamil. For example, the name “Rama” in English can have Tamil transliterations as both ராமா (rAmA) and ரமா (ramA).
- Some consonants like ‘r’, ‘l’, and ‘n’, has multiple transliterations in Tamil, namely - ‘r’ (ர, ற), ‘l’ (ல , ள , ழ) , ‘n’ (ண, ன , ந).
- Consonants like‘d’ ‘t’ may sound in Tamil as ‘த’ and ‘ட’. For example in “Nadarajan” – ‘da’ is transliterated as ‘ட’ and in “Dasarathan” – da is transliterated as ‘த’ .

We have designed our transliteration system by taking these factors into account.

3 Memory-based learning

Memory-based learning is a form of supervised learning based on similarity-based reasoning. Examples are represented as a vector of feature values with an associated class label. During training, a set of examples (the training set) is presented in an incremental fashion to the classifier, and added to memory. During testing, a set of previously unseen feature-value patterns (the test set) is presented to the system. For each test pattern, its distance to all examples in memory is computed, and the class label of the least distant instance(s) is used as the predicted class label for the test pattern. Performance of a memory-based system is measured as classification accuracy on the test set. It depends on the distance metric (or similarity metric) used. The most straightforward distance metric is

$$\Delta(X,Y) = \sum_{i=1}^{n} d_i(x_i, y_i)$$

where X and Y are the patterns to be compared, and $d_i(x_i , y_i)$ is the distance between the values of the i-th feature in a pattern with n features. Distance between two values is measured using an overlap metric, for symbolic features as $d_i (x_i, y_i) = 0$ if $x_i = y_i$ else $d_i (x_i , y_i) = 1$

If the pattern is ambiguous, the distribution of pattern over the different labels is kept, and the most frequently occurring label is selected. When this distance metric is used, all features describing an example are interpreted as being equally important in solving the classification problem. During transliteration, the source language word/alphabet to be assigned a label (target language alphabet) is obviously more relevant than any of the source language alphabets in its context. Therefore a weight is associated to each feature. Here the information gain, a number expressing the average amount of reduction of training set information entropy is used as weights and the distance measure is

$$\Delta(X,Y) = \sum_{i=1}^{n} G(f_i) d_i (x_i, y_i)$$

This is referred as IB1-IG algorithm.

3.1 Memory-based learning using IGTree

In typical NLP learning tasks where the focus is on discrete data, large number of examples and many attributes of differing relevance, the classification speed is a critical issue. Memory-based learning is an expensive algorithm in terms of memory and retrieval speed since for each test case, all feature values must be compared to the corresponding feature values of all training instances. Without optimization, it has an asymptotic retrieval complexity of *O(NF)*,where N is the number of items in memory, and F the number of features. These constraints demand non trivial data structures and speedup optimization for the core k-NN classifier. This resulted in IGTree decision tree. IGTree (Walter Daelemans, Jakub Zavrel, 1996) compresses the typical flat file organization found in k-NN implementations into a decision tree structure. IGTree combines two algorithms: one for compressing a case base into a tree, and one for retrieving classification information from these trees.

During the construction of IGTree decision trees, cases (instances) are stored as paths of connected nodes. All nodes contain a test and a class label. Nodes are connected via arcs denoting the outcomes for the test. Information gain is used to determine the order in which features are used as tests in the tree. The tree is constructed such a way that the maximal depth of the tree is equal to the number of features. At each level of the tree, all nodes have the same test feature. The reasoning behind this compression is due to the fact that the computation of feature relevance points to one feature is the most important in classification. Search can be restricted to matching a test case to those stored cases that have the same feature value at that feature. Besides restricting search to those memory cases that match only on this feature, the case memory can be optimized by further restricting search to the second most important feature, followed by the third most important feature, etc. A considerable compression is obtained as similar cases share partial paths.

The tree is compressed even more by restricting the paths to those input feature values that disambiguate the classification from all other cases in the training set. The feature values that do not contribute to the disambiguation of the case classification are not stored in the tree. Finally, the leaf nodes contain the unique class label corresponding to a path in the tree. Given the path, non-terminal nodes contain information about the most probable or default classification. A final compression is obtained by pruning the derived tree.

The classification of a new instance involves traversing the tree by matching all feature values of the test case with arcs in the order of the overall feature information gain. In case of a match, the leaf node at the end of the traversal provides the class label. If no match is found, the last matching non terminal node provides the default class label.

