Machine Transliteration

Devanshu Jain

A Thesis

in

Department

For the Graduate Group in Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Master of Science in Engineering

2018

Supervisor of Thesis

Dr. Chris Callison-Burch, Professor, University of Pennsylvania

Reader of Thesis

Dr. Dan Roth, Professor, University of Pennsylvania

Graduate Group Chairperson

Dr. Boon Thau Loo, Professor, University of Pennsylvania

Machine Transliteration

Contents

LIST OF TABLES iv

LIST OF ILLUSTRATIONS v

ACKNOWLEDGEMENT vi

ABSTRACT vii

CHAPTER 1 : Introduction 1

1.1 Contributions 2

1.2 Document Structure 4

CHAPTER 2 : Literature Review 5

2.1 Phoneme-based Methods 5

2.2 Grapheme-based Methods 8

2.3 Hybrid-models 10

2.4 Machine Transliteration in Low Resource Settings 11

CHAPTER 3 : Approach 13

3.1 Methodology 13

3.2 Data 13

3.3 Tool 14

3.4 Evaluation Metrics 14

CHAPTER 4 : Experiments and Results 17

4.1 Experiment 1 17

4.2 Experiment 2 17

4.3 Experiment 3 21

4.4 Experiment 4 22
4.5 Experiment 5 22

CHAPTER 5 : Bi-Lingual Lexicon Induction 25
5.1 Learning Translations via Matrix Completion 25
5.2 Experiment 1 25
5.3 Experiment 2 28
5.4 Experiment 3 29

CHAPTER 6 : When to translate vs transliterate? 34
6.1 Methodology 35
6.2 Data 35
6.3 Results 36

CHAPTER 7 : Conclusion 37
7.1 Future Work 37

APPENDIX 39

BIBLIOGRAPHY 39

List of Tables

TABLE 1 : Experiment 1: original data results 17
TABLE 2 : Experiment 2: Wikipedia backlinks augmented data at threshold 30% 18
TABLE 3 : Experiment 2: Wikipedia backlinks augmented data at threshold 40% 19
TABLE 4 : Experiment 2: Wikipedia backlinks augmented data at threshold 50% 19
TABLE 5 : Experiment 2: Wikipedia backlinks augmented data at threshold 60% 20
TABLE 6 : Experiment 3: Same script augmented data 21
TABLE 7 : Experiment 4: Compositional Transliteration 22
TABLE 8 : Experiment 5: Hindi-origin words (using Regex) 23
TABLE 9 : Experiment 5: English-origin words (using Regex) 24
TABLE 10 : Experiment 5: Hindi-origin words (using Wikidata) 24
TABLE 11 : Experiment 5: English-origin words (using Wikidata) 24
TABLE 12 : Identifying when to transliterate a word in the sentence 36

List of Figures

FIGURE 1 : Hindi - English pronunciation 3

FIGURE 2 : English backlink data Levenshtein distances 19

FIGURE 3 : Hindi backlink data Levenshtein distances 20

FIGURE 4 : Hindi-Urdu Bi-Lingual Lexicon Induction using Urdu as pivot language 27

FIGURE 5 : Adding transliterations (TRANSLIT) as signals for completing the translation matrix. 29

FIGURE 6 : Top-10 accuracy for Hindi (hi), Bengali (bn), Telugu (te), Tamil (ta), Nepali (ne), Ukranian (uk), Bulgarian (bg) and Siberian (sr) to English Translation 32

ACKNOWLEDGEMENT

I would like to thank Derry Wijaya and Chris Callison-Burch for their advice throughout the supervision of this project.

ABSTRACT

Machine Transliteration

Devanshu Jain

Dr. Chris Callison-Burch

Machine Transliteration is the process phonetic translation of a word across different scripts. This is an important and significant task in the field of Natural Language Processing, particularly because of the value it adds to many downstream applications such as Machine Translation, Entity Discovery, Information Retrieval in the context of multilingualism. This is also a difficult task, from the perspective of machine learning especially due to lack of high-quality training data. In this study, we treat the problem as a monotonic Statistical Machine Translation problem and perform various experiments to improve the results in a low resource environment. Furthermore, we incorporate this method in the Matrix Completion framework for Bi-Lingual Lexicon Induction by Wijaya et al. (2017).

CHAPTER 1 : Introduction

Machine Transliteration is the process of phonetic translation of a word across different scripts. When a word is translated from its native script to foreign script, then it is called *forward transliteration*. On the other hand, when it is translated from a foreign script back to its native script, then it is called *backward transliteration*. For example,

औषधालय → Aushdhalaya (forward transliteration)

फार्मेसी → Pharmacy (backward transliteration)

With growing multilingualism, machine transliteration plays an important role in many downstream applications. One such example is Machine Translation Systems. Almost always, named entities (such as names, addresses, technical terms, festivals, etc.) are transliterated while generating annotated parallel corpora. Sometimes, there are no words in the target language corresponding to a word in the source language. Here, machine transliteration proves to be an important module to improve the performance of machine translation.

Another application is Cross Lingual Information Retrieval (CLIR) Systems. Many of the search engines do not consider transliterated content while responding to a query. It can be observed that the same query (for example, song lyrics) returns significantly different results when submitted to a search engine in a transliterated form. In such cases, transliteration system can help improve the recall.

An interesting human behaviour is also observed on websites such as Facebook, Twitter, etc. which are sources for a lot of user generated content. A lot of these posts/tweets are written in user's native language transliterated to Roman Script. Machine Transliteration would be a useful component for text-based applications such as Question Answering, Sentiment Analysis, etc.

One of the major challenges in this problem is lack of direct mappings between language's phonetic structure. This can lead to ambiguity in forward as well as backward translitera-

tions. Once can find many such ambiguities in fig.1. For example, there are some Devanagari alphabets (like फ) that can map to two different combinations of Roman alphabets (*pha*, *fa*). Therefore

फाटक → phatak, fatak

Also, there are some cases where Roman script cannot differentiate between different Devenagari alphabets. For example, the Devanagari alphabets (त , ट) map to the same combination of Roman alphabets (*ta*)

टमाटर → tamatar

tamatar → टमाटर , तमातर

.

This ambiguity can also result from other factors. As Huang (2005) pointed out, source of origin of the word plays an important role in producing the transliterations.

Transliterating to English is different from Romanization, which is a more deterministic process. In Romanization, we look up the pronunciation table and substitute characters. For example, using figure 1[1], we can produce the romanization as follows. We can see that romanized output is very different from the transliterated output.

Input: ऐंजेला

Romanized Output: ainjela

Transliterated Output: Angela

.

1.1. Contributions

In this study, we treat the problem of Machine Transliteration as a monotonic Machine Translation problem by treating each symbol as a word in itself. The above mentioned

