Comparing Two Techniques for Learning Transliteration Models Using a Parallel Corpus

Hassan Sajjad **Nadir Durrani** **Helmut Schmid** **Alexander Fraser**
Institute for Natural Language Processing
University of Stuttgart
{sajjad,durrani,schmid,fraser}@ims.uni-stuttgart.de

Abstract

We compare the use of an unsupervised transliteration mining method and a rule-based method to automatically extract lists of transliteration word pairs from a parallel corpus of Hindi/Urdu. We build joint source channel models on the automatically aligned orthographic transliteration units of the automatically extracted lists of transliteration pairs resulting in two transliteration systems. We compare our systems with three transliteration systems available on the web, and show that our systems have better performance. We perform an extensive analysis of the results of using both methods and show evidence that the unsupervised transliteration mining method is superior for applications requiring high recall transliteration lists, while the rule-based method is useful for obtaining high precision lists.

1 Introduction

Urdu and Hindi are closely related languages which have a similar phonological, semantic and syntactic structure. Hindi is derived from Sanskrit and Urdu is a mixture of Persian, Arabic, Turkish and Sanskrit. Both share closed class vocabulary which they inherit from Sanskrit. They differ however in the open class vocabulary and in the writing script used. Hindi is written in Devanagari script and borrows most of the open class vocabulary from Sanskrit. Urdu is written in Perso-Arabic script and borrows most of the open class vocabulary from Persian, Arabic, Turkish and Sanskrit. Both languages have lived together for centuries and now share a large part of their vocabulary with each other. In an initial study on a small parallel corpus, we found that both languages share approximately 82% (tokens) and 62% (types) of the vocabulary. Transliterating overlapping words will help to bridge the scripting gap between Hindi and Urdu. The remaining words must be converted into the other language with a bilingual dictionary which is beyond the scope of this work.

In this paper, the term *transliteration pair* refers to a word pair where the words are transliterations of each other and the term *transliteration unit* refers to a character pair where the characters are transliterations of each other. We are interested in building joint source channel models for transliteration. Because we do not have a list of transliteration pairs to use as training data in building such a transliteration model, we use two methods to extract the list of transliteration pairs from a parallel corpus of Hindi/Urdu. The first method uses the transliteration mining algorithm of Sajjad et al. (2011) to automatically extract transliteration pairs. This approach does not use any language specific knowledge. The second method uses handcrafted transliteration rules specific to the mapping between Hindi and Urdu to extract transliteration pairs. We automatically align the two lists of extracted transliteration pairs at the character level and learn two transliteration models. We compare the results with three other transliteration systems. Both of our transliteration systems perform better than the other systems.

The 1-best output of the transliteration system built on the list extracted using the rule-based method is better than the 1-best output of the system built on the automatically extracted list. The rule-based extraction method is focused on obtaining a high precision list as compared to the automatic method which obtains a higher recall list. The 10-best and 20-best output of the transliteration system built on the automatically extracted list is better than the N-best outputs of the system built on the list extracted using the rule-based method. The wide coverage of transliteration units in the automatically extracted list helps the

transliteration system to produce difficult transliterations which are hard to learn using the rule-based list.

The transliteration task between Hindi and Urdu is non-trivial. The missing short vowels in the writing of Urdu and a missing short vowel in the writing of Hindi are a particular problem, and we identify other areas of difficulty. We provide a detailed error analysis to account for the complexities in Hindi to Urdu transliteration motivated by linguistic phenomena.

The paper is organized as follows. Previous work on transliteration is summarized in Section 2. The two methods used to extract lists of transliteration pairs are described in Section 3. The joint probability model for transliteration is explained in Section 4. The evaluation and the results in comparison with three other transliteration systems are presented in Section 5. A detailed discussion and error analysis is presented in Section 6. Section 7 concludes.

2 Previous Work

Transliteration can be done with phoneme-based or grapheme-based models. Knight and Graehl (1998), Stalls and Knight (1998), Al-Onaizan and Knight (2002) and Pervouchine et al. (2009) use the phoneme-based approach for transliteration. Kashani et al. (2007) and Al-Onaizan and Knight (2002) use a grapheme-based model to transliterate from Arabic into English. Al-Onaizan and Knight (2002) compare a grapheme-based approach, a phoneme-based approach and a linear combination of both for transliteration. They build a conditional probability model. The grapheme-based model performs better than the phoneme-based model and the hybrid model. This motivates our use of grapheme-based models.

