Statistical Models for Unsupervised, Semi-supervised and Supervised Transliteration Mining

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Hassan Sajjad

aus Lahore

Hauptberichter:	Prof. Dr. Hinrich Schütze
Mitberichter 1:	Prof. Dr. Alex Waibel
Mitberichter 2:	PD Dr. Helmut Schmid
Mitberichter 3:	Dr. Alexander Fraser
Tag der mündlichen Prüfung:	30. October 2012

Institut für maschinelle Sprachverarbeitung
der Universität Stuttgart

2012

Abstract

Transliteration is a process of converting a word written in one script to another script in such a way that pronunciation remains almost the same. It is useful in major applications of natural language processing such as machine translation and cross language information retrieval.

A transliteration system is generally built using two types of manually created resources – hand-crafted transliteration rules and a list of transliteration pairs. It either uses the transliteration rules with an edit distance metric to produce transliterations or automatically learns character alignments from transliteration pairs to build a model. The system requires language pair dependent resources for training which are not available for all language pairs.

Using transliteration mining, one can automatically extract a list of transliteration pairs from a parallel corpus. However, all the state-of-the-art transliteration mining techniques are supervised or semi-supervised and require language dependent information for training. Until the work described here was carried out, there was no fully unsupervised method in the literature.

In this thesis, I solve this issue by showing that transliteration mining can be done in an unsupervised fashion. The proposed method does not require any language pair dependent resources. I also incorporate transliteration into machine translation and word alignment and show that it improves the performance of the systems.

Following is the summary of the steps which I have gone through to accomplish this task:

In the first part of my work, I have shown the applicability of transliteration to machine translation. I have presented a novel machine translation model

that incorporates transliteration. During disambiguation, transliteration and translation options compete with each other and the decoder has to decide on the fly which translation or transliteration to choose. For closely related language pairs with significant vocabulary overlap, I showed that transliteration is effective for more than just translating out-of-vocabulary words.

I have proposed a heuristic-based transliteration mining system and showed that transliteration mining can be done in an unsupervised fashion. It shows competitive results when compared with the previous semi-supervised and supervised systems. This system has a few limitations. I then presented a novel model for unsupervised transliteration mining that consists of a transliteration sub-model and a non-transliteration sub-model. The unsupervised system performed better than most of the previous semi-supervised and supervised systems. I extended the unsupervised model to use the available resources and presented a semi-supervised and supervised version of it. I showed that if some labeled data is available, it is better to build a semi-supervised system than a supervised or unsupervised system.

I have incorporated unsupervised transliteration mining model to an unsupervised word aligner. The new alignment system is also fully unsupervised and showed a big improvement in both precision and recall when compared with the baseline alignment. This showed that the proposed unsupervised method of mining can be effectively used to improve the performance of natural language processing applications.

Zusammenfassung

Der Begriff "Transliteration" bezeichnet die Konvertierung eines Wortes aus einer Schrift in eine andere Schrift unter annähernder Beibehaltung der Aussprache. Transliteration ist für viele Anwendungen in der maschinellen Sprachverarbeitung nützlich, beispielsweise für die maschinelle Übersetzung oder für Cross-Language Information Retrieval.

Es gibt zwei wichtige Methoden, um Transliterationssysteme zu erstellen. Die erste Methode verwendet handgeschriebene Regeln und den String-Edit-Distance-Algorithmus, um die beste Transliteration zu bestimmen. Das andere Verfahren lernt ein statistisches Modell aus einer Liste von Transliterationspaaren. Beide Methoden erfordern sprachabhängige Ressourcen (hier Transliterationsregeln, dort Transliterationspaare), die manuell erstellt werden müssen und nur für wenige Sprachpaare verfügbar sind.

Mit Hilfe von Transliteration Mining ist es möglich, eine Liste von Transliterationspaaren automatisch aus einem parallelen Korpus zu extrahieren. Die besten bisherigen Transliteration-Mining-Methoden sind jedoch alle überwachte oder halbüberwachte Verfahren, die eine kleine Liste von Transliterationspaaren für das Training benötigen.

Die vorliegende Arbeit zeigt erstmals, dass Transliteration Mining auch mit einem rein unüberwacht trainierten System erfolgen kann. Die vorgestellte Methode erfordert keinerlei sprachabhängige Ressourcen. Ferner beschreibe ich, wie Transliteration zur Verbesserung der maschinellen Übersetzung und der Wortalignierung eingesetzt werden kann.

Im Folgenden fasse ich die Schritte zusammen, mit denen ich die Aufgabe gelöst habe:

Im ersten Teil meiner Arbeit zeige ich, dass Transliteration zur Verbesserung der maschinellen Übersetzung eingesetzt werden kann. Ich präsentiere ein neuartiges Übersetzungsmodell, in das ein Transliterationsmodul integriert ist. Während der maschinellen Übersetzung konkurrieren die Transliteration und die wortbasierte Übersetzung von Wörtern miteinander, und der Übersetzer entscheidet laufend zwischen der Verwendung der beiden Methoden. Ich zeige, dass bei der Übersetzung zwischen verwandten Sprachen mit einem großen gemeinsamen Wortschatz die Transliteration nicht nur für die Übersetzung unbekannter Wörter sondern auch bei anderen Wörtern nützlich ist.

Anschließend schlage ich ein heuristisches Transliteration-Mining-System vor und zeige damit, dass unüberwachtes Transliteration Mining möglich ist. Die Genauigkeit des Systems ist mit derjenigen von bisherigen halbüberwachten und überwachten Systemen vergleichbar. Das System hat jedoch einige Einschränkungen. Daher präsentiere ich eine weitere Methode für unüberwachtes Transliteration Mining, die zwei statistische Modelle kombiniert, eines für Transliterationen und eines für Nichttransliterationen. Dieses unüberwachte System ist genauer als die meisten bisherigen überwachten und halbüberwachten Systeme. Ich erweitere das unüberwachte System zu einem halbüberwachten System, um auch gegebenenfalls vorhandene manuell erstellte Transliterationspaare beim Training nutzen zu können. Ich stelle auch eine rein überwacht trainierte Variante meines Ansatzes vor. Wenn einige manuell annotierte Trainingsdaten zur Verfügung stehen, liefert das halbüberwachte System bessere Ergebnisse als das unüberwachte und das voll überwachte System.

Schließlich zeige ich, wie unüberwachtes Transliteration Mining die unüberwachte Wortalignierung verbessern kann. Das neue Wortalignierungssystem ist ebenfalls unüberwacht und wesentlich genauer als das Ausgangssystem, sowohl bzgl. Precision als auch bzgl. Recall. Damit wird gezeigt, dass die vorgestellte Transliteration-Mining-Methode die Leistung von Sprachverarbeitungssystemen effektiv verbessern kann.

Contents

List of Tables

List of Figures

1. Introduction

1.1. Transliteration

Languages often borrow words from other languages. One borrowing process is the conversion from one script to another script in such a way that pronunciation remains almost the same. This process is called *transliteration*. Words which are written differently and pronounced in the same way are transliterations of each other. Table 1.1 shows a few examples of English, Hindi and Urdu words. The English, Hindi and Urdu words are written in Latin, Devanagari and Arabic scripts respectively. However, they are pronounced in the same way and are transliterations of each other.