[1] http://www.lingvozone.com/Hindi

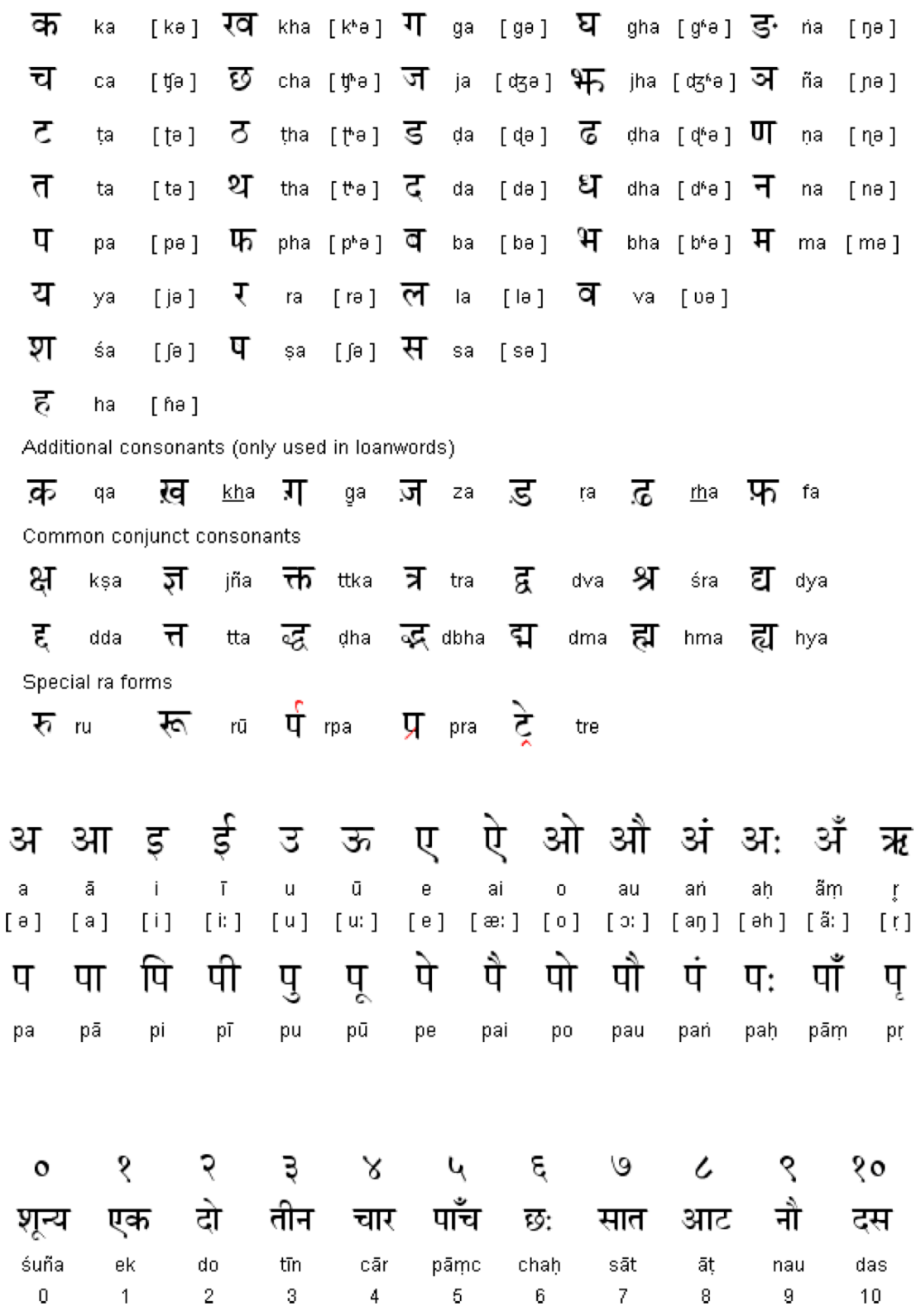

Figure 1: Hindi - English pronunciation

ambiguities can lead to the system learning various rules, all of which are correct but may introduce test error. Thus, we need an appropriate performance measure that takes into account the partial correctness of transliteration process. Since there is a lack of high quality training data which hurdles the creation of accurate models, we perform various experiments that can possibly help alleviate this problem.

We also use our Machine Transliteration system for the task of improving translation via matrix completion. Translation via matrix completion Wijaya et al. (2017) is a new method for learning translations for low resource language pairs. Typically machine translation relies on large volume of bi-lingual parallel text, but matrix completion provides a framework for machine translations using only a small bi-lingual dictionary. We examine the question of whether knowing transliteration and name pairs can improve the translations acquired via matrix completion.

1.2. Document Structure

The rest of the theses is organised as follows. Chapter 2 comprises of literature review and describes some of the related work done in the field of machine transliteration. In Chapter 3, we describe our machine transliteration system and how it is treated as a translation problem. In Chapter 4, we describe the Bi-Lingual Lexicon Induction task and the matrix completion framework by Wijaya et al. (2017). Then, we explain the integration of our machine transliteration system to the framework. Chapter 5 describes the experiments and results conducted by the authors. It begins with describing the experiments for machine transliteration and then goes over the results and its analysis for transliteration as well as for bi-lingual lexicon induction. In Chapter 6, we present our conclusion and provides some possible future work.

CHAPTER 2 : Literature Review

In this chapter, we provide some of the work that has already been done for machine transliteration. After reviewing the literature, we can broadly classify the techniques into three major categories:

- Phoneme-based Methods
- Grapheme-based Methods
- Hybrid Methods

In each of the next three sections, we give a broad overview of work that use these methods.

2.1. Phoneme-based Methods

A major work on the task of Statistical Machine Transliteration was done by Knight and Graehl (1998). They modelled the process of back-transliteration from Japanese (Katakana) to English as a generative process. They identified 5 sub-modules as follows:

1. English phrase is written
2. A translator pronounces it in English
3. The pronunciation is modified into the Japanese phonetic system
4. The sounds are written into Katakana, which is a special phonetic alphabet used to write loanwords
5. Katakana is written

Then, they model each of these sub-modules using the following probability distributions:

1. *P(w)* to model the generation of (scored) English phrases
2. *P(e | w)* to model the generation of English phonemes from the graphemes

3. *P(j | e)* to model the generation of Japanese phonemes from English phonemes

4. *P(k | j)* to model the generation of Katakan graphemes from Japanese phonemes

5. *P(o | k)* to model the misspellings caused by optical character recognition

They model each of these process using Weighted Finite State Machines. They used Acceptors for the first task and transducers for the rest. They modelled *P(w)* using simple unigram scoring method multiplying the scores of known words in the phrase. They used 262,000 frequency list from the Wall Street Journal (WSJ) corpus. They used CMU Pronunciation Dictionary to create the Transducer for *P(e | w)*. They learnt the transducer for modelling *P(j | e)* using an Expectation-Maximization algorithm to generate symbol-mapping probabilities. They manually constructed two transducer for *P(k | j)* - to merge long Japanese vowel sounds to new symbols and then to map the Japanese sounds to Katakana characters. They view Optical Character Recognition as a channel that introduces noise to the gold katakana sequences. They learnt the transducer using another Expectation-Maximization algorithm.

So, given a Katakana string o, they find the English word as follows:

$$w = argmax_{\hat{w}} \sum_{e,j,k} P(\hat{w})P(e|\hat{w})P(j|e)P(k|j)P(o|k)$$

They used the cascade in reverse by using Bayes' rule to get the English phrases from the Katakana phrases. For inference, they used Dijkstra's shortest-path algorithm. To generate, k-best transliterations, they also used Eppstein's k-shortest-path algorithm.

Later, Stalls and Knight (1998) extended this model to learn back-transliteration from Arabic to English, making some changes. Since the first two modules only used English, they were just used directly. Instead of modelling English phonemes to Japanese phonemes

and then to Katakana graphemes as a 2-level process, they integrate these steps. That is, they directly model the process of converting English pronunciation to Arabic characters. Due to the lack of a large English-Arabic dictionary at the time, they manually created a small 150 words dictionary and used English pronunciation dictionary to get mappings from English pronunciations to Arabic words. They used an Expectation-Maximization algorithm to learn the mappings from English phonemes to Arabic graphemes.

Meng et al. (2001) presented an English-Chinese transliteration. Following a phoneme-based approach, they translated the English phrases to phonemes. Since Chinese is monosyllabic in nature while English is not, they identify and apply some phonological rules to transform these pronunciations. Then, they learn phoneme alignments between English and Chinese using Weighted Finite State Transducers. They also compared the generated Chinese pronunciations with the references to create a confusion matrix. They used this confusion matrix to generate a lattice structure where each syllable has alternates which were determined using the statistics from the confusion matrix. Finally, they used a bi-gram syllable level language model to decode the Chinese word from these pronunciations.

Jung et al. (2000) built an English to Korean transliteration system. They used Oxford computer-usable dictionary to generate English pronunciations from words. To generate the most probable Korean word from this representation, they used a probabilistic tagger. Instead of using conventional markov window of size 2, they used a window of size 4 to use as contextual information to train the tagger. For inference, they used Viterbi algorithm to generate k-best transliterations.

Virga and Khudanpur (2003) built a phoneme-based English-Chinese transliteration system. They convert English words to pronunciations, deterministically using a dictionary. They translate English phonemes to syllabic units using traditional Source Channel models usually used in Statistical Machine Translation. Then, they convert these syllabic sequences into pin-yin symbols, which are then translated to character sequences. Unlike other works, which use accuracy as a metric to measure the performance of their transliteration system,

they used an extrinsic evaluation by testing their transliteration system for cross-lingual spoken document retrieval.

2.2. Grapheme-based Methods

The phoneme-based methods of transliterations correctly model the *transliterator* in the way that transliteration utilizes the phonetic representation of words while translating symbols. However, these methods have severe disadvantages as well. As Al-Onaizan and Knight (2002) point out, a major drawback is the generation of pronunciations. Assuming the source to be English language, pronunciation can be generated for well-known English words but for words whose origin is a foreign language, the pronunciation may not be always accurate. Also, due to various intermediary steps involved in the phoneme-based methods, the error propagates through the pipeline which adversely affects the final results. Furthermore, there are cases when words are transliterated based on their spelling in the source language. In such cases, a spelling based model would be more useful.