In this paper, we use a grapheme-based approach for transliteration from Hindi to Urdu. The phoneme-based approach would involve the conversion of Hindi and Urdu text into a phonemic representation which is not a trivial task as the short vowel 'a' is not written in Hindi text and no short vowels are written in Urdu text. The difficulty of this additional step would be likely to lead to additional errors.

Malik et al. (2008) and Malik et al. (2009) work on transliteration from Hindi to Urdu and Urdu to Hindi respectively. They use the rules of SAMPA (Speech Assessment Methods Phonetic Alphabets) and X-SAMPA[1] to develop a phoneme-based mapping scheme between Urdu and Hindi (J C. Wells, 1995).

Hindi	SAMPA	Urdu			
ज़	z	ز	ض	ظ	ذ
स	s	س	ص	ث	
ह	h	ہ	ھ	ح	
त	t_d	ت	ط		

Table 1: Ambiguous Hindi characters (characters which can transliterate to many different Urdu characters)

Malik et al. (2008) reported an accuracy of 97.9% for transliterating Hindi to Urdu. However, this number is not comparable to ours. Some Hindi characters can be ambiguously transliterated to several Urdu characters (see Table 1). Malik et al. (2008) do not deal with these ambiguous characters and count any occurrence of an ambiguous character as a correct transliteration in all scenarios. We discuss this further in Section 6.

In the previous work, a transliteration system is built on transliteration units learned either automatically from a list of transliteration pairs (Li et al., 2004), (Pervouchine et al., 2009) or using a heuristic-based method (Ekbal et al., 2006). We do not have a list of transliteration pairs for the training of our Hindi to Urdu transliteration system. Therefore we use two methods to extract transliteration pairs from parallel data of Hindi/Urdu. In the first approach, we use the transliteration mining algorithm proposed by Sajjad et al. (2011) to extract transliteration pairs. This method does not use any language dependent information. In the second approach, we use a rule-based method to extract transliteration pairs. Both processes are imperfect, meaning that there is noise in the extracted list of transliteration pairs. We build a joint source channel model as described by Li et al. (2004) and Ekbal et al. (2006) on the extracted list of transliteration pairs. The following sections describe the two mining approaches and the model in detail.

3 Extraction of Transliteration Pairs

We automatically word-align the parallel corpus and extract a word list, later referred to as *"list of word pairs"* (see Section 5, for details on training data). We use two methods to extract transliteration pairs from the list of word pairs. In the first

[1]SAMPA and XSAMPA are used to represent the IPA symbols using 7-bit printable ASCII characters.

approach, we automatically extract transliteration pairs using the transliteration mining algorithm as proposed in Sajjad et al. (2011). We align the transliteration pairs at character level using a character aligner. In the second approach, we use an edit distance metric and handcrafted equivalence rules to extract transliteration pairs from a parallel corpus. We align the list of transliteration pairs at character level using the edit distance metric. The transliteration system is then trained on these character aligned transliteration pairs which is described in Section 5. The following subsections describe the extraction methods in detail.

3.1 Automatic Extraction of Transliteration Pairs

In this section, we review the transliteration mining approach described by Sajjad et al. (2011) to automatically extract the transliteration pairs from the list of word pairs. The approach consists of two algorithms, Algorithm 1, which performs an iterative filtering of the word pair list, and Algorithm 2, which determines when Algorithm 1 should be stopped. The details of this process follow.

Algorithm 1 is based on an iterative process. In each iteration, it first builds a joint transliteration model using g2p (grapheme-to-phoneme converter (Bisani and Ney, 2008)) on the current list of word pairs. It then filters out 5% of the word pairs which are least likely to be transliterations according to their normalized joint probability, resulting in a reduced word pair list, after which the next iteration begins. In each iteration the word pair list is reduced by 5%.