Transliteration is important to improve the quality of multilingual NLP tasks such as cross language information retrieval (CLIR) and machine translation. For instance, machine translation systems face the problem of translating out of vocabulary words (OOVs). OOVs are the words which do not occur in the training data. State-of-the-art systems fail to produce a translation of these words. The most trivial way to handle OOVs is to automatically transliterate unknown source words into target language script. This improves the quality of the machine translation output and makes it more readable and understandable (Durrani et al., 2010). Similarly cross language information retrieval systems are dependent on the translation lexicon extracted from a parallel corpus. Their performance drops when the query contains OOVs that are unknown to the lexicon. OOVs are often proper nouns and technical terms. They are crucial for the successful retrieval of an CLIR system as they are the most distinct terms in the query. Saravanan et al. (2010) addressed the problem of

English	Hindi	Urdu
October	अक्तूबर	اکتوبر
Attachment	अटैचमैंट	اٹیچمنٹ
Anwar	अनवर	انور
Apeal	अपील	اپیل
April	अप्रेल	اپریل
Ireland	आयरलैंड	آئرلینڈ
Order	आर्डर	آرڈر
Health	हैल्थ	ہیلتھ
Society	सोसायिटी	سوسائٹی

Figure 1.1.: Examples of transliteration

OOVs in CLIR by transliteration mining and transliteration generation and showed that transliteration is effective in improving the retrieval results.

There are various ways to build a transliteration system for a language pair. It requires the mapping of source language characters to target language characters in order to learn the relationship between the language pair. I later call these character mappings as *multigrams*. For a rule-based system, the multigrams are manually defined, requiring knowledge of both languages. Rule-based systems use multigrams together with the edit distance metric to generate transliterations of the source words. A model-based system learns multigrams automatically from a list of transliteration pairs. It uses the multigrams and their probabilities calculated from the training data to generate transliterations. Both of these methods require language dependent resources to build a system. Such resources are not readily available for all language pairs and creating them is expensive in terms of time and money.

1.2. Transliteration Mining

Given a list of word pairs containing transliterations and non-transliterations, the task of transliteration mining is to extract only transliteration pairs from the list. Kumaran et al. (2010) presented a shared task on transliteration mining which aims at automatically extracting a list of transliteration pairs from parallel corpora which can eventually be used to build a high quality transliteration system. This task is called NEWS10. It will be used throughout the thesis.

Transliteration mining can be done using rule-based methods, semi-supervised methods or supervised methods. Rule-based methods require transliteration rules/multigrams and an edit distance metric that assigns a score to every word pair in the list. Only pairs having a score below a certain threshold are considered as transliteration pairs.

Supervised transliteration mining systems require a list of transliteration pairs for training. They automatically learns multigrams from the training

data. The trained model is then applied to the word pairs list. The word pairs having a probability greater than a certain threshold are classified as transliteration pairs.

Similarly to supervised approaches, the semi-supervised system also requires a list of transliteration pairs. However, here the list is generally small. So the system does not fully rely on the training data to mine transliteration pairs. The system uses training data for initial training and scores every word pair from the list of word pairs (similar to the supervised system). Then it extracts a few word pairs from the list of word pairs which are most likely to be transliterations and adds them to the training data. It again builds a transliteration mining system on the new training data and the procedure is iterated, adding few more transliteration pairs to the training data. The system stops when there are no more transliteration pairs in the list of word pairs.

All the above described transliteration mining methods are dependent on resources. Rule-based systems requires hand-crafted transliteration rules (multigrams). These rules are not available for all language pairs and building these rules is a tedious task. Supervised and semi-supervised transliteration mining systems require a list of transliteration pairs which is also not available for all language pairs. The NEWS10 shared task provides a small list of transliteration pairs for initial training. So, all systems that participated at the shared task are semi-supervised or supervised (Kumaran et al., 2010). To the best of my knowledge, prior to my work, there was no fully unsupervised transliteration mining system that uses only the list of word pairs as training data and does not require any pre-existing resources.

In this thesis, I present the first fully unsupervised transliteration mining system. The system learns multigrams from the list of word pairs and requires neither hand-crafted transliteration rules nor a list of transliteration pairs. I evaluate it on parallel corpora and show that it has competitive results to semi-supervised and supervised systems. Later, I present a novel model for unsupervised transliteration mining. This system is more efficient than my previous system and outperforms most of the semi-supervised and supervised systems which participated in NEWS10.

1.3. Contributions

In this section, I present contributions that I will make in my dissertation. I divide them into theoretical and resource contributions. They are described as follows:

1.3.1. Theoretical Contributions

Following are my theoretical contributions:

- **Transliteration in machine translation:** I present a novel model which combines translation and transliteration models into a single MT model. It considers both translation and transliteration when translating a particular source word given the context. For closely related languages with significant vocabulary overlap, I show that transliteration is helpful for more than just translating out-of-vocabulary words and named entities. I use it as a tool for disambiguation of homonyms which can be translated or transliterated or transliterated differently based on different contexts.

- **The first fully unsupervised algorithm for transliteration mining:** I present the first unsupervised algorithm for transliteration mining. It is an iterative algorithm that trains on unlabeled data. In every iteration, it filters out a few word pairs from the unlabeled data which are least likely to be transliterations and retrains on the modified unlabeled data. The process is iterated until the unlabeled data contains all transliteration pairs. The unsupervised mining system shows competitive results with the state-of-the-art semi-supervised and supervised transliteration mining systems.

- **A novel model for unsupervised transliteration mining:** I propose a novel model to automatically extract transliteration pairs from parallel corpora. My model is very efficient and language pair independent. It models unlabeled data which consists of transliterations and

non-transliterations. I define it as an interpolation of a transliteration sub-model and a non-transliteration sub-model. The results show that it is better than most of the semi-supervised and supervised transliteration mining systems.

A framework for transliteration mining: I show that the model for unsupervised transliteration mining can easily be extended and generalized. It can be used as a common framework for both semi-supervised and supervised learning, when labeled data is available. This is the first framework that uses only one model for all three kinds of learning.

- **An application to word alignment:** I incorporate my unsupervised transliteration mining system into an unsupervised word alignment system. The resulting word alignment system is also fully unsupervised. It shows a large gain in both precision and recall over the baseline alignment system when compared with gold standard alignments.

1.3.2. Resource Contributions

The following are my resource contributions:

- **Gold standard for transliteration mining:** I evaluate my transliteration mining systems using English/Hindi and English/Arabic parallel corpora. This is the first evaluation of transliteration mining on this dataset. I built a gold standard dataset by annotating a subset of the training data. The annotation task consists of disambiguating word pairs as either transliterations or non-transliterations. This gold standard is freely available to the research community. It will be useful for future system's evaluations.

- **A transliteration mining tool:** I implement the transliteration mining system. The system has four modules - character aligner, unsupervised transliteration mining, semi-supervised transliteration mining and supervised transliteration mining. The system with its source code is freely available to the research community.

1.4. Outline of the Dissertation

I have shown that a list of transliteration pairs is required to build a transliteration system which can further be used in many major applications of NLP. In this section, I present the flow of my dissertation describing the motivation of my work, the way I solved the transliteration mining problem and its application to an NLP task.