This has encouraged people to look into grapheme-based methods which directly translate between scripts without going through the process of modelling pronunciation. Kang and Choi (2000) follow such a method to learn transliterations from English to Korean. They learn the alignments between English and Korean symbols using a modified Covington's alignment algorithm. Covington's algorithm used match and skip operations on each step while stepping through the words. It only produced one-to-one alignments. However, Korean-English transliteration has many-to-many correspondence. To take this into account, they introduced a bind operation. After learning the alignments, they generated a training set to predict the English symbol(s) using the neighbouring characters in a window of size 6 (3 left and 3 right). They learnt 26 decision trees - for each English alphabet. To infer the Korean transliteration for an English word, they step through the word and use the corresponding decision tree to generate the corresponding Korean symbol. The final string is the concatenation of these symbols. To prevent the decision trees from overfitting, they used the post-pruning method (reduced error pruning). This method can be used to

generate transliteration in either direction.

AbdulJaleel and Larkey (2002) also use a grapheme-based approach to learn transliterations from English to Arabic. First, they use GIZA++ to learn alignments between English and Arabic words. Whenever an Arabic symbol is aligned with multiple English symbols, the English symbol sequence is added to the English alphabet and the English words are re-segmented. Alignment model is re-trained on this new dataset (with re-segmented English words). Conditional probabilities are calculated based on the alignment counts. To infer the Arabic words, each English word is re-segmented as above and alignment model is used to infer all possible transliterations. These transliterations are scored by the product of alignment probabilities and an Arabic conditional character-level bi-gram model. Pingali et al. (2008) learn English to Hindi transliteration system using a similar approach. However, instead of using simple count to calculate probabilities, they use Conditional Random Field. For each aligned symbol pair, they generate features which consisted of neighbouring English alphabets in a window of 5 characters.

Haizhou et al. (2004) used the grapheme-based approach to learn English-Chinese transliteration. Instead of learning in any particular direction, they learn a joint model, that is, how source and target words can be generated simultaneously. First, they learn alignments between English and Chinese words using an Expectation-Maximization algorithm. Then, for each word pair (E,C), they identify each aligned symbol pairing as a transliteration unit. They learn a n-gram transliteration model as the conditional probability of a transliteration unit given the n immediate predecessor pairs. The probability of the word pair (E,C) is the product of probability of each of its transliteration units as calculated by this model. These joint probability distribution can be easily marginalized to calculate conditional distribution for transliteration in both the directions.

In 2009, Named Entities Workshop (NEWS) introduced a Machine Transliteration task. The organizers provided standard dataset for various language pairs like English-Hindi, English-Hebrew, Chinese-English, Arabic-English, etc. Many of the participating teams

used neural network to learn the target transliterations. Looking at the participating teams, it seems that neural networks are becoming more predominant for solving the machine transliteration task and they consistently outperform the phrase-based machine translation systems. NICT's submission (Finch et al. (2016)) was the best performing team in 2016 version of the task. They trained a LSTM-based RNN to encode the input sequence to a hidden representation and decode that representation to produce the output sequence. It has been observed that since the errors accumulate while decoding, the transliteration quality of the suffixes degrades. To fix this, they used target-bidirectional models which learn to generate the target from left to right and from right to left producing 2 k-best lists. Then, they learn an agreement model to combine them. They used ensembles of such neural networks to generate the transliterations by linearly interpolating (with equal weights) the probability distributions over the target vocabulary during beam-search decoding process.

2.3. Hybrid-models

Al-Onaizan and Knight (2002) extended their previous phoneme-based models to develop a hybrid approach. They developed a grapheme-based model that was integrated with a phoneme-based model to learn Arabic-English transliterations. They learn grapheme and phoneme based discriminative models to learn the $P(w|a)$, the probability of w (english word) being the transliteration of a (arabic word). Then, they linearly interpolate these scores to generate the final score for the transliteration. Inference is done by searching for the English word that maximises this final score. They also performed some additional post-processing to improve their final results. First, they created another Finite State Machine to correct the misspellings. The weights of this machine were set manually as enough misspelling training data was not present to tune the parameters empirically. Furthermore, the outputs from spelling based model were discarded which had 0 web counts.

2.4. Machine Transliteration in Low Resource Settings

Most of the approaches described required extensive training data and other linguistic resources (for phoneme based approaches) which may not be readily available for low-resource languages. This section describes the work undertaken to solve the task of transliteration in a low-resource setting.

Chinnakotla and Damani (2009) focused on general transliteration (forward/backward agnostic) in a low-resource setting. They built their system for transliterating between Hindi and English languages. They employed hand-written character mappings for converting each Hindi word to an English word and vice-versa. Using these individual character mappings, they generate an unordered list of target candidates. They also trained a Character-Level Language Model using massively available monolingual resources for both the languages and used the language model probability to re-rank the candidate list. To resolve the bias towards shorter words, they augmented the target language's alphabets with common digraphs, double letters, etc.

Kumaran et al. (2010) built a language-independent general transliteration system. They used a Conditional Random Field (CRF) model to learn transliteration between languages. For their experiments, they focused on Indian languages and English. As a preprocessing step, they trained a Character-Level Language Model for Indian origin languages collectively as well as English language. Then, they trained a classifier to classify the input words as originating from Indian language or not. Then, they separated the training data using this classifier and trained a CRF based transliteration model for each class. They also experimented with compositional transliteration to account for the cases when no direct data is available for learning transliteration from one language to another. An intermediate language was chosen and two separate CRF models were learnt for (source $\rightarrow$ intermediate) and (intermediate $\rightarrow$ target). Considering that this makes transliteration process noisier, the top-1 accuracy dropped only by less than 10% as compared to the direct transliteration modelling. They also experimented with combining the transliteration score from the direct

transliteration and compositional transliteration for the final system output. This increased their top-1 accuracy by 8% relative to the direct transliterations.

CHAPTER 3 : Approach

3.1. Methodology

For this project, we treated machine transliteration as a Hierarchial Phrase-based machine translation task. We treat words as sentences and characters as words. We employ Statistical Machine Translation tools to learn the transliterations. The transliteration task is much simpler than translation in principle - In translation, the phrases can be reordered in the target language, however, character sequences are always monotonic in transliteration.

3.2. Data

Irvine et al. (2010) used data comprising of named-pairs extracted from Wikipedia articles. Wikipedia maintains links between articles written in different languages. They assumed that person named-entities are mostly transliterated rather than translated from their canonical language to the target language. They mined English Wikipedia category pages like: "1961 births" to get a list of people who were born or died in years ranging from 0 to 2010. They made use of such listings and the language links associated with each person to create data consisting of people names in various languages. Data was collected for approximately 200 languages.

Data was further cleaned for use. Some titles were not consistently transliterated. Reasons for this included abbreviations (A.P.J. Abdul Kalam in Roman written as Abul Pakir Jainulabdeen Abdul Kalam in Devanagari). Another reason was that sometimes, middle name was omitted during transliteration i.e. Abbot Suger in Roman was written as Suger in Russian. They computed word-alignments for the data and chose a threshold score for removing the extreme cases of such occurrences.

We have used the same data for our experiments as well. Particularly, we use the language pair Hindi-English. Hindi is written in Devanagari script and English is written in Roman script. The data comprised of approximately 10,287 pairs.

3.3. Tool

For our study, we used Joshua Li et al. (2009) tool. A step-by-step process of training and decoding is described as follows:

1. **Data preprocessing**: The parallel training data consists of words written in different scripts per line. Each such word is processed by lower-casing the characters and replacing the spaces with special character: underscore ('_'). The resulting word is then represented as space separated list of characters.

2. **Creating alignments**: We used Berkeley aligner to create alignments between the source and the target language. We trained 2 IBM model 1s jointly - one for each direction for 5 iterations and used these parameters to initialize HMM alignment model.

3. **Training language model**: We used Berkeley LM to create a Language model of order-8 for the source and target languages.

4. **Extracting grammar**: We used 'Thrax' to extract translation grammar rules using the alignment model created above along with their feature function scores.

5. **Training**: We used Joshua's Minimum Error Rate Training (MERT) to learn the weights of the features for the translation grammar rules extracted above.