Algorithm 2 is used to select the optimal stopping iteration for Algorithm 1. Algorithm 2 is an extension of Algorithm 1. It divides the original list of word pairs into two halves which are used as training and held-out data. The division is done using a special splitting method which keeps the morphologically related word pairs from the list of word pairs either in the training data or in the held-out data. It builds a joint sequence model on the training data (approximately half of the list of word pairs) and filters out those 5% word pairs which are least likely to be transliteration pairs. Then it builds a transliteration system using the Moses toolkit (Koehn et al., 2003) on the filtered data and tests it on the source side of the held-out data. It repeats this process for 100 iterations. The iteration which best predicts the held-out data is selected as the stopping iteration for the transliteration mining algorithm.

We first ran Algorithm 2 on the list of word pairs for 100 iterations. It returned the 45th iteration as the best stopping iteration for Algorithm 1. Then we ran Algorithm 1 for 45 iterations and obtained a list of 2245 transliteration pairs. Due to data sparsity, there were two Hindi characters which were missing in the extracted list of transliteration pairs. We could either add complete word examples or just transliteration units of the missing Hindi characters to the list of transliteration pairs. Adding examples will provide context information which may bias the results of the evaluation. Thus we added only the two missing 1-to-1 transliteration units to the list of transliteration pairs.

We align the list of transliteration pairs at the character level using a character aligner[2]. The aligner uses the Forward-Backward algorithm to learn the character alignments between the transliteration pairs. It allows only 0 or 1 character on either side of the transliteration unit. So, a source character can align either to a target character or to $\emptyset$ and a target character can align either to a source character or to $\emptyset$. We get three kinds of alignments of Hindi characters to Urdu characters i.e. $\emptyset \rightarrow 1$, $1 \rightarrow \emptyset$ and $1 \rightarrow 1$. We modify the $\emptyset \rightarrow 1$ alignments by merging the Urdu character with the left neighboring aligned pair. If it is the left-most character, then it is merged with the right neighboring aligned character pair. Table 2 shows the alignment of Hindi characters with Urdu characters before and after the merging of unaligned Urdu characters.

a)	Hindi	$\emptyset$	b	c	$\emptyset$	e	f
	Urdu	A	X	C	D	$\emptyset$	F
b)	Hindi		b	c		e	f
	Urdu		AX	CD		$\emptyset$	F

Table 2: Hindi-Urdu alignment pairs for transliteration where a) shows initial alignment with NULL alignments and b) shows final alignments after merging of NULL alignments

3.2 Rule-based Extraction of Transliteration Pairs

As an alternative to automatic extraction of transliteration pairs, we use our own knowledge

[2]We were unable to get character alignments from g2p. We use a separate character aligner to align the list of transliteration pairs at the character level.

of the Hindi and Urdu scripts to make the initial transliteration units. The rules are further extended by looking into available Hindi-Urdu transliteration systems and other resources (Gupta, 2004; Malik et al., 2008; Jawaid and Ahmed, 2009). Table 3 shows some examples of equivalence rules. Each transliteration unit is assigned a cost. A Hindi character which is always mapped to the same Urdu character is assigned zero cost. In some cases, a Hindi character, say H_1, can be mapped to several different Urdu characters, say U_1, U_2 and U_3. We assign an equal cost of 0.3 to all three mappings H_1 to U_1, H_1 to U_2 and H_1 to U_3 as shown in the last three rows of Table 3.

Hindi character	SAMPA	Urdu character	Cost
ब	b	ب	0
ल	l	ل	0
ज़	z	ض	0.3
ज़	z	ظ	0.3
ज़	z	ذ	0.3

Table 3: Hindi-Urdu handcrafted equivalence rules

The edit distance metric allows insert, delete and replace operations. The handcrafted rules define the cost of replace operations as shown in Table 3. Each insert and delete operation costs 0.6, except for the deletion of Hindi diacritics where the cost is 0.

If two identical characters occur next to each other in an Urdu word then either only one character is written with a shadda sign ّ after it or both characters are written next to each other. The shadda sign is treated as a diacritic by most Urdu writers and is thus frequently omitted in Urdu text. We deleted all shadda characters in a preprocessing step in order to obtain a consistent representation. Hindi, on the other hand, uses a special joining symbol between two characters to write conjuncts. If the joining symbol is used between two identical characters then it will be transliterated with a shadda in Urdu. Assume the joining symbol is "z" and L is a character in Hindi.