4 Transliteration using Memory-based learning

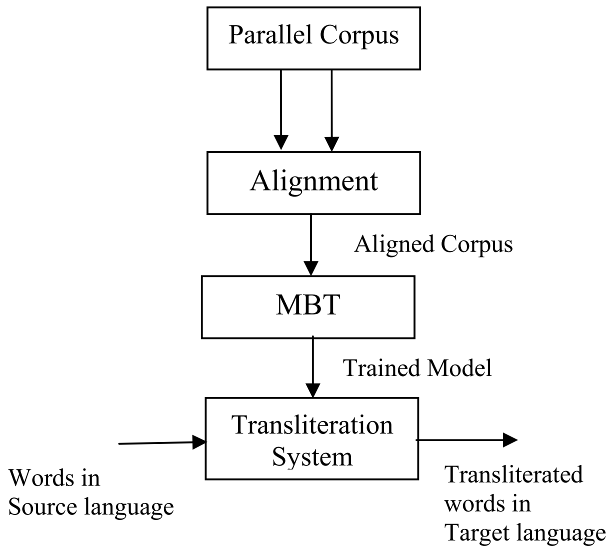
The transliteration model based on Memory-based learning builds three data structures, a lexicon, a case base for known source language alphabets and a case base for unknown alphabets. These are automatically extracted from the given aligned corpus. Case bases are indexed using IGTree.

Lexicon: A lexicon is built by computing the number of times each source language alphabet occurs with each class label. The lexicon is referred during prediction of target language alphabets for a given source language alphabet. Depending on whether or not they can be found in the lexicon, a case representation is constructed for the source language alphabets and they are retrieved either from known case base or the unknown case base.

Case base (known): A windowing approach is used to represent the transliteration task as a classification problem. For each alphabet, the context information is stored in the case base. A case consists of information about the focus source and target language alphabets, and their left and right contexts.

Case base (unknown): If a source language alphabet is not present in the lexicon, its ambiguous target language alphabet cannot be retrieved. In such case, a class label can be guessed only on the basis of the form or the context of the alphabet.

During transliteration, each alphabet in the novel source language word is looked up in the lexicon. If it is found, its lexical representation is retrieved and its context is determined. The resulting pattern is looked up in the known alphabet case base. If a source language alphabet is not present in the lexicon, its lexical representation is computed on the basis of its form, its context is determined, and the resulting pattern is looked up in the unknown alphabet case base. In each case, output is a best guess of the target language alphabet (class label) for the source language alphabet in its current context.



The transliteration process consists of three phases: preprocessing phase, training phase using MBT[1] and transliteration phase. The names in source language and target language are aligned and converted into the format required by MBT during preprocessing stage. MBT is a memory-based tagger-generator and tagger. It is used for training and transliteration.

[1] http://ilk.uvt.nl/mbt/

4.1 Preprocessing Phase

During the preprocessing phase, the source language names are segmented and aligned with the corresponding segmented target language names. We have created a parallel corpus consisting of 30,000 Indian person names and 30,000 Indian place names. This corpus is used in training the model. Preprocessing involves the following steps:

1. **Romanization**: Tamil names are romanized using mapping rules and English names are converted into lower case.

English	**Tamil**	**Romanized Tamil**
shiva	ஷிவா	shivA
Priya	பிரியா	Piriya
karthik	கார்த்திக்	kArTTik
n e h r u </s> w E ^ r u </s>	கௌரி	Kauri

2. **Segmentation**: English Names are segmented based on vowels, consonents, digraphs and trigraphs like sh, bh, ksh, th, ch, ng, nj etc. Similarly romanized Tamil names are segmented based on vowels, consonents digraphs and trigraphs like sh, TT, kk, ss, pp, ngk, njs etc. into transliteration units.

English	**Romanized Tamil**
sh i v a	sh i v A
p r i y a	p i r i y a
k a r th i k	k A r TT i k
n e h r u	w E ^ r u

3. **Alignment**: The corresponding transliteration units in English and Tamil words are aligned if the number of units in both English and Tamil words are equal. For example:
   sh i v a (4 units)
   sh i v A (4 units)

   A mismatch in the number of transliteration units in English and Tamil will result in misalignment. This is resolved by combining two or three adjacent transliteration units at the required position in Tamil names or by inserting empty symbol ε or ^ using rules without losing their phonetic structure. For example:

```
p   r   i   y   a       ( 5 units )
p   i   r   i   y   a   ( 6 units )
```

is aligned as

```
 p | r  | i | y | a       ( 5 units )
pi | r  | i |y  |a        ( 5 units )
```

Similar alignment is to be done for plosives க, ச, ட, த, ப, ற.