6. **Decoding test sentences**: Given the grammar rules and the source sentence, s in the test set, we used Joshua's decoding algorithm (based on Chart Parsing) to find k-best derivations $<s, t>$ where t is the sentence in target language.

3.4. Evaluation Metrics

We used the following evaluation metrics to calculate the system performance following Duan et al. (2016). Instead of having a list of reference transliteration in the dataset (which is assumed by Duan et al. (2016)), we only have 1 reference transliteration. The

evaluation formulae are simplified according to this. Here, N is the number of datapoints. The transliteration system outputs a list of transliterated candidates of length n_i for each word w_i. c_{ij} is the j^{th}-best transliterated word for word w_i. r_i is the correct transliteration (reference) for w_i

1. **Average Normalised Edit Distance:** We calculate standard Levenshtein distance between two words to calculate the similarity between two words. It measures the number of insertions, deletions and substitutions we need to make in order to transform one word to another. This measure is then normalised by the length of reference string and is averaged over the test data.

2. **Top-1 Word Accuracy:** It measures the proportion of data which produces the correct transliteration (exact matching) as the top transliterated candidate with respect to the reference transliteration.

$$accuracy_i = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} 1 & \text{if } r_i = c_{i1} \\ 0 & \text{else} \end{cases}$$

3. **Top-K Word Accuracy:** It measures the proportion of data which produces the correct transliteration (exact matching) in the top-k transliterated candidate list with respect to the reference transliteration.

$$top\ k\ accuracy_i = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} 1 & \text{if } \exists j : r_i = c_{i,j}, 1 \leq j \leq k \\ 0 & \text{else} \end{cases}$$

4. **Top-1 F-Score:** The F-score is the harmonic mean of Precision and Recall. Precision and Recall is calculated using Edit Distance between the top transliterated

candidate and the reference. Following formulae are calculated:

$$LCS(c, r) = \frac{1}{2}(|c| + |r| - ED(c, r))$$

Here, LCS is Least Common Subsequence and ED is the edit distance. Then,

$$precision_i = \frac{LCS(c_{i,1}, r_i)}{|c_{i,1}|}$$
$$recall_i = \frac{LCS(c_{i,1}, r_i)}{|r_i|}$$

CHAPTER 4 : Experiments and Results

We performed the experiments using Monte-Carlo Cross Validation. We randomly sampled (without replacement) some fraction of the data for training and used the remaining data for tuning and testing. We repeated this process 10 times.

4.1. Experiment 1

The objective of this experiment was to confirm the hypothesis that the with increase in the the size of training data, the system performance improves. The Hindi-English data was divided into 3 parts: training, tuning and testing for each repetition. For each repetition, we held back 60%, 70%, 80% and 90% data for the training. Then, we used half the remaining data for tuning and the other half for testing. We calculate the results after averaging the scores over all the repetitions. The results are presented in table 1. Observe that the performance improves as the training data increases

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.1934	0.4363	0.4906	0.8619
70% training data	0.1846	0.4658	0.5191	0.8648
80% training data	0.1793	0.4867	0.5413	0.8668
90% training data	0.1723	0.5105	0.5642	0.8698

Table 1: Experiment 1: original data results

4.2. Experiment 2

The previous experiment validates our claim that increasing training data indeed improves the performance of the system. To build up on that, we tried to extract more data from Wikipedia. Wikipedia API[1] provides the ability to access the backlinks to every every wikipedia page. Using the data already collected, we used this service to gather the titles of all wikipedia pages which are linked to these pages. For example, given a page titled "*Barack Obama*", then its backlinks in the English wikipedia consists of pages titled:

[1] `https://en.wikipedia.org/w/api.php?action=query&list=backlinks&bltitle=Barack\%20Obama&bllimit=5&blfilterredir=redirects`

1. Barak Obama
2. Barack H. Obama
3. Barach Obama
4. Barack Hussein Obama, Jr
5. 44th President of the United States

and so on.

We also extracted the backlink pages for the Hindi wikipedia. We augment the data with all such name variants. This increases out data set size to approximately 100,000.

However, as can be observed in the above example, this data needs to be cleaned. The 1^{st}, 2^{nd} and 3^{rd} item seems fine for augmenting. The 4^{th} item seems okay but 5^{th} item should definitely not be augmented in the training data. To analyse the distance between the original word and the name variants, we calculated the Levenshtein distance between the variant and the original word and normalised it by the length of the original word. The histogram of these distances is shown in figures 4.2 and 4.2. We re-run the experiment 1 as before with the augmented data created by using different level of thresholds of the normalized Levenshtein distance. We used the thresholding levels of 30%, 40%, 50% and 60%. Our hypothesis is that the performance should improve for this augmented data.

The results for all the above thresholds are presented in tables 2, 3, 4 and 5.

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.3525	0.3616	0.4231	0.8560
70% training data	0.3477	0.3846	0.4434	0.8582
80% training data	0.3402	0.4058	0.4655	0.8609
90% training data	0.3494	0.4215	0.4743	0.8596

Table 2: Experiment 2: Wikipedia backlinks augmented data at threshold 30%

As can be observed, the system achieves the best performance when a threshold of 40%

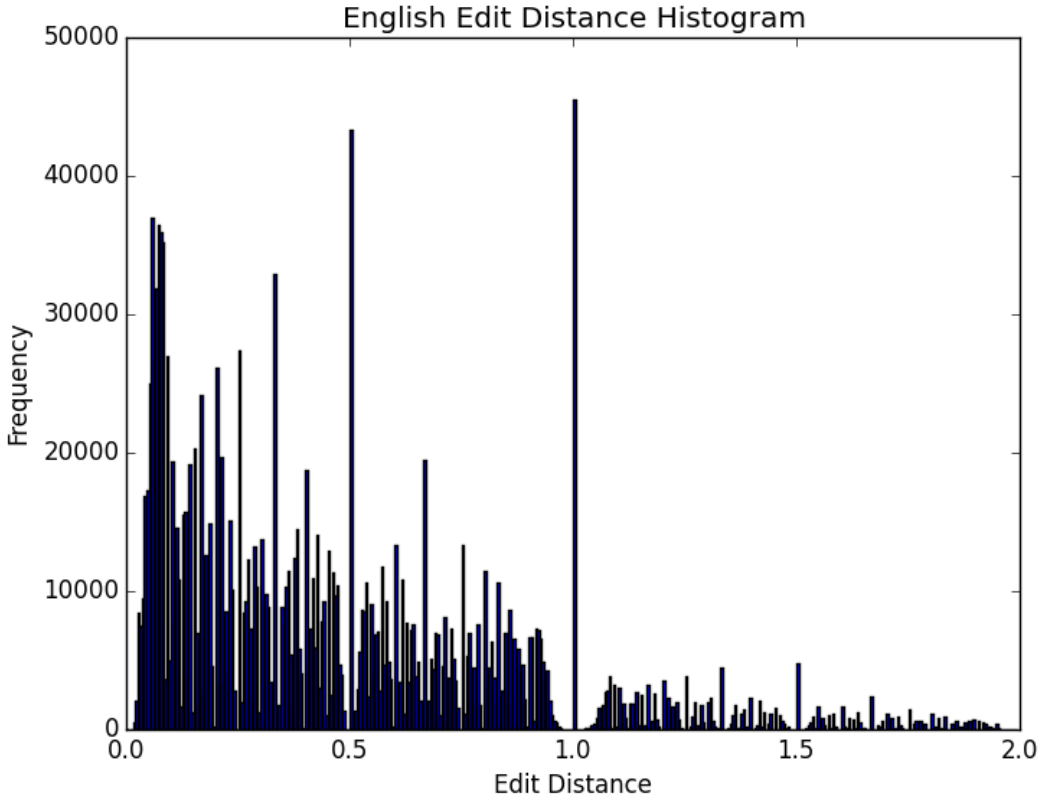

Figure 2: English backlink data Levenshtein distances

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.3435	0.3697	0.4311	0.8578
70% training data	0.3370	0.3945	0.4552	0.8605
80% training data	0.3296	0.4138	0.4764	0.8632
90% training data	0.3212	0.4442	0.5029	0.8656

Table 3: Experiment 2: Wikipedia backlinks augmented data at threshold 40%

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.3126	0.2956	0.3996	0.8763
70% training data	0.3094	0.3110	0.4188	0.8781
80% training data	0.3030	0.3263	0.4371	0.8801
90% training data	0.3022	0.3516	0.4630	0.8807