Chapter 2 is divided into four parts. First, I describe the literature of machine translation systems that use transliteration information for OOVs and named entities. Later, I describe various models and alignment methods that are used in the literature for the character alignment of transliteration pairs. Then, I summarize the previous work on transliteration. In the last part of this chapter, I present the previous work on semi-supervised and supervised transliteration mining (there is no unsupervised system in the literature).

I consider the fact of choosing a transliteration based on context and present a novel model which incorporates transliteration information at decoding time. For every word, the translation and transliteration compete with each other on the fly and decoder decides which translation or transliteration to choose. The system shows an improvement in translation quality when compared with the baseline systems. This shows that transliteration is helpful for machine translation systems in more than just transliterating out-of-vocabulary items. Chapter 3 describes the system and its evaluation.

Chapter 4 presents the unsupervised transliteration mining algorithm. The algorithm uses no form of supervision, and does not require linguistically informed preprocessing. I compare it with semi-supervised and supervised systems and show that it has competitive results.

Chapter 5 proposes a novel model to automatically extract transliteration pairs from parallel corpora. I model transliteration mining as an interpolation

of transliteration and non-transliteration sub-models. The model is efficient, language pair independent and mines transliteration pairs in a consistent fashion in all unsupervised, semi-supervised and supervised settings. In evaluation, I show that my model-based unsupervised system outperforms most of the state-of-the-art semi-supervised and supervised systems.

In Chapter 6, I integrate a transliteration module to unsupervised word alignment and show that it improves word alignment quality. The transliteration module is trained on the transliteration pairs which my transliteration mining method (presented in Chapter 5) extracts from parallel corpora. The new word alignment system with transliteration module is also unsupervised.

Chapter 7 describes future work and concludes the dissertation.

2. Background

2.1. Introduction

I divide the background work into three parts.

First, I lay down the foundation of statistical transliteration and transliteration mining. Both take labeled data (a list of transliteration pairs) and character align it in training. Several tools have been used in the literature for character alignment. The commonly used tools are based on either a conditional probability model or a joint probability model. Section 2.2 presents an overview of them.

In Section 2.3, I describe the previous work on building transliteration systems. Heuristic-based systems use a list of hand-crafted transliteration rules with an edit distance metric for transliteration generation. The substitution, insertion and deletion costs need to be manually adjusted. Heuristic-based methods are language dependent and require a lot of manual effort to build transliteration rules. Statistical transliteration systems avoid this manual effort by using a list of transliteration pairs to automatically learn transliteration rules and their probabilities. In Section 2.3, I focus on statistical approaches to transliteration.

The third and last part of the literature review discusses mining of transliteration pairs from parallel corpora. Simple heuristic-based systems use the edit distance metric with hand-crafted transliteration rules to score a list of candidates. Word pairs with edit distance less than a threshold are chosen as transliteration pairs. Statistical transliteration mining methods use a list of transliteration pairs to train their model. There are generally two approaches

to statistical transliteration mining – generative and discriminative. Generative models use positive labeled examples (a list of transliteration pairs) while discriminative approaches require positive and negative labeled examples for training. The major limitation of previous transliteration mining techniques is that they require labeled data for training. Section 2.4 summarizes the previous work on transliteration mining. Section 2.5 summarizes this chapter.

2.2. Transliteration Foundation

A transliteration system converts a string written in one script to another script. It uses either a list of hand-crafted transliteration rules or is trained on a list of transliteration pairs. The development of hand-crafted transliteration rules is a tedious task. Section 2.3.1 talks about it in detail. Here, I focus on the transliteration systems that use transliteration pairs for training.

A list of transliteration pairs consists of source and target language strings that are transliterations of each other. A few examples of English/Hindi/Urdu transliterations are shown in Figure 1.1. A statistical transliteration system aligns the list of transliteration pairs at character level. It extracts the character mappings with their probabilities from the aligned pairs to build a list of transliteration rules. In test mode, given a test word, the system generates its transliteration based on the probabilities of the transliteration rules learned from the training data.

A transliteration mining system on the other hand automatically extracts transliteration pairs from a list of word pairs. Similar to the transliteration system, it is trained on a list of transliteration pairs. At test time, the trained model classifies whether a given word pair is a transliteration pair or not.

In this section, I present the foundations of building transliteration systems and transliteration mining systems. Both require the alignment of transliteration pairs. As in machine translation, the alignment plays a vital role in the performance of the transliteration (mining) systems. In transliteration and transliteration mining, alignment is easier than in machine translation as it

One-to-one	Source		b	c	d	p	f
	Target		X	C	D	∅	F
One-to-one	Source	∅	b	c	d	p	f
	Target	X	∅	C	D	∅	F
Many-to-many	Source		b	cd		pf	
	Target		X	CD		F	

Table 2.1.: Example of one-to-one and many-to-many character alignments of a word pair *bcdpf XCDF* where ∅ represents a null character

does not involve reordering. In the following section, I first describe the commonly used models and then describe various tools used for this purpose.

2.2.1. Alignment

Consider a transliteration pair (e, f) where $e_1^J = e_1, ..., e_j, ..., e_J$ is the source string and $f_1^I = f_1, ..., f_i, ..., f_I$ is its transliteration. J and I are the number of characters in e and f respectively. A transliteration pair can be aligned in more than one way. Table 2.1 shows a few possible alignments of the word pair (*bcdpf XCDF*). The first row is an example of a one-to-one alignment which means that a source character is aligned to only 0 or 1 target character and the same is true for the target language characters. In many-to-many alignment, there can be more than one characters of source and target that can be aligned to each other like the alignment subsequence $cd \rightarrow CD$ and $pf \rightarrow F$.

I call alignment subsequences *multigrams*, often referring to a particular multigram using the variable q, later on. $cd \rightarrow CD$, $pf \rightarrow F$, $b \rightarrow X$ and $\emptyset \rightarrow X$ are examples of multigrams. In Section 2.2.4, I describe alignment methods in detail.

2.2.2. Joint Probability Model

The joint probability of the word pair (e, f) is the sum over all alignment sequences:

$$p(e, f) = \sum_{a \in Align(e,f)} p(a)$$

where $Align(e, f)$ is the set of all possible alignments of a word pair, a is one alignment sequence and $p(a)$ is the probability of that sequence. Consider an alignment sequence $(q_1, q_2, ..., q_{|a|})$ where q is the multigram as defined in Section 2.2.1 and $|a|$ is the length of the alignment sequence. The probability of an alignment sequence is the product of the probabilities of all multigrams it contains:

$$p(a) = p(q_1, q_2, ..., q_{|a|}) = \prod_{j=1}^{|a|} p(q_j)$$

The above equation assumes that the multigrams are independent of each other. It can be modified to consider the preceding context:

$$p(e, f) = \prod_{j=1}^{|a|} p(q_j | q_{j-M+1}^{j-1})$$

where M is the ngram size.

2.2.3. Conditional Probability Model

The conditional probability distribution $p(e|f)$ models the relationship between the values of two random variables. It is defined as:

$$p(e|f) = \frac{p(e, f)}{p(f)}$$

The conditional probability captures how the source string can be generated from the target string. The joint model models how the source and target

strings can be generated together. Li et al. (2004) present a comparison of these two models for transliteration and show that the joint probability model performs better than the conditional model. In my transliteration mining model, I use the joint probability model.