The occurrence L"z"L in Hindi will be transliterated as L ّ in Urdu. In the handcrafted rules, we add separate entries mapping Hindi L"z"L to Urdu L.

Urdu and Hindi differ in their word definition for some particular categories. For example, in Hindi the case marker is always attached to the pronoun, whereas in Urdu, the case marker can be written either as a separate token after the pronoun or can be attached to the pronoun. The edit distance metric was modified to avoid penalizing spaces in Urdu text.

The raw list of word pairs contains translations (that are not transliterations), transliterations and alignment errors. We apply the edit distance metric to the list of word pairs and extract the list of transliteration pairs. We optimized the costs on a held-out set. We filter out word pairs with a cost of more than 0.6 thus allowing only one deletion/insertion or at most three ambiguous replacements in the Hindi-Urdu pairs (Table 3). If we decrease the filtering threshold or increase the replacement cost, the number of types extracted reduces significantly. We obtained 1695 types in the list of transliteration pairs. Due to data sparsity, there were about 5 Hindi characters which were not covered in the list of transliteration pairs. We added transliteration units for the missing Hindi characters to the list of transliteration pairs.

We align the list of word pairs at the character level using the same handcrafted equivalence rules and the edit distance algorithm. We get three kinds of alignments of Hindi characters to Urdu characters i.e. $\emptyset \rightarrow 1$, $1 \rightarrow \emptyset$ and $1 \rightarrow N$. The character alignments produced using the edit distance metric differ from those produced using the character aligner (Section 3.1). The character aligner allows only one character on the source and the target side. The edit distance metric allows a Hindi character to align to more than one Urdu character. We postprocess the alignment $\emptyset \rightarrow 1$ as described in Section 3.1.

4 Transliteration Model

The character-based translation probability $p_{char}(H, U)$ is defined as follows:

$$p_{char}(H,U) = \sum_{a_1^n \in align(H,U)} p(a_1^n) \quad (1)$$

$$= \sum_{a_1^n \in align(H,U)} \prod_{i=1}^{n} p(a_i | a_{i-k}^{i-1}) \quad (2)$$

where a_i is an aligned pair consisting of the i-th Hindi character h_i and a sequence of 0 or more Urdu characters. Usually a Hindi character is aligned with one Urdu character, but some

Hindi characters map to zero or two Urdu characters. The short vowels except the short vowel 'a' are always written in Hindi while in Urdu short vowels are usually not written. Hence, Hindi short vowels should be aligned to zero Urdu characters. $align(H, U)$ is the set of all possible alignments between the characters of U and H.

During transliteration we need to maximize $P(H, U)$ over all possible sequences U but we can not efficiently compute the sum over all possible different alignment pairs in equation 1. Therefore we resort to the Viterbi approximation and extract the most probable alignment.

The parameter k in equation 2 indicates the amount of context used (e.g. if $k = 2$, we use a trigram model on character pairs). A good value of k for our transliteration system is 4. Table 5 (Section 5) shows the variation of results on different values of k.

The SRILM-Toolkit (Stolcke, 2002) was applied in the implementation. Add-one smoothing was used for unigrams and Kneser-Ney smoothing was used for order > 1.

5 Evaluation Setup

5.1 Training and Test Data

We use a Hindi-Urdu parallel corpus taken from the EMILLE corpus[3]. In both Urdu and Hindi, there are cases where one character can be represented either as one Unicode character or as a combination of two Unicode characters. These characters are normalized to have only one representation. In Urdu, short vowels are represented with diacritics which are usually missing in written text. In order to keep the corpus consistent, all diacritics were removed from the Urdu corpus.

A Hindi news corpus of 5000 tokens (1330 types) was randomly selected from BBC News. The tokens that can be transliterated into Urdu were manually extracted and a test corpus of 819 transliteration pairs was obtained.

