Substring-based Transliteration with Conditional Random Fields

Sravana Reddy and **Sonjia Waxmonsky**
Department of Computer Science
The University of Chicago
Chicago, IL 60637
{sravana, wax}@cs.uchicago.edu

Abstract

Motivated by phrase-based translation research, we present a transliteration system where characters are grouped into substrings to be mapped atomically into the target language. We show how this substring representation can be incorporated into a Conditional Random Field model that uses local context and phonemic information.

1 Introduction

We present a transliteration system that is motivated by research in phrase-based machine translation. In particular, we borrow the concept of *phrases*, which are groups of words that are translated as a unit. These phrases correspond to multi-character *substrings* in our transliteration task. That is, source and target language strings are treated not as sequences of characters but as sequences of non-overlapping substrings.

We model transliteration as a *sequential labeling task* where substring tokens in the source language are labeled with tokens in the target language. This is done using Conditional Random Fields (CRFs), which are undirected graphical models that maximize the posterior probabilities of the label sequence given the input sequence. We use as features both *local contexts* and *phonemic information* acquired from an English pronunciation dictionary.

2 The Transliteration Process

Our transliteration system has the following steps:

1. **Pre-processing** of the target language.
2. **Substring alphabet generation** for both the source and target. This step also generates training data for the CRFs in Step 3 and 4.
3. **CRF training** on aligned data from Step 2.
4. **Substring segmentation and transliteration** of source language input.

Our training and test data consists of three sets – English to Hindi, English to Kannada, and English to Tamil (Kumaran and Kellner, 2007) – from the NEWS 2009 Machine Transliteration Shared Task (Li et al., 2009).

2.1 Step 1: Pre-Processing

The written words of Hindi, Tamil, and Kannada correspond almost perfectly to their phonological forms, with each character mapping to a phoneme. The only exception to this arises from the implicit vowel (which may be a schwa /ə/ or a central vowel /ɐ/) that is inserted after consonants that are not followed by the *halanta* or 'killer stroke'. Hence, any mappings of an English vowel to a target language schwa will not be reflected in the alignment of the named entity pair.

To minimize misalignments of target language strings with the English strings during training, we convert the Indic abugida strings to an internal phonemic representation. The conversion maps each unicode character to its corresponding phonemic character and inserts a single symbol (representing the schwa/central vowel) after all consonants that are not followed by the *halanta*.

These phoneme sequences are used as the internal representation of Indic character strings for all later steps in our system. Once transliteration is complete, the phonemic symbols are converted back to unicode by reversing the above process.

2.2 Step 2: Substring alphabet generation

Our decision to use substrings in the transliteration task is motivated by the differences in orthography and phonology between the target and source languages, which prevent trivial one-to-one character level alignment. We first discuss the cause of the poor character alignment between English and the

Indic languages, and then describe how we transform the input into substring representation.

English uses several digraphs for phonemes that are represented by single characters in Indic scripts, which are either part of standard orthographic convention (*oo*, *ch*, etc.), or necessitated by the lack of a single phoneme that approximates an Indic one (as in the case of aspirated consonants). Conversely, English sometimes uses a single character for a biphone (such as *x* for /ks/, or *u* for /ju/ as in *museum*), which is represented by two characters in the target languages. In certain cases, a digraph in English is transliterated to a digraph in the target, as a result of metathesis (*le* → /əl/, in words like *temple*). Further, all three target languages often insert vowels between English consonant clusters; for instance, Hindi inserts a schwa between *s* and *p* in 'transport', transliterated as ʈrɑnsəporʈ (ट्रांसपोर्ट).

To handle these cases, we borrow the concept of *phrases* from machine translation (Och and Ney, 2004), where groups of words are translated as a unit. In the case of transliteration, the 'phrases' are commonly occurring *substrings* – sequences of characters – in one language that map to a character or a substring in the other. We use the term 'substrings' after a previous work (Sherif and Kondrak, 2007) that employs it in a noisy channel transliteration system. Zhao et al. (2007) also use substrings (which they call 'blocks') in a bi-stream HMM.

We bootstrap the induction of substrings by aligning all named entity pairs in the training data, using the GIZA++ toolkit (Och and Ney, 2003). The toolkit performs *unidirectional one-to-many* alignments, meaning that a single symbol in its source string can be aligned to *at most one* symbol in its target. In order to induce many-to-many alignments, GIZA++ is run on the data in both directions (source language to target language and target language to source), and the bidirectional alignment of a named entity pair is taken to be the union of the alignments in each direction. Any inserted characters (maps within the alignment where the source or target character is *null*) are combined with the preceding character within the string. For example, the initial bidirectional alignment of *shivlal* → ʃivəlɑl (शिवलाल) contains the maps [*sh* → ʃ, *i* → i, *v* → v, *null* → ə, *l* → l, *a* → ɑ, and *l* → l]. The *null* → ə map is combined with the preceding map to give *v* → və, and hence a one-to-one alignment.