2.2.4. Alignment Methods

In this section, I provide a summary of various alignment methods that are used in transliteration systems and transliteration mining systems for the character alignment of labeled data.

GIZA++

GIZA++ is a word alignment tool which can be used for the character alignment of transliteration pairs. Normally, GIZA++ is used to align words in sentences. In transliteration, word pairs are treated as parallel sentences. By putting a space after each character of a word, every character can be treated as if it were a word in word alignment.

GIZA++ is based on a conditional probability model which captures how a target string can be generated from a source string. It builds two conditional probability models, one in each direction – from source to target and from target to source. The Viterbi alignments of both directions of a string pair are combined using several heuristics to generate a many-to-many alignment sequence (Och and Ney, 2003).

Consider a source string $e_1^J = e_1, ..., e_j, ..., e_J$ and its corresponding target string $f_1^I = f_1, ..., f_i, ..., f_I$. Alignment models tend to define the relationship between source and target strings with a hidden alignment variable a_1^J which is the mapping from a source position J to a target position a_j. The probability of a source string given the target string is the sum over all its possible alignments:

$$P(e_1^J|f_1^I) = \sum_{a_1^J} p_\theta(e_1^J, a_1^J|f_1^I)$$

θ represents the unknown parameters. Their values are determined using expectation maximization (EM) training that maximizes the likelihood of the training data. For a given string pair, the goal is to find the best alignment which is called the Viterbi alignment of the string pair (e_1^J, f_1^I):

$$\hat{a}_1^J = \underset{a_1^J}{\operatorname{argmax}}\, p_{\hat{\theta}}(e_1^J, a_1^J | f_1^I)$$

GIZA++ has an option of using a combination of five IBM models and an HMM. All models contain a lexicon (for transliteration, it is a list of character pairs) and a set of parameters to describe the probability of a character pair. They differ in the decomposition of $(e_1^J, a_1^J | f_1^I)$. Here I briefly describe the difference between IBM Models. For details, see Och and Ney (2000, 2003); Koehn (2010). I describe the HMM in Section 2.2.4.

Model 1 is a uniform probability model. The probability of an alignment of a string pair depends on the lexical probability $p(e_j | f_{a_j})$ so the character order does not affect the alignment probability (Och and Ney, 2003):

$$p(e_1^J, a_1^J | f_1^I) = \frac{p(J|I)}{(I+1)^J} \cdot \prod_{j=1}^{J} p(e_j | f_{a_j}) \tag{2.1}$$

In contrast to Model 1, Model 2 explicitly models the position of source and target characters. However, the alignment probabilities are independent from their neighboring alignments:

$$p(e_1^J, a_1^J | f_1^I) = p(J|I) \cdot \prod_{j=1}^{J} [p(a_j | j, I, J) p(e_j | f_{a_j})] \tag{2.2}$$

Model 1 and Model 2 have a limitation of not modeling the fertility which is the number of target characters aligned to a source character e_j. Model 3 adds a fertility model $p(\phi | f)$.

Model 4 introduces a relative distortion model which exploits the fact that characters that are close to each other tend to stay next to each other in the output. So, it models the alignment of an input character dependent on the alignment of the preceding input character.

Model 3 and Model 4 are deficient alignment models. They have the problem that multiple output characters can be placed at the same position according to the model. Model 5 fixes this problem by introducing a variable that keeps a record of occupied places.

To summarize, a major difference between the models is the contextual information. The HMM model, IBM Model 4 and Model 5 are first order models whereas Model 1, Model 2 and Model 3 are zero order models.

The character alignment of a transliteration pair is easier than the word alignment of a parallel sentence as it does not involve reordering. For transliteration pairs, GIZA++ is used with the assumption that the EM training would learn from the monotonic nature of the training data and would give low probability to the alignments that involve reordering.

I did not use GIZA++ for alignment of word pairs in unsupervised transliteration mining because of the training data which is unlabeled. Most part of the unlabeled data consists of noise and there are only a few transliteration pairs. GIZA++ would learn wrong multigrams from the noisy pairs and would not work in the context of transliteration mining.

Hidden Markov Model-based Aligner

The Hidden Markov Model (HMM) is composed of a set of states which are interconnected through transitions. Every state has an emission probability distribution and a transition probability distribution. The emission probability is the probability of emitting the observed symbol. The transition probability is the probability of moving from the current state to another state (Rabiner, 1990).

The states are hidden in an HMM model. For transliteration pairs, they represent character alignments. The Expectation Maximization algorithm can be used to learn the model parameters from unaligned data. The Viterbi algorithm is then used to find the best alignment sequence for the data.

The Hidden Markov Model aligner, as opposed to IBM Model 1, 2 and 3, is a first order model so the alignment position a_j is dependent on the previous

alignment position a_{j-1}. The alignment model is given by (Och and Ney, 2003):

$$P(e_1^J, a_1^J | f_1^I) = P(J|f_1^I) \prod_{j=1}^{J} P(e_j, a_j | e_1^{j-1}, a_1^{j-1}, f_1^I)$$

$P(J|f_1^I)$ is the length probability. $P(e_j, a_j | e_1^{j-1}, a_1^{j-1}, f_1^I)$ can be written as a product of an alignment probability $P(a_j | e_1^{j-1}, a_1^{j-1}, f_1^I)$ and a lexical probability $P(e_j | e_1^{j-1}, a_1^j, f_1^I)$.

The above equation is a general model of alignment. We may modify it by applying a few assumptions. The alignment probability is only dependent on its previous alignment. The lexical probability is independent of context. The equation for the HMM is given by:

$$P(e_1^J, a_1^J | f_1^I) = P(J|I) \sum_{a_1^j} \prod_{j=1}^{J} [p(a_j | a_{j-1}, I).p(e_j | f_{a_j})]$$

Due to the first-order assumption, the HMM performs better than Model 1 and Model 2. The HMM alignment is sensitive to the value of initial parameters.

Pair Hidden Markov Model for Alignment

A Pair Hidden Markov Model (PHMM) is an extension of the Hidden Markov Model that generates two observation sequences in parallel. It is first introduced by Durbin et al. (1998) for aligning biological sequences. Mackay and Kondrak (2005) modify their model to use it for string similarity to identify cognates. A PHMM alignment model contains three states described as follows:

- Substitution state (S): matches the source character sequence with the target character sequence. The emission probability is the probability of emitting a pair of characters (x,y) where x is the source language character and y is the target language character

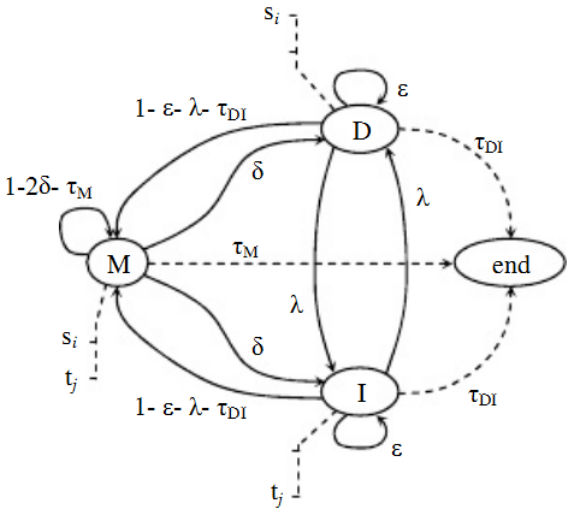


Figure 2.1.: PHMM for transliteration

- Deletion state (D): matches the source character sequence with a gap on the target language side. The emission probability is defined as p(x, -) i.e. probability of aligning source language character x to NULL
- Insertion state (I): matches a gap on the source character sequence with the target language character sequence. The emission probability p(- , y) is defined as the probability of aligning target language character y to NULL