An example of using the empty symbol for resolving misalignment is:

```
n   e   h   r   u    ( 5 units )
n   e   r   u        ( 4 units )
```

is aligned as

```
n  |e   |h  |r  |u    ( 5 units )
n  |e   |^  |r  |u    ( 5 units )
```

Thus in our model, we consider English unigrams, n-grams, empty alphabet, Tamil roman characters and Tamil n-grams. Here the target language n-grams are considered as labels in sequence labeling and multi classification.

4.2 Training Phase

The aligned source language (English) and target language (Tamil) names are given as input sequence and label sequence respectively for training in the format as required by MBT. The context information required for training is defined with a window size of 5. The core alphabet is in the third position. A lexicon and two case bases are built by MBT in the training phase. MBT uses two algorithms for Memory based learning. The IGTree algorithm is used for known case base and IB1-IG algorithm is used for the unknown case base. The trained model is used for transliterating English words into Tamil words.

4.3 Transliteration Phase

The list of English words which need to be transliterated are segmented and converted into the MBT Tool format and transliterated using the trained model. MBT uses IB1-IG and IGTree decision tree for predicting class labels for each alphabet in English word and the sequence of predicted class labels form a transliterated Tamil word.

5 Results

Our model produced the exact Tamil transliteration for English words with an accuracy of 84.16%. We attribute the misclassifications to the delicate characteristics of the Tamil language while transliterating vowels and the consonants 'r','l','n','d','t' as discussed earlier. We then made several possible transliterations based on the output present in the ambiguity file for the vowels and the above mentioned consonants. The accuracy is increased when the number of possible transliterations is increased. The transliteration accuracy of our model is shown in Table 1.

Output	**Accuracy**
Top 1	84.16%
Top 2	85.96%
Top 3	88.07%
Top 4	90.05%
Top 5	93.33%

Table. 1: Transliteration Accuracy

6 Conclusion

We have demonstrated a transliteration model for English to Tamil transliteration using Memory based learning by reformulating the transliteration problem as sequence labeling and multi classification. We have prepared a parallel corpus of 30,000 person names and 30,000 thousand place names and used it to train our transliteration model. We also tested the accuracy of our model with 1000 English names that were out of corpus. Our model produces an exact transliteration in Tamil from English words with an accuracy of 84.16%. The accuracy can be increased by generating possible transliterations using the ambiguity file.

References

Al-Onaizan Y, Knight K, 2002. *Machine translations of names in Arabic Text* Proceedings of the ACL conference workshop on computational approaches to Semitic languages.

Arababi Mansur, Scott M. Fischthal, Vincent C. Cheng, and Elizabeth bar. 1994. *Algorithms for Arabic name transliteration*. IBM Journal of Research and Development.

Collins M, 2002, *Discriminative Training for Hidden Markov Models: theory and Experiments with perceptron algorithms*, In proceedings of EMNLP

Dmitry Zelenko, Chinatsu Aone, 2006, *Discriminative methods for Transliteration*, Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing(EMNLP2006), pages612–617

Knight Kevin and Graehl Jonathan, 1997, *Machine transliteration*. In proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, pp 128-135

Punyakonak V, Roth D, 2001, *The use of Classifiers in Sequential Inference*, Proceedings of the Conference on Advances in Neural Information Processing Systems.

Surya Ganesh, Sree Harsha, Prasad Pingali, Vasudeva Varma, 2008, *Statistical transliteration for Cross Langauge Information Retrieval using HMM alignment and CRF*, The Second International Workshop on Cross Lingual Information Access-Addressing the Informaion Need of Multilingual Socoeties.

Sathiya Keerthi S, Sundararajan S, 2007, *CRF versus SVM-Struct for Sequence Labeling*, Yahoo Research technical report.

Taskar B, Lacoste-Julien S, and Klein D. 2005, A Discriminative Matching Approach to Word Alignment. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Walter Daelemans, Jakub Zavrel, 1996, MBT: *A Memory-Based Part of Speech Tagger-Generator,* proceedings of WVLC

Zhang Min, LI Haizhou SU Jian, *Direct Orthographical Mapping for Machine Transliteration, 2004,* Proceedings of the 20th international conference on Computational Linguistics.