Multicharacter units formed by bidirectional alignments are added to source and target alphabets. The above example would add the substrings 'sh' to the source alphabet, and və to the target. Very low frequency substrings in both languages are removed, giving the final substring alphabets of single and multicharacter tokens. These alphabets (summarized in Table 1) are used as the token set for the CRF in Step 3.

We now transform our training data into a substring-based representation. The original named entity pairs are replaced by their bidirectional one-to-one alignments described earlier. For example, the ⟨*s h i v l a l*⟩ → ⟨ʃ i v ə l ɑ l⟩ training pair is replaced by ⟨*sh i v l a l*⟩ → ⟨ʃ i və l ɑ l⟩. A few (less than 3%) of the pairs are *not* aligned one-to-one, since their bidirectional alignments contain low-frequency substrings that have not been included in the alphabet.[1] These pairs are removed from the training data, since only one-to-one alignments can be handled by the CRF.

2.3 Step 3: CRF transliteration

With the transformed training data in hand, we can now train a CRF sequential model that uses substrings rather than characters as the basic token unit. The CRF algorithm is chosen for its ability to handle non-independent features of the source language input sequence. We use the open-source CRF++ software package (Kudo, 2005).

Ganesh et al. (2008) also apply a CRF to the transliteration task (Hindi to English) but with different alignment methods than those presented here. In particular, multicharacter substrings are only used as tokens on the target (English) side, and a null token is used to account for deletion.

We train our CRF using unigram, bigram, and trigram features over the source substrings, as well as pronunciation information described in §2.3.1. Table 2 describes these feature sets.

2.3.1 Phonetic information

Since the CRF model allows us to incorporate non-independent features, we add pronunciation data as a token-level feature. Doing so allows the CRF to use phonetic information for local decision-making. Word pronunciations were obtained from

[1]Note that if we did not filter out any of the substrings, every pair would be aligned one-to-one.

Target Language	Source		Target	
	# of Tokens	Longest Token	# of Tokens	Longest Token
Hindi	196	*augh, ough*	141	ɑjə (आय), ksə (क्स)
Kannada	197	*aine*	137	ɑjə, mjɑ
Tamil	179	*cque*	117	mij, ɑjə

Table 1: Overview of the substring alphabets generated in Step 2.

Feature Set	Description
U	Unigram: s_{-1}, s_0, and s_1
B	Bigram: s_{-1}+s_0
T	Trigram: s_{-2}+s_{-1}+s_0, s_{-1}+s_0+s_1 and s_0+s_1+s_2
P	Phoneme assigned to s_0 from dictionary lookup

Table 2: Feature sets used for CRF in Step 3. s_i is the substring relative to the current substring s_0.

the CMU Pronouncing Dictionary[2]. Just over a third of the English named entities have pronunciation information available for some or all the constituent words.

The CMU dictionary provides a sequence of phoneme symbols for an English word. We include these phonemes as CRF features if and only if a one-to-one correspondence exists between phonemes and substring tokens. For example, the English word *simon* has the segmentation ⟨*s i m o n*⟩ and pronunciation ⟨*S AY M AH N*⟩, both of length five. Additionally, a check is done to ensure that vowel phonemes do not align with consonant characters and vice-versa.

2.4 Step 4: Substring segmentation

In order to apply our trained model to unseen data, we must segment named entities into non-overlapping substrings that correspond to tokens in the source alphabet generated in Step 2. For instance, we need to convert the four character *desh* to the three token sequence ⟨*d e sh*⟩.

This is a non-trivial task. We must allow for the fact that substrings are not inserted *every* time the component character sequence appears. For instance, in our English/Hindi training set, the bigram *ti* always reduces to a single substring token when it occurs in the *-tion* suffix, but does not reduce in any other contexts (like *martini*). There are also cases where more than one non-trivial segmentation is possible. For example, two possible segmentations of *desh* are ⟨*d es h*⟩ and ⟨*d e sh*⟩, with the latter being the one that best corresponds to the three-character Hindi d̪eʃ (देश).

One solution is to greedily choose the most likely multi-character substring – in the example cited, we can choose ⟨*d e sh*⟩ because *sh* reduces more frequently than *es*. However, this creates the problem in cases where no reduction should occur, as with the *ti* in *martini*. Since contextual information is necessary to determine the correct substring segmentation, we model segmentation with a CRF, using a combination of character unigram, bigram, and trigram features.

We use an approach motivated by the Inside/Outside representation of NP-chunking which treats segmentation as a tagging process over words (Ramshaw and Marcus, 1995). As in NP-chunking, our goal is to identify non-overlapping, non-recursive segments in our input sequence. Our tagset is {**I**, **O**, **B**} where **I** indicates that a character is inside a substring, **O** indicates a character is outside a substring, and **B** marks a right boundary.