5.2 Word Alignment

We automatically generate two word alignments using GIZA++ (Och and Ney, 2003), and refine them using the grow-diag-final-and heuristic (Koehn et al., 2003). We extracted a total of 107323 alignment pairs from the sentence aligned parallel corpus of 7007 sentences. The M-N and N-1 alignment pairs were ignored as they are unlikely to be transliterations. Most of the 1-N alignment pairs are cases where the Urdu part of the alignment actually consist of two (or three) words which are sometimes written without a space because of lack of standard writing convention in Urdu (Durrani and Hussain, 2010). For example جا سکتے (can go ; d_ZA s@kt_de) is alternatively written as جاسکتے (can go ; d_ZAs@kt_de) , i.e., without a space before the "s" sound. These are always written as a single token in Hindi. We drop 1-N alignments with gaps, but keep alignments with contiguous words. We refer to the word-aligned corpus generated from 1-1 and 1-N alignments as "list of word pairs" later on.

5.3 Baselines

5.3.1 Phrase-based MT

Our first baseline is a phrase-based machine translation system (PSMT) for transliteration built using the Moses toolkit. We use the default settings but the distortion limit is set to zero (no reordering). Minimum error rate training (MERT) is used to optimize the parameters. The list of transliteration pairs is divided into 90% training and 10% development data (used for MERT).

5.3.2 External Transliterators

We also compare our systems with three Hindi-Urdu transliteration systems, HUMT[4] , CRULP[5] and Malerkotla[6] (MAL), available on the internet. HUMT is based on finite state transducers. It implements a phoneme-based mapping scheme between Hindi and Urdu. The HUMT system is described in Section 2 (Malik et al., 2008). CRULP is a rule-based transliterator which uses a direct orthographic mapping between Hindi and Urdu. Little information is available on the method of the Malerkotla transliterator. If there are two legal transliterations of a Hindi word, it transliterates it to the most frequent Urdu word. We suspect that Malerkotla may use a bilingual word list to override the basic transliteration scheme.

5.4 Experiments

Phrase-based MT: We first build a PSMT system on the list of word pairs. Due to the amount of noise in the training data, it shows 45.9% accuracy.

[3] http://www.emille.lancs.ac.uk/

[4] http://www.puran.info/HUMT/HUMT.aspx

[5] http://www.crulp.org/software/langproc/h2utransliterator.html

[6] http://www.malerkotla.org/Transh2u.aspx

No filtering	Automatic	Rule-based
45.9%	70.3%	72.7%

Table 4: Results of Phrase-based MT

The low score of the PSMT system supports our work of extracting the transliteration pairs from the list of word pairs to build a transliteration system.

The PSMT is then trained on the transliteration pairs extracted using the automatic method and the rule-based method. The purpose of this experiment is to compare the quality of the extracted lists by building an identical model on them. The PSMT shows best accuracy on the transliteration pairs extracted using the rule-based method (Table 4). The rule-based extraction method is based on high precision and thus extracted fewer transliteration pairs than the the automatic method. The list extracted using the automatic method contains close transliterations as well, which are word pairs which only differ by one or two characters from correct transliterations. The close transliteration pairs help to learn transliteration information but also add noise to the system.

Our systems: We build two versions of our system, using the list of transliteration pairs extracted in Section 3.1 (AUTO) and using the list of transliteration pairs extracted in Section 3.2 (RULE). We use a context size of $k = 4$ (see eq. 2) for our systems. The results of our transliteration system RULE with different context sizes are shown in Table 5. The accuracy of the transliteration system is stable at context sizes greater than three.

1	2	3	4	5
64.5%	76.3%	80.7%	81.6%	81.6%

Table 5: Accuracies of RULE for different context sizes

AUTO shows an accuracy of 76% on the test data of 819 types as shown in Table 6. It could not learn certain language specific phenomena due to data sparsity. The system had problems to learn the mapping of a Hindi character to an Urdu conjunct. The system could not learn the shadda cases (see Section 3.2). There are 18 types (2% of the test data) with shadda phenomena. AUTO correctly transliterates only 28% of these types. This might be due to the character aligner which can not capture the information where a Hindi character can be aligned to more than one Urdu character

AUTO	RULE	MAL	CRULP	HUMT
76%	81.6%	73.4%	69.8%	69.5%

Table 6: Accuracies of the joint model built on lists from AUTO and RULE, compared with the three baseline transliterators

and vice versa. The other factor is the preprocessing step where we delete diacritics and the character joiner from the Hindi word aligned corpus.