Table 4: Experiment 2: Wikipedia backlinks augmented data at threshold 50%

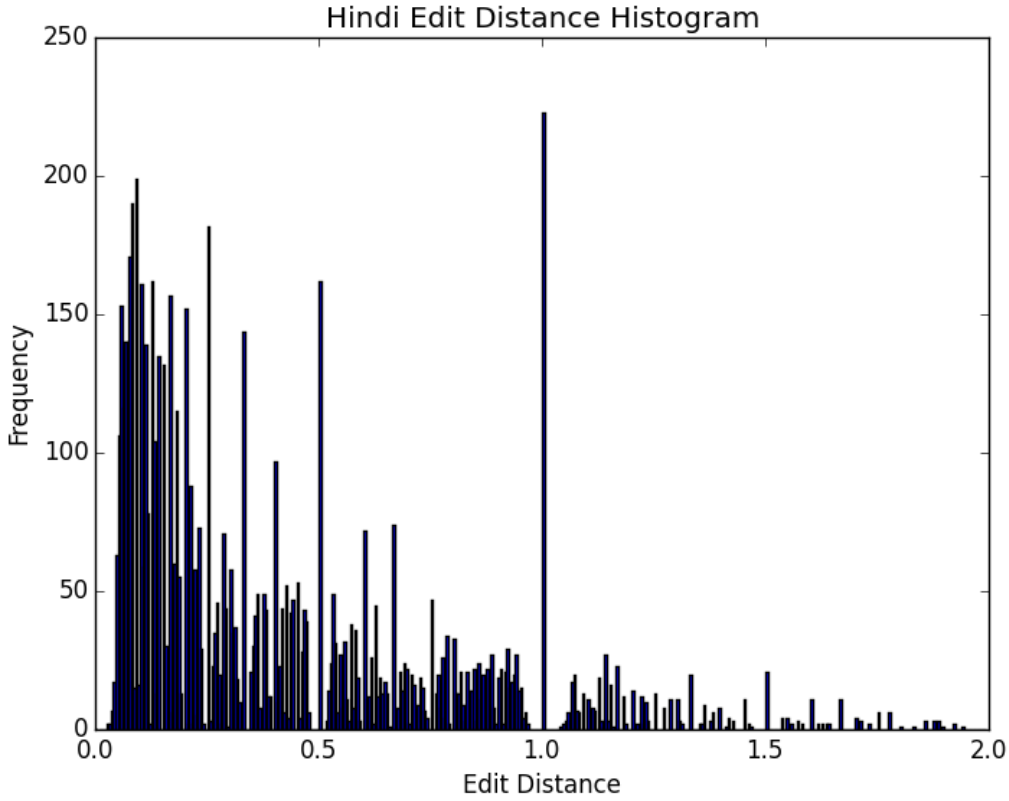

Figure 3: Hindi backlink data Levenshtein distances

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.3370	0.2211	0.3283	0.8748
70% training data	0.3331	0.2287	0.3407	0.8768
80% training data	0.3322	0.2403	0.3545	0.8783
90% training data	0.3274	0.2432	0.3610	0.8804

Table 5: Experiment 2: Wikipedia backlinks augmented data at threshold 60%

was chosen. However, this performance does not beat the performance of the system which was trained on the original data (without augmentation). We think that this is because the name variants add too much noise in the dataset.

We also observe that even though the accuracy metrics show a decline in performance but the average normalised edit distance is still about the same, which shows that the decoded

words in the target language are still discernible which is a good sign.

4.3. Experiment 3

Although the experiment of increasing our dataset by using wikipedia backlinks didn't work out, but that was mainly because of the noisy data. In this experiment, we tried to augment our dataset by including data from other languages written in the same script as Hindi: Devanagari. We created our new dataset by combining the parallel data from languages[2]

1. Sanskrit-English
2. Marathi-English
3. Bihari-English
4. Nepali-English
5. Hindi-English

The final dataset consists of 27,680 items. The hypothesis here is that accumulating data from same script will increase the training data and according to the results from the first experiment, it should improve the system performance significantly. The results from the experiment are presented in the table 6.

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.1836	0.4798	0.5340	0.8652
70% training data	0.1776	0.5090	0.5607	0.8677
80% training data	0.1716	0.5351	0.5854	0.8702
90% training data	0.1653	0.5549	0.6065	0.8721

Table 6: Experiment 3: Same script augmented data

We observed that the system's performance improved significantly as compared to the case when we just used Hindi-English parallel data, thus confirming the hypothesis.

[2] `https://en.wikipedia.org/wiki/Devanagari`

4.4. Experiment 4

In this experiment, we try compositional transliteration, that is, given the source, intermediate and target languages (X, Y and Z), we train the transliteration system and create a model for the pairs: X→Y and Y→Z. Then, for the test set of words in the source language X, we use the transliteration model for X→Y pair and get the words transliterated in the intermediate language Y. Then we use the transliteration model for Y→Z pair to get the words transliterated in the target language.

This might be useful in the cases, where no or very little data exists for a pair of languages but we can use another language as an intermediate for which parallel data exists for both the languages and obtain the transliterations. For the purpose of this experiment, we chose Arabic as our intermediary language for transliteration between Hindi and English. The experiment for each repetition was created as follows: follows: Some percentage of data for X→Z was held for testing and the rest of the data along with script Y was used to train 2 different models. The results are presented in the table 7.

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.3787	0.1232	0.1079	0.8077
70% training data	0.3753	0.1244	0.1088	0.8083
80% training data	0.3538	0.1298	0.1171	0.8138
90% training data	0.3340	0.1310	0.1252	0.8187

Table 7: Experiment 4: Compositional Transliteration

We observed that the system's performance declined significantly from before. But still, the average normalized edit distance is not significantly bad. The decoded words are still discernible which might be useful for low-resource languages.

4.5. Experiment 5

Huang (2005) have shown that having the data corresponding to the word's source of origin may boost transliteration performance. They do it by using the group-average agglom-

erative clustering technique to group the data into different classes and then using the transliteration system on each of those classes. To get the named-entity source of origin for our data, we first tried to get the entity's nationality by parsing the original Engish wikipedia article and using regular expression to parse B from "A is a B" type of structure in the first sentence. For example, say the sentence is:

Amitabh Bachchan (born 11 October 1942) is an Indian film actor, producer, television host, and former politician.

Then, the parser will produce '*Indian*'.

We grouped the dataset according to the nationality of entities (since all of the items in the data set are people's names). Thereafter, we divided the training data into 2 parts: those which were Indian and those which were English and then, trained separate models for them and calculated the performance for each group separately. The hypothesis is that since geographically-distant languages have a very different structure, the variety of structure in the training data may make it hard for the learning algorithm to learn the correct rules. So, we create a group for those items with the nationalities: India, Pakistan, Bangladesh, Nepali, etc. and another group for items with nationalities: United States, United Kingdom, Candada, Australia, etc. The results are presented in the table 8 and 9.

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.1641	0.4580	0.5225	0.8728
70% training data	0.1591	0.4786	0.5430	0.8750
80% training data	0.1560	0.4962	0.5590	0.8754
90% training data	0.1643	0.5010	0.5641	0.8739

Table 8: Experiment 5: Hindi-origin words (using Regex)

Next, we used Wikidata API to get more accurate information about the citizenship of the items. Wikidata provides the country of citizenship[3] data as a property for the wikipedia page titles which can be easily queried. We re-grouped the items as above using this data and

[3] `https://www.wikidata.org/wiki/Property:P27`

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.1847	0.3573	0.3989	0.8599
70% training data	0.1783	0.3893	0.4322	0.8623
80% training data	0.1707	0.4212	0.4647	0.8646
90% training data	0.1514	0.4654	0.5124	0.8687

Table 9: Experiment 5: English-origin words (using Regex)

re-run the experiment. The new results are presented in the tables 10 and 11. We observe that although the performance is better for the Hindi-origin group, but the performance declines for the English-origin group. The main reason for that would be because there is very less data for English-origin group: 1700 English group versus 5000 Hindi group.