The standard PHMM model has three transition parameters: δ (for transition from the match state M to a gap state), ϵ (for transition of staying on a gap state) and τ (for transition to the end state). It keeps the transition probability τ to the end state equal for all other states. Mackay and Kondrak (2005) separate τ to τ_S for match state, and τ_{DI} for the gap states. This would help the model to capture end word operations which are important in the identification of cognates. The PHMM of Mackay and Kondrak (2005) is shown in Figure 2.1.

The standard PHMM model assumes that a deletion operation followed by an insertion operation is equal to a substitution operation. This is not true for transliteration where a substitution operation is only applied for a legal character alignment between a source and a target character sequence. To fix this, Mackay and Kondrak (2005) introduce a pair of transitions between state D and I. The transition parameter is denoted by λ. The parameters can be learned using EM.

PHMM is similar to the joint sequence model used in g2p (see Section 2.2.4) except that there are only three states. They are stronger than the zero-order joint sequence models (one state) but are weaker than the first-order joint sequence models (one state per multigram).

Phrase-based Alignment

A phrase-based alignment system learns alignment of large transliteration units. Generally, a many-to-many character alignment is generated using an alignment tool such as the HMM aligner or GIZA++. The phrase extraction algorithm is then applied to the many-to-many alignment to extract larger units. The details of the phrase extraction algorithm are described in Koehn (2010). In the context of transliteration, phrase level alignments are important in aligning non-alphabetic languages like English/Japanese and Japanese/Chinese. These language pairs require many-to-one/one-to-many/many-to-many character mappings to learn correct transliteration rules. The larger phrases also manage insertions and deletions in a better way.

In the Moses toolkit, GIZA++ is used to produce an initial alignment for the phrase extraction algorithm. GIZA++ produces a one-to-many and a many-to-one alignment. A set of heuristics are applied on the two alignments to extract reliable many-to-many alignments. The phrase extraction algorithm is then applied to them. I use phrase-based alignment to build a transliteration system in Chapter 4. In the context of unsupervised transliteration mining, the phrase-based alignment tends to learn noise and the system performs poorly. Section 5.7.5 presents results of my higher order unsupervised transliteration

mining system. The system learns noise by memorizing larger units and does not perform well.

Grapheme-to-Phoneme Converter g2p

Grapheme-to-phoneme conversion is a similar task to transliteration. g2p is a grapheme-to-phoneme conversion tool (Bisani and Ney, 2008). It is based on a joint sequence model as presented in Section 2.2.2. The system uses expectation maximization (EM) to learn many-to-many character alignments from the training data. I use it in my unsupervised transliteration mining algorithm as a transliterator (see Chapter 4). However, I restrict the transliteration units to 0-1,1-1,1-0 character alignments as in the PHMM described in Section 2.2.4. For unsupervised transliteration mining, larger transliteration units tended to learn noise.

M2M-aligner

Ristad and Yianilos (1998) proposed a stochastic string edit distance based method to learn character alignments from transliteration pairs. The model is presented as a memoryless stochastic transducer. It automatically learns the substitution, insertion and deletion costs from the training data. Ristad and Yianilos (1998) calculate two alignment scores from the transducer. The first one is the Viterbi score – the score of the most likely alignment between the two strings. The second score is the sum of the scores of all alignments of a given string pair which is called stochastic edit distance. Sherif and Kondrak (2007b) use the string edit distance based method of Ristad and Yianilos (1998) in the transliteration extraction system which is explained in Section 2.4.

The model of Ristad and Yianilos (1998) is equivalent to my unigram model presented in Chapter 5. My system differs in training where I model the data using two sub-models – transliteration and non-transliteration. This is important when modeling unlabeled data that contains both transliterations and non-transliterations.

The stochastic transducer model produces one-to-one alignments. Jiampojamarn et al. (2007) extend it to infer many-to-many alignments of grapheme to phoneme pairs. They present an algorithm that modifies the Forward-Backward training of the one-to-one aligner to calculate the partial counts of subsequence alignments. The probabilities are calculated by normalizing the partial counts. They produce the most likely alignment of a word pair using the Viterbi algorithm.

I use one-to-one character alignment for unsupervised transliteration mining. The many-to-many alignment does not work for unsupervised transliteration mining (see chapter on transliteration mining model, Section 5.7.5).

2.3. Previous Work on Transliteration

Transliteration can be defined as a process of converting the text written in one script to another script. The pronunciation of the text generally remains the same. A transliteration system requires two main modules: mapping table and transliteration. The former is a list of multigrams that can be built either manually or automatically using the alignment methods described in Section 2.2.4. In the following sections, I first discuss the work on the extraction of multigrams. Later, I talk about several methods that use these units to build a transliteration system.

I build two transliteration systems – one using a joint sequence model and another using a phrase-based machine transliteration system. The former is used with a machine translation system in Chapter 3 and the latter is used in unsupervised transliteration mining in Chapter 4.

2.3.1. Extraction of Multigrams

A mapping table between source and target language characters can be produced by either manually creating hand-crafted transliteration rules/multigrams (Tao et al., 2006; Sajjad et al., 2011a) or by automatically character aligning the transliteration pairs (Kashani et al., 2007b; Noeman and Madkour, 2010).

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 2.2.: Ambiguous Hindi characters (characters which can transliterate to many different Urdu characters)

The manual generation of multigrams is a complex task and requires a lot of effort. A source language character may map to 0, 1 or many target language characters and the same holds for target language characters as well. Also there can be more source characters that may map to the same target character and vice versa. Consider the example of the Hindi and Urdu language. They have similar sound systems but transliteration from Hindi to Urdu is still hard because some phonemes in Hindi have several orthographic equivalents in Urdu. Table 2.2 shows examples of ambiguous Hindi/Urdu transliteration units. The "z" sound can only be written as ज़ whenever it occurs in a Hindi word but can be written as ذ, ز, ض and ظ in an Urdu word. Transliteration becomes non-trivial in cases where multiple orthographic equivalents for a Hindi word are valid Urdu words. In order to build an edit distance based transliteration system on the hand-crafted multigrams, one needs to define the cost of choosing a multigram.

In Section 3.5.1, I extracted a list of character aligned transliteration pairs using hand-crafted multigrams and an edit distance based metric (Sajjad et al., 2011a) and used it to build a statistical transliteration system. The system shows high accuracy but is not robust and is unable to produce complex transliterations with several ambiguous character transliterations. Secondly, the multigrams built for one language pair can not be used for other language pairs. In this section, I describe the previous work of automatic extraction of multigrams from a list of transliteration pairs. The transliteration models that

are trained on the transliteration data are discussed in Section 2.3.2.