The rule-based system (RULE) shows the best results of 81.6%. It obtains 100% accuracy in transliterating the shadda cases. Due to the inclusion of transliteration units in the training data (Section 3.2), it contains at least one entry of every transliteration unit in its training corpus.

Results of other transliteration systems: We test three other transliterators (HUMT, CRULP and MAL) on the test corpus of 819 types. The results are shown in Table 6. The HUMT system performs worst with an accuracy of 69.5%. The HUMT system does not handle ambiguous characters as mentioned in Section 2. It maps each ambiguous Hindi character to the most frequent matching Urdu character without taking into account the transliteration context. CRULP has difficulty in disambiguating Hindi characters which map to several different Urdu characters. Table 11 shows some examples of such transliteration units. The ambiguous Hindi characters (Table 1) can not be predicted correctly on the basis of the neighboring characters but these Hindi characters (Table 11) can be predicted correctly by looking at the context. MAL mostly performs well on ambiguous Hindi characters. The results of MAL are discussed in detail in the next section.

6 Discussion & Error Analysis

In this section, we discuss the errors made by the transliteration systems by dividing the test data into different subclasses. The transliteration between Hindi and Urdu is strongly motivated by the language of origin and script of the word to be transliterated.

Proper nouns: The test corpus contains a large number of words borrowed from other languages which are differently transliterated to Hindi and to Urdu. Words borrowed from Arabic contain ambiguous characters which make the transliteration task more challenging. Proper nouns form 19% of the test corpus. In a second set of experiments, we evaluated only on the proper nouns from the test

AUTO	RULE	MAL	CRULP	HUMT
59.1%	65.6%	56.5%	56.5%	57.1%

Table 7: Accuracies of AUTO, RULE and three baseline transliterators on proper nouns

corpus. All five transliterators perform poorly in transliterating proper nouns as shown in Table 7.

Most of the proper nouns were names borrowed from English and other languages. We observed that there is sometimes a difference between the pronunciation of borrowed words in Hindi and Urdu. Consider the English name "Donald": the character "a" in "Donald" is transliterated using a long vowel into Hindi as डोनाल्ड (donAld) and using a short vowel into Urdu as ڈونلڈ (don@ld). There are some foreign words which are directly transliterated in Hindi and borrowed from another language in Urdu. Consider the word "America" which is transliterated as अमेरिका (@"mErIk@) in Hindi but borrowed from Arabic as امریکہ (A@mrikA) in Urdu. Table 7 shows the results of our transliterators in comparison with other transliterators.

Ambiguous characters: The ambiguous characters frequently occur in Hindi text and are found in 52% of the types in the test corpus. There are four ambiguous characters as shown in Table 1. For each such character, we extract the tokens containing this character from the test corpus. There were 15%, 19%, 13% and 3.8% occurrences of words with ह (h), स (s), त (t_d) and ज़ (z) respectively. Table 8 shows the results of the three baseline transliterators on these four cases.

	MAL	CRULP	HUMT
ह (h)	74.4%	60.8%	60%
स (s)	69.8%	62.9%	66%
त (t_d)	76.4%	66%	66%
ज़ (z)	32.3%	41.9%	3.2%

Table 8: Results of the baseline transliteration systems on words containing ambiguous characters

Malerkotla shows poor results on words containing ज़ (z). These words form only 3.8% types of the test corpus and thus do not substantially affect the overall accuracy achieved by Malerkotla. Table 9 shows the results of Malerkotla and our transliteration systems. RULE performs best on all cases of ambiguous characters.

Sometimes, the use of several ambiguous characters in a string leads to two legal Urdu words as shown in Table 10. The disambiguation between two legal Urdu words requires word context.