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.1719	0.4867	0.5498	0.8703
70% training data	0.1708	0.5112	0.5704	0.8710
80% training data	0.1722	0.5339	0.5919	0.8719
90% training data	0.1670	0.5674	0.6197	0.8740

Table 10: Experiment 5: Hindi-origin words (using Wikidata)

	Average Normalised Edit Distance	Top-1 Accuracy	Top-10 Accuracy	F-Score
60% training data	0.1962	0.3613	0.4101	0.8598
70% training data	0.1809	0.4062	0.4539	0.8629
80% training data	0.1714	0.4334	0.4816	0.8665
90% training data	0.1688	0.4641	0.5074	0.8673

Table 11: Experiment 5: English-origin words (using Wikidata)

CHAPTER 5 : Bi-Lingual Lexicon Induction

In this chapter, we experimented with using transliteration to improve Bi-Lingual Lexicon Induction task. We integrate our approach with Wijaya et al. (2017)'s matrix completion framework. In the following sections, we describe their framework briefly and then we describe our experiments with the integration and the results obtained.

5.1. Learning Translations via Matrix Completion

In this section, I will briefly describe the matrix completion framework by Wijaya et al. (2017) for bi-lingual lexicon induction. They formulate the problem as follows: *Given a set of observed translation between languages F and E, say, T = <f,e>: f ∈ F and e ∈ E, a word f from F and a set of the target words e: <f, e> ∉ T, the objective is to identify the score $x_{f,e}$ which determines how likely is e a translation of f.*

They model this problem as that of completing a matrix X: E × F. They approximate the matrix X as $\hat{X} = PQ^T$, where P: |F| × k and Q: |E| × k. Each row in P (or Q) can be seen as a feature vector for the word in language F (or E). They used the Bayesian Personalized Ranking (BPR) to do this matrix factorization.

However the initial matrix can be very sparse due to small seed dictionary. To overcome that problem, the authors used a variety of bilingual signals such as Wikipedia interlingual links to add additional entries in the matrix T. They also used monolingual signals such as word embeddings as word features. These word embeddings were bi-lingually informed. This was achieved by training a neural network to map all the word embeddings to a common space.

5.2. Experiment 1

In this experiment, we experimented with using a 3^{rd} language as a pivot in the framework to translate a word from the source language to the target language. We chose our source language as Hindi and the target language as English. We chose the intermediary language as Urdu.

5.2.1. Hindi-Urdu Transliteration or Translation

The hypothesis here is that the Hindi and Urdu words are pronounced similarly most of the time. If this hypothesis is true, then it would mean that words written in Urdu and Hindi should be transliterated and not translated. So, to test this hypothesis, we extracted a small dictionary of English - Urdu words [1]. The data consists of the English words, their Urdu translations and the transliterated version of the Urdu words. For example:

1. weather موسم meosem
2. head سر ser
3. colors رنگ renegue

Our final data consisted of approximately 600 words. This data comprised of words from categories such as numbers, colours, body parts, time, days of the week, food items, animals, places, objects, clothes, nature, weather, relationships, verbs and some popular urdu phrases as well among others.

Then, we manually translated the English words to the Hindi and verified the proportion of Hindi words that approximately or exactly sounds like the Urdu words. The results showed that approximately 82% of the time, the urdu words were direct transliterations of the Hindi words.

5.2.2. Methodology

The methodology can be explained through the diagram in figure 5.2.2. The diagnol regions 1, 5 and 9 are simply the identity sub-matrices. The entries in regions 3 and 9 are filled using an existing seed Urdu-English dictionary. The entries in regions 6 and 8 are filled using a transliteration model trained to transliterate words from Hindi to Urdu. The regions 2 and 4 are filled using the Hindi-English training data and it also consists of the testing data.

[1] `http://mylanguages.org/learn_urdu.php`

The testing data are empty rows / columns in the sub-matrix.

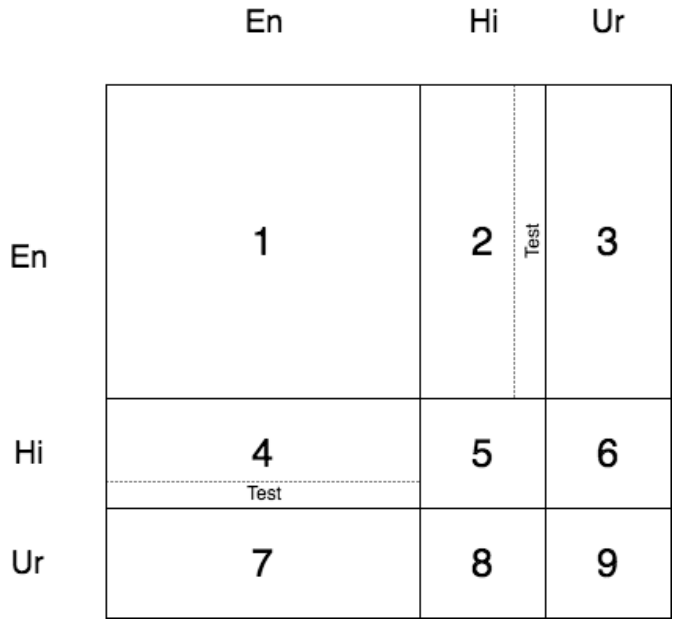

Figure 4: Hindi-Urdu Bi-Lingual Lexicon Induction using Urdu as pivot language

Our hypothesis is that since Hindi and Urdu are mostly transliterated, augmenting the matrix with Urdu transliterations and Urdu-English seed dictionary would provide additional signals to improve the quality of Hindi-English translations.

5.2.3. Data

Based on the above result, we assume every Hindi-Urdu word pair to be a transliteration pair. We train a transliteration system with source as Hindi and the target as Urdu language. We extracted the data by using Wikipedia API to extract the Hindi and Urdu Wikipedia articles of the people who were born or died in years from 0 to 2017. This was done because the data extracted earlier from English wikipedia consists of many pages with no interlingual links in both: Hindi as well as Urdu. Also, there may exist some pages in Hindi/Urdu wikipedia with no link to English wikipedia and therefore would have been missed out during the extraction process. Our final data consisted of approximately 24,000 word pairs. We use this data to learn a transliteration model from Hindi to Urdu language.

For our experiment, we used 9,000 pairs of English-Hindi words as training data and 1,000 English-Hindi words for testing. For each of the Hindi word, we generated top-100 Urdu transliterations. We also used a seed dictionary between Urdu and English.

For monolingual auxillary signals, we trained a Word2Vec Embedding model for Urdu language. The data used to train the Urdu model consisted of 107,000 documents and 50 million words from BBC news data. The data was preprocessed using sentence splitting, tokenizer and normalizer. We trained a neural network to learn mapping from English-Urdu embeddings.

5.2.4. Results

We evaluated the results by measuring the top-10 accuracy, i.e. if the correct English translation was present in the top 10 candidates generated by the system. We achieved a top-10 accuracy of 67.7%, which is slightly lower than that achieved without using the pivot language. We think that this is because of the poor quality of seed English-Urdu dictionary. By inspecting, we found that there were too many cases of polysemy. For example, *dirt* and *opaque* were both translated to غلاظت

5.3. Experiment 2

5.3.1. Methodology

For our second experiment, we move away from the idea of using a pivot language. Instead, we augmented the matrix by adding the English transliterations of the Hindi words to the vocabulary as well. After doing so, we augment our training data to include the top-3 English transliterations for each Hindi word as well as the English words in the original vocabulary within an edit distance of 1 from these transliterations.

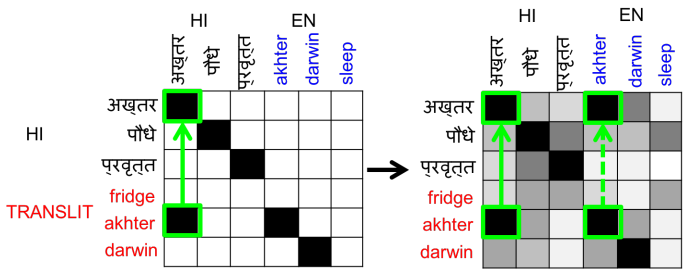

Figure 5: Adding transliterations (TRANSLIT) as signals for completing the translation matrix.

5.3.2. Data

We train a model to learn transliterations from Hindi to English language as described in Chapter 4. We extracted the data using Wikipedia interlingual links to get the English Wikipedia pages for all the people who were born or died in years from 0 to 2017. We used the titles of these pages and the interlingual links (to access the Hindi Wikipedia) to generate the transliteration training data.

5.3.3. Results

We used the modified matrix in Wijaya et al. (2017)'s matrix completion framework. We evaluated the results using top-10 accuracy as before. We achieved a top-10 translation accuracy of 49.4 % which is considerably worse than what was achieved before.