Kashani et al. (2007b) use GIZA++ to learn multigrams from a list of transliteration pairs. GIZA++ generates non-monotonic alignments for a few word pairs. They use this effect to filter out bad transliteration pairs from the list of transliteration pairs. Rama and Gali (2009); Shishtla et al. (2009); Tiedemann and Nabende (2009) also use GIZA++ for the alignment of transliteration pairs. Rama and Gali (2009) apply a post-processing step to handle non-monotonic alignments.

Knight and Graehl (1998) use a conditional probability model for back transliteration. They use a series of models which step-by-step convert the Japanese string to an English string. In this way, the output of the first model is the input of the second model and so on. Stalls and Knight (1998) modify their generative process and use it for back transliteration from Arabic to English. These systems are explained later.

Noeman (2009) uses the HMM aligner with the Forward-Backward algorithm to align transliteration pairs. The character alignments are further extended to character sequence level using various heuristics (similar to the one used in the phrase based machine translation system (Koehn et al., 2003)).

Li et al. (2004) present a joint probability model for transliteration. It is equivalent to an HMM whose states correspond to pairs of input and output symbols and whose emission probabilities are one for the symbol pair encoded in the state and zero for all other symbols pair. They apply Expectation Maximization (EM) to learn multigrams from the training data. Ekbal et al. (2006) also use a joint probability model for transliterating English/Bengali. In contrast to Li et al. (2004), they use linguistic knowledge to extract transliteration units from an English/Bengali transliteration corpus.

Nabende (2009) trains a transliteration system using the PHMM alignment model and shows that the model works better than one trained using a weighted Finite State Transducer.

Finch and Sumita (2008) use phrase-based alignment for both transliteration and back transliteration of English/Japanese. Rama and Gali (2009); Noeman and Madkour (2010) show that phrase-based alignment combined with a

phrase-based statistical machine translation system can be successfully used for machine transliteration. I also build a phrase-based transliteration system in Chapter 4. The phrase-based alignment is normally used for the alignment of parallel sentences which involves several complex steps such as reordering. They do not exist in the alignment of transliteration pairs. The phrase-based alignment combined with phrase-based decoding is an approximation of the true transliteration system with the assumption that the system will not learn the phenomena which are not prevalent in the data.

Jiampojamarn et al. (2009) and Ammar et al. (2012) use the M2M-aligner for the alignment of transliteration pairs.

I did not use a statistical alignment method in aligning the Hindi/Urdu list of transliteration pairs in Chapter 3. Instead the list is aligned from a parallel corpus using a list of multigrams and an edit distance metric.

2.3.2. Transliteration Methods

The multigrams extracted from the transliteration pairs are used in various models to build a transliteration system. This section summarizes these methods.

Knight and Graehl (1998) present a phonetic-based system for back transliteration. They divide the transliteration process into five models. The output of one model is the input of the next model and so on. The phonetic-based transliteration method involves mapping the source language words to their phonemic representation and converting the phonemic form to the target language script.

The process of transliterating Japanese to English is divided into the following five generative steps:

- $p(w)$ – probability of generating English word sequences
- $p(e|w)$ – given an English word sequence, produce its phonetic transliteration which maximizes $p(e|w)$

- $p(j|e)$ – probability of producing Japanese sound sequences given English sound sequences
- $p(k|j)$ – producing katakana script from Japanese sounds (in Japanese, most foreign words are written in katakana)
- $p(o|k)$ – katakana is written which is extracted by optical character recognition (OCR)

Knight and Graehl (1998) implement the conditional probability models using a Finite State Transducer, (FST) and the unigram word based model $p(w)$ using a Finite State Acceptor (FSA). The best transliterations are extracted using Dijkstra's shortest path algorithm and k-shortest-path algorithm.

Stalls and Knight (1998) modify the generative model proposed by Knight and Graehl (1998) for back transliteration from Arabic into English. The problem of Arabic to English transliteration is more challenging as short vowels are not written in Arabic. The true phonemic representation of Arabic words is hidden so it is difficult to generate the correct Arabic phonemic mapping from the English phonemic mapping. They build a new model $p(a|e)$ to generate Arabic letter sequences directly from English phonemic sequences. The first two models of Knight and Graehl (1998) – $p(w)$ probability of an English word and $p(e|w)$ probability of producing an English pronunciation given an English word, are used without any change. The probabilities are learned from an Arabic/English dictionary of 150 word pairs. The system is tested on 2800 Arabic names. It generates the correct transliteration of 900 words and fails to correctly transliterate the rest. The error analysis shows that the transliteration of foreign words into Arabic is dependent on their language of origin. It is difficult to correctly back transliterate them in English.

The phonetic-based systems can only be applied to words whose pronunciation is known. The conversion of grapheme to phoneme adds an additional layer of vulnerability. Al-Onaizan and Knight (2002) compare a grapheme-based transliteration model with a phonetic-based transliteration model and show that the grapheme-based method performs better than the phonetic-

based method. In this chapter, I will focus on grapheme-based methods. Later in Chapter 3 and Chapter 4, I present two grapheme-based transliteration systems.

Kashani et al. (2007b); Shishtla et al. (2009); Tiedemann and Nabende (2009) and Rama and Gali (2009) present grapheme-based methods for transliteration. They all use GIZA++ for the alignment of transliteration pairs. However, they differ in the transliteration method. Kashani et al. (2007b) use two generative steps, followed by a comparative step to find the transliteration of a word. The first step takes the test corpus as input and outputs k candidates for each Arabic word. The k candidates are then used in step 2 to generate another k candidates. The difference between step 1 and step 2 is only in handling the English characters aligned to null. In the third step, the Google unigram dataset was used to filter out non-English words. They use edit distance to calculate the similarity between dictionary entries and the candidates and to filter names that have minor errors. The Viterbi score of step 1 and 2, and Levenshtein distance of step 3 is combined to select the right candidate.

The transliteration system of Rama and Gali (2009) is different from others which used GIZA++ for alignment in the decoding step where they use a beam search algorithm. Noeman (2009) also uses a monotone beam search decoder to generate k best transliterations of a source word. All candidates are then ranked on their transliteration model and monolingual language model probabilities. In my ngram-based transliteration system, I use the Viterbi algorithm to find the best transliteration of the source word. My phrase-based transliteration system uses the beam search algorithm in decoding.

Shishtla et al. (2009) align transliteration pairs using GIZA++ and Ammar et al. (2012) use the M2M-aligner for that. They both use the Conditional Random Field (CRF) framework for transliteration generation. The character alignment is transformed to a sequence by assigning each source language character to a label which is a sequence of one or more target language characters. Ammar et al. (2012) use the size of the label, unigram and bigram labels and unigram, bigram and trigram context as features for the model.

The phrase-based statistical machine translation system (PSMT) (Koehn

et al., 2003) is often used for transliteration generation. Noeman and Madkour (2010) use the PSMT system to build a substring based transliteration system. Tiedemann and Nabende (2009) align the transliteration pairs using GIZA++ and use PSMT with monotonic decoding for generation. They compare the PSMT system with a WFST system and show that PSMT performs better. I also built a phrase-based system for transliteration in Section 4.3.2. The major advantage of PSMT is the phrase table which contains larger transliteration units. The contextual information helps to generate better transliterations.