	AUTO	RULE	MAL
ह (h)	69.6%	78.4%	74.4%
स (s)	69.8%	74.8%	69.8%
त (t_d)	77.4%	79.3%	76.4%
ज़ (z)	64.5%	74.2%	32.3%

Table 9: Results of Malerkotla and our transliteration systems on words containing ambiguous characters

Hindi	SAMPA	Urdu-1	Urdu-2
बाज़	bAz	بعض Some	باز Eagle
असरार	A@srAr	اصرار Insist	اسرار Israr
सहरा	s@hrA	صحرا Desert	سہرا Credit

Table 10: Highly ambiguous words in Urdu that have the same sound but that are written with different characters and represent different meanings

Ambiguous transliteration units: There are some characters in Hindi that may map to different Urdu characters depending on the context. Table 11 shows some examples. In the first column, the Hindi characters may map to any of the three Urdu characters in the same row. Sometimes, there is no phonological difference between the Urdu characters but conventionally they are written in one way or the other.

Hindi	SAMPA	Urdu		
◌ा	A	ا	ہ	ع
आ	A	ا	آ	ع
ऊ	AU	ا	او	و

Table 11: Some ambiguous transliteration units

Pronunciation differences between Hindi and Urdu speakers: Different pronunciations of Hindi and Urdu speakers also cause confusion for the transliteration systems. For example, the English word "bazaar" is written in Hindi as बाजार (bAd_ZAr) and in Urdu as بازار (bAzAr). The transliteration system has to disambiguate by mapping the character representing "d_Z" in Hindi to either the "d_Z" sound or the "z" sound in Urdu. Table 12 shows some of these examples.

N-best analysis of RULE and AUTO: The transliterators show poor performance on words containing ambiguous characters. In the 20-best

Hindi	Urdu
बाजार (bAd_ZAr)	بازار (bAzAr)
फोल (fol)	پھول (p_hul)
श्रीलंका (SIril@nkA)	سری لنکا (sIril@nkA)

Table 12: Pronunciation differences between Hindi and Urdu

output, we find the correct solution for many words with ambiguous characters as shown in Table 13. However, if a word contains two ambiguous characters, it was difficult for the transliterator to transliterate it correctly. We hope that the tokens with ambiguous characters can be correctly transliterated using context by a statistical machine translation system. The unknown transliterations in the 20-best output will get lower scores from the language model as compared to known words. If two words in the 20-best output are known, the language model helps to choose the right output based on the word context.

The 10-best and 20-best results of AUTO are competitive with RULE[7]. The automatically extracted list obtains high recall and thus contains close transliterations which RULE's list does not contain. Close transliterations are word pairs which only differ by one or two characters from correct transliterations. The close transliteration pairs are useful for the transliteration system as they provide information about transliteration units and help avoid the problem of data sparseness. However, the transliteration system also learns noise from them and might not produce correct 1-best output. Table 14 shows two examples which are correctly transliterated by AUTO but are wrongly transliterated by RULE in the 10-best output. These examples are difficult to transliterate as most of the characters are ambiguous and have more than one possible transliteration. The system built on AUTO is able to transliterate them correctly as it contains more instances of infrequent ambiguous characters. For the incorporation of a transliteration model in a machine translation system, AUTO would be a better option as it is language independent and has better 10-best and 20-best scores.

[7]We also aligned AUTO with the edit-distance based aligner to verify that alignments differences were not important. The results dropped a little less than 1-point for 1-best, 10-best and 20-best, which is still better than RULE for 10-best and 20-best, so the differences in alignment did not unduly influence the results.

	AUTO	RULE
1-Best	76%	81.6%
10-Best	93.8%	91.5%
20-Best	95.1%	92.3%

Table 13: Comparison of 1-Best, 10-Best and 20-Best outputs of our transliteration systems

Hindi	Urdu	SAMPA
अफ़ीका	افریقہ	AfrIcA
आउट	آؤٹ	OUT

Table 14: In the 10-best output, these examples are correctly transliterated by AUTO but are wrongly transliterated by RULE

7 Conclusion

We have implemented a joint source channel model to transliterate Hindi words into Urdu words. We have used two approaches to extract transliteration pairs from a parallel corpus of Hindi/Urdu – an unsupervised transliteration mining method and a method based on handcrafted rules. We then built models on the automatically aligned orthographic transliteration units of the extracted Hindi/Urdu transliteration pairs. Our best transliteration system achieved an accuracy of 81.6% which is 8% better than the best of three other systems. The 10-best and 20-best results of our transliteration system built on the automatically extracted transliteration pairs showed that it is suitable for integration with machine translation which will allow the use of translation context to choose the best transliteration (Hermjakob et al., 2008; Durrani et al., 2010).