5.4. Experiment 3

5.4.1. Methodology

For our third experiment, we try to re-rank the list of translation candidates output by the system using the transliteration data, heuristically. To generate candidate translations for a source word, we first produce the top-3[2] transliterations of the source word in the target language using our transliteration system. We then collect all words in the target

[2]We choose to use only the top-3 transliterations for each source word for efficiency reason, as we need to compute their edit distances to all the words in the target language vocabulary, which in our experiments contains the 100K most frequent words in the Wikipedia of the language

language that are close – defined as having an edit distance of 1 or less – to any of the transliterations. We select target word(s) with the most number of close transliterations as candidate translations for the source word.

We use candidate translations that are generated from transliterations as observed translations in the translation matrix X (TRANSLIT in Figure 5). Given the observed translations, we infer the score of how likely a source word f (in the row of the matrix) and a target word e (in the column of the matrix) to be translation of each other using matrix factorization (MF) Koren et al. (2009):

$$\hat{x}_{f,e}^{mf} = p_f^T q_e$$

where p_f and q_e are respectively, the row and the column of two low rank matrices that are used to approximate X.

Since the observed candidate translations that we obtain from transliterations may be sparse, similar to BPR, we use auxiliary signals for measuring translation equivalence that are based on similarities between the words' embeddings in the joint embedding space of the source and the target language:

$$\hat{x}_{f,e}^{aux} = \theta_f^T \theta_e + \beta^T \theta_e$$

where θ_e represents the target word's embedding in the joint space and θ_f is the feature vector whose dot product with θ_e models the extent to which the embedding of the source word f matches the embedding of the target word e.

We combine the MF and auxiliary formulations for defining the transliteration-based translation score $\hat{x}_{f,e}^{trlit}$:

$$\hat{x}_{f,e}^{trlit} = \hat{x}_{f,e}^{mf} + \hat{x}_{f,e}^{aux}$$

Although transliteration does not always imply translation e.g., words can be transliterated to words that sound the same but have different meanings; our intuition is that transliter-

ations can be useful for translating words such as numbers or named entities. Therefore, we use our transliteration-based scores to add to BPR translation scores and re-rank its translation output when the source word is either a number, a single character, or a name. To infer when transliterations should be used for re-ranking, we use heuristics that look at the current BPR translation output.

Specifically, if the top-1 translation is a number or a single character, this may signal that the source word is also a number or a single character. In this case, since numbers are usually transliterated between languages and retain the same length (i.e., number of characters), we use transliterations-based translations that have the same length as the top-1 translation and add them to BPR output.

Secondly, if any of the transliteration-based translations has an edit distance of 1 or less to any of the BPR output, they may be good additional translations of the source word and we add them to BPR output. Finally, if more than a quarter of BPR output are names[3], this may signal that the source word is also a name. Since names are often transliterated between languages, we add all the transliteration-based translations to BPR output. Our method is summarized in Algorithm 1.

5.4.2. Data

We train a model to learn transliterations from Hindi (hi), Bengali (bn), Telugu (te), Tamil (ta), Nepali (ne), Ukranian (uk), Bulgarian (bg) and Siberian (sr) to English language as described in Chapter 4. We extracted the data using Wikipedia interlingual links to get the English Wikipedia pages for all the people who were born or died in years from 0 to 2017. We used the titles of these pages and the interlingual links (to access the language's Wikipedia) to generate the transliteration training data.

To determine if a candidate is a name, we extracted the list of English categories from DBpedia and Yago and checked if the candidate is a valid category in the list.

[3]We collect named entities in the target language from Wikipedia names and surnames categories in the language and YAGO knowledge base of Wikipedia named entities: `http://www.yago-knowledge.org/`

5.4.3. Results

We evaluated the system's performance using top-10 accuracy. The results for all the languages are presented in figure 5.4.3. As can be seen, the performance increases substantially for all the languages.

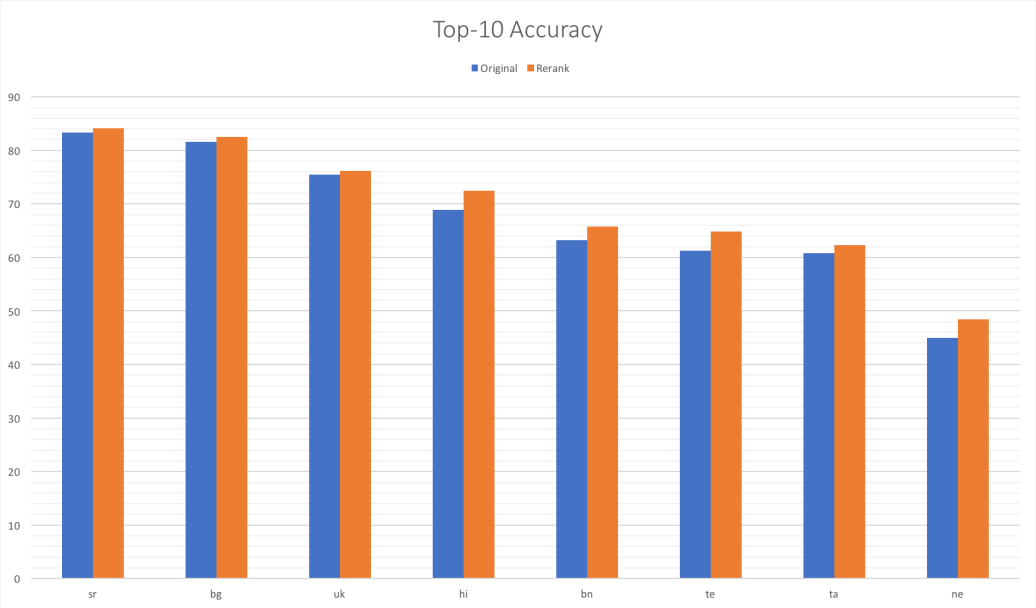

Figure 6: Top-10 accuracy for Hindi (hi), Bengali (bn), Telugu (te), Tamil (ta), Nepali (ne), Ukranian (uk), Bulgarian (bg) and Siberian (sr) to English Translation

Algorithm 1 Re-ranking BPR translations

Input: BPR top-10 translations for a source word f i.e., $\text{bpr}_{10}(f) = e_1...e_{10}$ with BPR scores $\hat{x}^{bpr}_{f,e_i}$ where $i = 1...10$, and transliteration-based translations for the word f i.e., $\text{trlit}(f) = g_1...g_K$ with scores $\hat{x}^{trlit}_{f,g_i}$ where $i = 1...K$, $K = |\text{trlit}(f)|$

Output: re-ranked BPR translations $\text{bpr}_{rr}(f) = h_1...h_{10}$ where $h_i \in \text{bpr}_{10}(f) \cup \text{trlit}(f)$ with scores $\hat{x}^{rr}_{f,h_i} = \hat{x}^{bpr}_{f,h_i} + \hat{x}^{trlit}_{f,h_i}$

Initialization: $\text{bpr}_{rr} \leftarrow \text{bpr}_{10}$

begin:

reranked $\leftarrow$ **false**

if e_1 is a number or a single character **do**:

    for $i = 1...K$ **do**:

        add g_i to bpr_{rr} **if** $|g_i| = |e_1|$

        reranked $\leftarrow$ **true**

    end for

else:

    for $i = 1...K$ **do**:

        if $\exists e \in \text{bpr}_{10}(f)$ s.t. $\text{editDist}(g_i, e) \leq 1$

        do:

            add g_i to bpr_{rr}

            reranked $\leftarrow$ **true**

        end if

    end for

end if

if $\neg$ reranked **do**:

    if $\frac{\#\text{names}(\text{bpr}_{10})}{10} \geq 0.25$ **do**:

        for $i = 1...K$ **do**:

            add g_i to bpr_{rr}

        end for

    end if

end if

sort bpr_{rr} and retain only the top-10

end

CHAPTER 6 : When to translate vs transliterate?