Jiampojamarn et al. (2009) present a discriminative transliteration system, DirecTL. The transliteration pairs are aligned at substring level by using EM. Every word from the training data is represented as a vector of features – ngram context, HMM-like transition features and linear-chain features. An ngram context feature contains information of neighboring letters of a source character when it is aligned to a target character. The HMM like transition feature acts as a language model to restrict the production of non-target language words. They use first order markov assumption which is a bigram feature vector. The linear chain features are like the ngram context features except that the single target language character is replaced with a bigram sequence of characters. A linear model is trained in the feature space using iterative training. Each iteration produces the n most likely output words for an input word. The parameter vector of an iteration is updated by comparing the correct output and the n best output using the margin infused relaxed algorithm (MIRA). The training is iterated using the new values of the parameters.

2.4. Previous Work on Transliteration Mining

The transliteration mining systems use the same alignment model as transliteration systems. The alignment methods are described in Section 2.2.4. In the following section, I provide a summary of the systems used for the mining of transliteration pairs from parallel corpora. I divide them into generative methods (Noeman and Madkour, 2010; Darwish, 2010; Nabende, 2010) and

discriminative methods (Jiampojamarn et al., 2010). All systems in the literature (that I am aware of) use manually labeled (seed) data for initial training or for the optimization of parameters so they are either supervised or semi-supervised. There is no unsupervised system. A lot of work has been done on discovering and learning transliterations from comparable corpora by using temporal and phonetic information (Tao et al., 2006; Klementiev and Roth, 2006; Sproat et al., 2006). I do not have access to such data and will not discuss these approaches here. My focus is on extracting transliteration pairs from parallel corpora.

In Chapter 4, I present a heuristic-based unsupervised mining system based on a joint probability model. It is an iterative algorithm that filters out a few word pairs from unlabeled training data in every iteration until the training data contains only transliteration pairs. In Chapter 5, I propose a novel model for unsupervised transliteration mining which consists of a transliteration model and a non-transliteration model. In the following section, I summarize the previous transliteration mining systems which are either supervised or semi-supervised and compare them with my unsupervised systems where needed.

2.4.1. Generative Approaches

Edit Distance based Measures

Edit distance based measures are often used for transliteration mining (Jiampojamarn et al., 2010; Noeman and Madkour, 2010). They require a table with the mapping of source and target language characters. Noeman and Madkour (2010) achieve this mapping by aligning labeled data at character level using GIZA++ and extract the most probable character alignments with their probability. Jiampojamarn et al. (2010) align labeled data using the M2M-aligner. For a target character sequence, they take the highly probable English character sequence according to the trained model as its romanization. They define a uniform cost of substitution, insertion and deletion for the experiments. To increase the precision of the mining system, they remove all those

candidate pairs from the list of word pairs in which the English word ends with a consonant and the foreign word ends with a vowel. This heuristic helps them to filter out morphological variants which only differ by an ending character from a transliteration.

In initial experiments, I build hand-crafted transliteration rules and use them with the edit distance metric to mine transliteration pairs from a parallel corpus. The deletion, substitution and insertion costs are optimized on the development set. I extrinsically evaluated the mined list by building a transliteration system on it. The system achieves high precision but could not produce complex transliterations. The details of the system are presented in Section 3.5.1.

Graph Reinforcement

The labeled data is generally small and might not include the complete set of possible multigrams of a language pair. The idea of graph reinforcement is to use the existing labeled data and deduce new multigrams which are not directly visible in the labeled data.

Kahki et al. (2011) propose a supervised transliteration mining system which uses graph reinforcement to deduce character mappings which are missing in the labeled data. They build a baseline transliteration mining system on the labeled data which is aligned using the HMM aligner of He (2007) which uses word-dependent transition probabilities and Bayesian learning. The baseline system is evaluated on four language pairs. It shows high precision but low recall of transliteration pairs. The labeled data does not contain examples of all multigrams. Consider an example of an English/Arabic corpus (from Kahki et al. (2011)): the Arabic letter ق/qa can be aligned to English letters q, k and c. However, in the alignment of the training data, no alignment from ق to c exists. The Arabic letter ک/ka can also be aligned to English letters q, k and c, and all these mappings are prevalent in the training corpus. Using the overlapping English characters mapped to ق and ک, a number of paths can be deduced which lead to the alignment of ق to c. In Figure 2.2, there are two

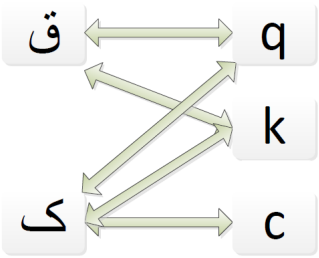


Figure 2.2.: Motivation of graph reinforcement technique

paths: ق → q → ک → c and ق → k → ک → c which give evidence to a new multigram ق → c. The method is formalized as follows.

Consider a bipartite graph G consisting of a set of source language character sequences S, a set of target language character sequences T and a mapping M between them. The Bayesian learner used in the baseline system provides mappings of source character sequences to target character sequences with conditional probabilities calculated on labeled training data. I refer to these mappings as multigrams. The probability of each multigram is defined as $m(t|s)$ where s and t are source and target character sequences. The new multigrams are deduced by traversing the alignment graph from $S \rightarrow T \rightarrow S \rightarrow T$. The induced multigram score $m(t'|s')$ is defined as follows:

$$m(t'|s') = 1 - \prod_{\forall s \in S, t \in T} (1 - m(t'|s)m(s|t)m(t|s'))$$

where s' and t' represents the source and target character sequence of a new multigram. The term $1 - m(t'|s)m(t|s)m(t|s')$ ensures that a multigram should get high probability if it is achieved from multiple paths.

Apart from deducing good multigrams, the graph reinforcement also introduces irrelevant multigrams. Kahki et al. (2011) apply link reweighting to decrease the weight of those multigrams whose target character sequence has more source character sequences that map to it.

The Kahki et al. (2011) system is built only on labeled data. They learn

character alignments using an HMM aligner which is based on a conditional probability model while I use a joint probability model for alignment. Due to the training on the labeled data, they learn larger alignment units. I restrict myself to one-to-one alignment as my systems are unsupervised and use only unlabeled data for training.

Self Training

Self training is a semi-supervised method of learning. It first builds a classifier using labeled data. The trained model is then applied to unlabeled data and assigns a score to every example. Based on the score, it extracts the most confident predictions of the classifier and adds them to the labeled data. A classifier is trained on the new labeled data and the process is iterated. At every iteration, a few examples are added to the labeled data. The procedure stops when there are no more examples the classifier is confident about. A complete self-training procedure is shown on Page 43.

Self-training Algorithm (Abney, 2007)

1: $L_0 \leftarrow$ labeled data
2: $U \leftarrow$ unlabeled data
3: $c \leftarrow \text{train}(L_0)$
4: **repeat**
5: $\quad L \leftarrow L_0 + \text{select}(\text{label}(U,c))$
6: $\quad c \leftarrow \text{train}(L)$
7: **until** Stopping criterion is met

Sherif and Kondrak (2007a) apply self-training to mine transliteration pairs. They build a transliteration system on 14 carefully selected transliteration pairs. They use the Forward-Backward algorithm for the training. Every word pair from the unlabeled data is scored using the trained model. Based on the forward probability of word pairs, a few are selected as highly probable transliterations. The selected transliteration pairs are added to the labeled data and the process is iterated. The training stops when there are no more probable transliteration pairs in the unlabeled data.