After the test data has been segmented into its substring representation, it can be passed as input to the CRF model trained in Step 3 to produce our final transliteration output.

3 Results

We first report our results on the development data provided by the NEWS task, for different feature sets and segmentation methods. We then present the performance of our system on the test data.[3]

3.1 Development Data

Table 3 shows the results across feature sets. Noting that the trigram feature **T** provides a sizable improvement, we compare results from **U+B+T+P** and **U+B+P** feature sets. Of the improved cases, 75-84% are a single vowel-to-vowel

[2]The CMU Pronouncing Dictionary (v0.7a). Available at http://www.speech.cs.cmu.edu/cgi-bin/cmudict

[3]For the development runs, we use the *training* set for training, and the *development* for testing. For the final test runs, we use both the *training* and *development* sets for training, and the *test* set for evaluation.

Language	Feature Set	ACC	F-Score
Hindi	U+P	24.6	86.2
	U+B+P	26.2	86.5
	U+B+T+P	**34.5**	**88.6**
	U+B+T	34.2	88.3
Tamil	U+P	26.7	87.8
	U+B+P	27.6	88.0
	U+B+T+P	**34.9**	**89.8**
	U+B+T	33.1	89.7
Kannada	U+P	22.5	86.0
	U+B+P	22.6	86.2
	U+B+T+P	**28.7**	**88.0**
	U+B+T	27.5	87.9

Table 3: Accuracy (ACC) and F-score results (in %) for CRF model on the ***development data***.

Language	Feature Set	ACC	F-Score
Hindi	**U+B+T+P**	**34.4**	**90.2**
	U+B+T	33.6	89.5
Tamil	**U+B+T+P**	**29.1**	**91.1**
	U+B+T	25.5	90.6
Kannada	**U+B+T+P**	**27.2**	**89.8**
	U+B+T	23.4	89.2

Table 4: Results on development data, restricted to NEs where **P** is included as a feature.

change, with the majority of the changes involving a schwa/central vowel.

We see small gains from using the phonetic feature in both accuracy and F-Score. We further examine only those named entities where dictionary information is applied, and as expected, this subset shows greater improvement (Table 4).

Table 5 compares our the Inside/Outside tagging approach with a greedy approach described earlier. The greedy approach only inserts a multi-character substring when that substring reduces more than 50% of the time in the overall training corpus. Since the Greedy method uses no local contextual information, results are significantly lower given the same feature set.

Language	Segmentation	ACC	F-Score
Hindi	I-O-B	**34.5**	**88.6**
	Greedy	30.3	86.7
Tamil	I-O-B	**34.9**	**89.8**
	Greedy	28.2	87.5
Kannada	I-O-B	**28.7**	**88.0**
	Greedy	25.0	86.7

Table 5: Comparison of segmentation methods on development data, using the **U+B+T+P** feature set.

3.2 Test Data

Our model produces 10 candidates for each named entity in the test data, ranked by the probability that the model assigns the candidate. We filter out candidates below the rank of 5 whose scores are less than 0.5 lower than that of the highest ranking candidate. Table 6 shows our results on the test data, using a CRF trained on the training and development data, with the feature set **U+B+T+P**.

	Hindi	Kannada	Tamil
Accuracy	41.8	36.3	43.5
F-Score	87.9	87.0	90.2
MRR	54.6	48.2	57.2
MAP_{ref}	41.2	35.5	43.0
MAP_{10}	18.3	16.4	19.5
MAP_{sys}	24.0	21.8	26.5

Table 6: Final results on the ***test data*** (in %).

References

Surya Ganesh, Sree Harsh, Prasad Pingali, and Vasudeva Varma. 2008. Statistical transliteration for cross language information retrieval using HMM alignment model and CRF. In *Proceedings of the 2nd Workshop on Cross Lingual Information Access*.

Taku Kudo. 2005. CRF++: Yet another CRF toolkit. Available at http://chasen.org/ taku/software/crf++/.

A. Kumaran and Tobias Kellner. 2007. A generic framework for machine transliteration. In *Proceedings of SIGIR-07*.

Haizhou Li, A Kumaran, Min Zhang, and Vladimir Pervouchine. 2009. Whitepaper of NEWS 2009 machine transliteration shared task. In *Proceedings of ACL-IJCNLP 2009 Named Entities Workshop (NEWS 2009)*.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Franz Josef Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of WVLC-3*.

Tarek Sherif and Grzegorz Kondrak. 2007. Substring-based transliteration. In *Proceedings of ACL-07*.

Bing Zhao, Nguyen Bach, Ian Lane, and Stephan Vogel. 2007. A log-linear block transliteration model based on bi-stream HMMs. In *Proceedings of NAACL HLT 2007*.