In this chapter, we discuss ways to identify when to transliterate a word in a sentence. Machine Translation techniques do not perform well to identify the correct transliterations vs translations and depend entirely on the phrase-table built using the training data. For example, if the English sentence is:

*He was the captain of the **national** championship team in 1982.*

*He was the chief of the **National** Basketball Association in 1990.*

A machine translation system should translate the word *national* in the first example, but, the same word should be transliterated in the second sentence. However, simply relying on the Named-Entity Recogniser to identify named-entites in a sentence and transliterating them does not work. This is because not all name-entities should be transliterated. For example,

*He was a resident of **Kolkata**, a city in **West Bengal**.*

Here, named entities are marked as bold. Although Kolkata and Bengal should be transliterated, but West is just a direction and should be transliterated instead. Another problem is sometimes words which are not part of any named entity become a candidate of transliteration. For example,

We ordered dal makhani, paneer tikka, rice and naan bread in the Indian restaurant.

The food items in the above sentence are not named entities but they should be transliterated while translating the sentence to some other language.

Hermjakob et al. (2008) considered this as a sequence tagging problem. They trained a averaged perceptron tagger using the *SEARN* algorithm over an annotated corpus. After that they proposed to add the transliteration to the phrase-translation tables of the SMT system based on the tagged output (determining whether the word should be transliterated

or not).

6.1. Methodology

We follow a similar approach to the one used by Hermjakob et al. (2008) to learn whether a word should be transliterated or not in a sentence. We describe it as follows:

1. We generate alignments for a Hindi-English parallel corpora

2. We created a test set using 120 sentences by manually annotating every word in the Hindi sentence with the label indicating whether its aligned English word is a transliteration or not.

3. From the remaining parallel aligned data, we generated the training and validation data set. We filter out the pairs in which either the Hindi or the English word is a stopword or a number or a single character. For the remaining pairs, we tag it as a transliteration pair if the edit distance between the Hindi word and the Hindi transliteration of the English word is less than or equal to 1 for top-10 transliterations. The final training data consists of approximately 340,000 sentences while the validation data consists of 30,000 sentences.

4. We extracted the features: context words in a window of size 5, their 2,3,4-grams and their prefixes and suffixes for the training, validation and testing data.

5. We trained a CRF classifier model [1] on the training data.

6.2. Data

We used the Hindi-English parallel training data Kunchukuttan et al. (2017), which consists of approximately 1.5 million sentences. We used Berekeley aligner [2] to create alignments between Hindi and English sentences.

[1] `https://taku910.github.io/crfpp/`
[2] `https://github.com/mhajiloo/berkeleyaligner`

To train the transliteration model, we used the data extracted using Wikipedia interlingual links as explained in chapter 4.

6.3. Results

We evaluate the system performance using the metrics: Precision, Recall and F1 score. The results are presented in the table 12.

Data Type	Precision	Recall	F-Score
Validation	79.5	77.5	78.4
Test	91.2	64.4	75.4

Table 12: Identifying when to transliterate a word in the sentence

CHAPTER 7 : Conclusion

In this study, we have explored the task of Machine Transliteration. We created a base transliteration system by treating it as a monotone machine translation task. We then explored ways to improve the system performance using additional data/techniques which may be helpful in a low resource setting.

We tried augmenting the training data using backlinks but it didn't improve the performance at all. We found that using languages written using same script to increase the training data improves the performance considerably. We have also shown that although using a 3rd language as intermediary during transliteration, the performance declines but F-score suggests that the transliterations are still intelligible. Clustering the data according to the source of origin can also help in improving system performance but the reduced dataset can affect that in an adversarial manner as well.

We also successfully demonstrated that machine transliteration could be used as signals for improving bi-lingual lexicon induction. We did this by following a heuristic-based approach and re-ranking the list of candidates output by the current matrix completion framework.

We also implemented a sequence tagger to identify the words that need to be transliterated in a sentence. This can be used as another component to improve the machine translation system's performance.

7.1. Future Work

Our method of clustering words based on the source of origin resulted in good performance boost. However, our data was collected from Wikipedia itself. In a real-world scenario or in a low-resource setting, such data may not be accessible. To deal with such situations, we can experiment with using other clustering techniques such as K-means.

Another useful study would be to explore into how to use machine transliteration in a machine translation system and study the impact on the performance. Another possible

area of application is Cross-Lingual Entity Linking, where the objective is to link the entities present in a document written in non-English to English knowledge base.

APPENDIX

BIBLIOGRAPHY

Huang, F. Cluster-specific Named Entity Transliteration. Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing. Stroudsburg, PA, USA, 2005; pp 435–442.

Wijaya, D. T.; Callahan, B.; Hewitt, J.; Gao, J.; Ling, X.; Apidianaki, M.; Callison-Burch, C. Learning Translations via Matrix Completion. Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017. 2017; pp 1452–1463.

Knight, K.; Graehl, J. *Comput. Linguist.* **1998**, *24*, 599–612.

Stalls, B. G.; Knight, K. Translating Names and Technical Terms in Arabic Text. Proceedings of the Workshop on Computational Approaches to Semitic Languages. Stroudsburg, PA, USA, 1998; pp 34–41.

Meng, H. M.; Lo, W.-K.; Chen, B.; Tang, K. Generating phonetic cognates to handle named entities in English-Chinese cross-language spoken document retrieval. Automatic Speech Recognition and Understanding, 2001. ASRU '01. IEEE Workshop on. 2001; pp 311–314.

Jung, S. Y.; Hong, S.; Paek, E. An English to Korean Transliteration Model of Extended Markov Window. Proceedings of the 18th Conference on Computational Linguistics - Volume 1. Stroudsburg, PA, USA, 2000; pp 383–389.

Virga, P.; Khudanpur, S. Transliteration of Proper Names in Cross-lingual Information Retrieval. Proceedings of the ACL 2003 Workshop on Multilingual and Mixed-language Named Entity Recognition - Volume 15. Stroudsburg, PA, USA, 2003; pp 57–64.

Al-Onaizan, Y.; Knight, K. Machine Transliteration of Names in Arabic Text. Proceedings of the ACL-02 Workshop on Computational Approaches to Semitic Languages. Stroudsburg, PA, USA, 2002; pp 1–13.

Kang, B.; Choi, K. Automatic Transliteration and Back-transliteration by Decision Tree Learning. Proceedings of the Second International Conference on Language Resources and Evaluation, LREC 2000, 31 May - June 2, 2000, Athens, Greece. 2000.

AbdulJaleel, N.; Larkey, L. S. English to Arabic Transliteration for Information Retrieval: A Statistical Approach. 2002.

Pingali, P.; Ganesh, S.; Yella, S. H.; Varma, V. Statistical Transliteration for Cross Language Information Retrieval using HMM alignment model and CRF. Third International Joint Conference on Natural Language Processing, IJCNLP 2008, Hyderabad, India, January 7-12, 2008. 2008; pp 42–47.

Haizhou, L.; Min, Z.; Jian, S. A Joint Source-channel Model for Machine Transliteration.

Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics. Stroudsburg, PA, USA, 2004.

Finch, A. M.; Liu, L.; Wang, X.; Sumita, E. Target-Bidirectional Neural Models for Machine Transliteration. Proceedings of the Sixth Named Entity Workshop, NEWS@ACL 2016, Berlin, Germany, August 12, 2016. 2016; pp 78–82.

Chinnakotla, M. K.; Damani, O. P. Character Sequence Modeling for Transliteration. 2009.

Kumaran, A.; Khapra, M. M.; Bhattacharyya, P. **2010**, *9*, 13:1–13:29.

Irvine, A.; Callison-Burch, C.; Klementiev, A. Transliterating from all languages. Proceedings of the Conference of the Association for Machine Translation in the Americas (AMTA). 2010; pp 100–110.

Li, Z.; Callison-Burch, C.; Dyer, C.; Khudanpur, S.; Schwartz, L.; Thornton, W.; Weese, J.; Zaidan, O. Joshua: An Open Source Toolkit for Parsing-Based Machine Translation. Proceedings of the Fourth Workshop on Statistical Machine Translation. Athens, Greece, 2009; pp 135–139.

Duan, X.; Banchs, R. E.; Zhang, M.; Li, H.; Kumaran, A. Report of NEWS 2016 Machine Transliteration Shared Task. Proceedings of the Sixth Named Entity Workshop, NEWS@ACL 2016, Berlin, Germany, August 12, 2016. 2016; pp 58–72.

Koren, Y.; Bell, R.; Volinsky, C. *Computer* **2009**, *42*.

Hermjakob, U.; Knight, K.; III, H. D. Name Translation in Statistical Machine Translation - Learning When to Transliterate. ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA. 2008; pp 389–397.

Kunchukuttan, A.; Mehta, P.; Bhattacharyya, P. *CoRR* **2017**, *abs/1710.02855*.