Acknowledgments

The authors wish to thank the anonymous reviewers for their comments. Hassan Sajjad and Nadir Durrani were funded by the Higher Education Commission (HEC) of Pakistan. Helmut Schmid was supported by Deutsche Forschungsgemeinschaft grant SFB 732. Alexander Fraser was funded by Deutsche Forschungsgemeinschaft grant Models of Morphosyntax for Statistical Machine Translation. This work was supported in part by the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886. This publication only reflects the authors' views.

References

Yaser Al-Onaizan and Kevin Knight. 2002. Machine transliteration of names in Arabic text. In *ACL Workshop on Computational Approaches to Semitic Languages*, Morristown, NJ, USA.

Maximilian Bisani and Hermann Ney. 2008. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5).

Nadir Durrani and Sarmad Hussain. 2010. Urdu word segmentation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 528–536, Los Angeles, California, June. Association for Computational Linguistics.

Nadir Durrani, Hassan Sajjad, Alexander Fraser, and Helmut Schmid. 2010. Hindi-to-Urdu machine translation through transliteration. In *Proceedings of the 48th Annual Conference of the Association for Computational Linguistics*.

Asif Ekbal, Sudip Kumar Naskar, and Sivaji Bandyopadhyay. 2006. A modified joint source-channel model for transliteration. In *Proceedings of the COLING/ACL poster sessions*, pages 191–198, Sydney, Australia. Association for Computational Linguistics.

Swati Gupta. 2004. Aligning Hindi and Urdu bilingual corpora for robust projection. Masters project dissertation, Department of Computer Science, University of Sheffield.

Ulf Hermjakob, Kevin Knight, and Hal Daumé III. 2008. Name translation in statistical machine translation - learning when to transliterate. In *Proceedings of ACL-08: HLT*, pages 389–397, Columbus, Ohio. Association for Computational Linguistics.

J C. Wells. 1995. Computer-coding the IPA: a proposed extension of SAMPA. University College, London.

Bushra Jawaid and Tafseer Ahmed. 2009. Hindi to Urdu conversion: beyond simple transliteration. In *Conference on Language and Technology 2009*, Lahore, Pakistan.

Mehdi M. Kashani, Fred Popowich, and Anoop Sarkar. 2007. Automatic transliteration of proper nouns from Arabic to English. In *Second Workshop on Computational Approaches to Arabic Script-based Languages*, Stanford University, USA.

Kevin Knight and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4):599–612.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference*, pages 127–133, Edmonton, Canada.

Haizhou Li, Zhang Min, and Su Jian. 2004. A joint source-channel model for machine transliteration. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 159–166, Barcelona, Spain. Association for Computational Linguistics.

M G Abbas Malik, Christian Boitet, and Pushpak Bhattacharyya. 2008. Hindi Urdu machine transliteration using finite-state transducers. In *Proceedings of the 22nd International Conference on Computational Linguistics*, Manchester, UK.

M G Abbas Malik, Laurent Besacier, Christian Boitet, and Pushpak Bhattacharyya. 2009. A hybrid model for Urdu Hindi transliteration. In *Proceedings of the 2009 Named Entities Workshop, ACL-IJCNLP*, Suntec, Singapore.

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

Vladimir Pervouchine, Haizhou Li, and Bo Lin. 2009. Transliteration alignment. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th IJCNLP of the AFNLP*, Suntec, Singapore.

Hassan Sajjad, Alexander Fraser, and Helmut Schmid. 2011. An algorithm for unsupervised transliteration mining with an application to word alignment. In *Proceedings of the 49th Annual Conference of the Association for Computational Linguistics*, Portland, USA.

Bonnie G. Stalls and Kevin Knight. 1998. Translating names and technical terms in Arabic text. In *Proceedings of the COLING/ACL Workshop on Computational Approches to Semitic Languages*.

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *Intl. Conf. Spoken Language Processing*, Denver, Colorado.