Huang (2005) presents a mining system to find named entity pairs from a bilingual corpus. He proposes a binary cost function that defines a cost zero for identical characters and one otherwise. The cost function with an edit distance metric is applied to English/X list of word pairs where X is the romanized form of the target language. Based on a threshold on the edit distance score, a list of transliteration pairs is extracted which is later used as seed data for self training. Huang (2005) builds a joint-sequence model on the list of transliteration pairs and scores the list of word pairs. He filters top 500 word pairs based on their probability and adds them to the training data. The process is iterated until there is no more increase in the NE extraction accuracy.

Huang (2005)'s transliteration mining system is the only system in literature which does not use any labeled data for training. However, it requires labeled data and other language dependent resources at various steps of the mining process. His system uses labeled data to find the stopping iteration of self training. The candidate pairs (list of word pairs) are built using source and target language named entity taggers. The general cost function requires both source and target languages to be in Latin script which is not always deterministic. The advantage of Huang's system over my unsupervised mining systems (presented in Chapter 4 and Chapter 5) is that it works for both alphabetic and non-alphabetic languages. My systems work only for alphabetic languages. The advantage of my unsupervised systems is that they do not require any language dependent resource and use only a list of word pairs for training.

Darwish (2010) uses a generative transliteration model initially built on labeled data. The alignment between the transliteration pairs is learned using the HMM alignment model of He (2007). The learner models a full transition model including the self transition probability and the probability of transition from one state to another state. Darwish (2010) uses 1000 transliteration pairs for training. The labeled data is small and might not contain all possible multigrams of a language pair. He applies a variant of SOUNDEX on the English side to reduce the number of characters. Darwish (2010) performed six

runs of his system by varying the training data. The first two runs trained on the labeled data, one of them aligns unlabeled data and the other aligns the modified form of the unlabeled data where the English side is modified with SOUNDEX. In both runs, the system extracts a few word pairs from the unlabeled data that are very probable to be transliteration pairs. In the next four runs, he takes a combination of the mined pairs from the first two runs, uses them to train the system and tests it to different variations of the unlabeled data.

The method of Darwish (2010) is different from Sherif and Kondrak (2007a) as Darwish does not add the probable transliteration pairs to the labeled data and uses only the probable transliteration pairs for training and mining. Huang (2005)'s work is similar to Sherif and Kondrak (2007a). It differs in the seed data which Huang (2005) automatically mined from the bilingual corpus. The major limitation of Huang (2005)'s technique is the requirement of language dependent resources for the optimization of parameters. My unsupervised transliteration mining systems are different as I do not use labeled data and do not require any language specific resource. My heuristic-based system trains the model on the unlabeled data and filters out a few word pairs that are less likely to be a transliteration pair, which is the opposite of what they do. My model-based system models the unlabeled data.

Finite State Automata Framework

Noeman and Madkour (2010) propose a generative model for transliteration mining using the Finite State Transducer Framework. They train the transliteration model on the labeled examples using a phrase based machine translation approach. The labeled examples are character aligned using the Hidden Markov Model (HMM) implemented in GIZA++ (Och et al., 1999). The alignments are further modified by applying heuristics (Koehn et al., 2003) to generate character sequence alignments. They extract the most probable alignments and generate an Arabic to English mapping table which is then used to build a Finite State Transducer (FST). For each source word, the FST generates a

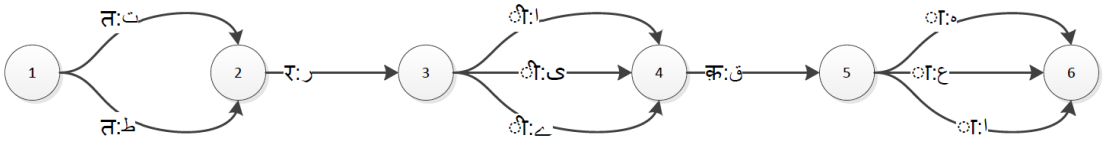


Figure 2.3.: Finite State Transducer

list of candidate transliteration pairs. The candidates which are recognized by a Finite State Acceptor of the target language are considered as transliterations. Figure 2.3 shows an FST model of a Hindi/Urdu word pair तरीक़ा طریقہ (tareeka). The last state is the end state.

Nabende (2010) uses PHMM training for the alignment of the training data and a WFST technique to mine transliteration pairs. The PHMM model is trained using the Expectation Maximization (EM) algorithm. He applies a threshold on the Forward probability to filter transliteration pairs.

PHMM is similar to the joint sequence model that I used in my heuristic-based system and my model-based system except that it has only three states. I also used EM for training. However, my systems are trained on unlabeled data while Nabende (2010) system is trained on labeled data.

2.4.2. Discriminative Approaches

Discriminative approaches require positive and negative labeled data for training. Both types of labeled data is not available for most of the language pairs. I build a generative model for transliteration mining. My system is unsupervised so it does not require any labeled data.

There is some work done on generating negative data using positive labeled data and unlabeled data. However, the automatically generated negative data might not represent all kinds of negative examples that exist in the unlabeled data. The quality of the positive and negative examples is crucial for the performance of the discriminative approaches. Following is the description of the transliteration mining systems that are based on discriminative approaches.

String Similarity

Jiampojamarn et al. (2010) propose a discriminative approach based on string similarity features to learn transliteration information from positive and negative labeled examples. They align the positive labeled word pairs using the M2M-aligner (Jiampojamarn et al., 2007). The positive features are the substrings whose character sequence alignments are consistent with the one-to-one character alignments. This helps the classifier to identify close transliterations as non-transliterations. For negative examples, they generate a list of word pairs from the positive labeled examples by generating all possible source-target word pairs and extract the word pairs which are not transliterations and have a longest common subsequence (LCS) ratio above 0.58. A Support Vector Machine (SVM) classifier is then trained on the positive and negative examples.

String Kernel

The generation of negative examples using LCS requires the target language to be romanized which additionally adds a layer vulnerable to error. Jiampojamarn et al. (2010) use a string kernel method for transliteration mining. The kernel-based methods map the data to a high dimension kernel matrix. A variety of algorithms can then be used to get and analyze the information contained in the matrix. Jiampojamarn et al. (2010) use a string kernel with an SVM which enhances the capability of the SVM to handle long sequences of strings. They use a list of 1000 positive labeled examples for the training. For the negative examples, every word on the source side of the positive labeled examples is paired with every word on the target side of the labeled examples to form a list of negative examples. An M2M-aligner is trained on the positive labeled examples and is applied to 20k randomly selected word pairs from the list of negative examples. For negative examples, they choose the 10k word pairs which are successfully aligned by the aligner.

DirecTL+

Jiampojamarn et al. (2010) use an extension of the transliteration system of Jiampojamarn et al. (2009) for transliteration mining (DirecTL+). They include joint n-gram features to enable the model to learn longer source-target character substrings. The training procedure is as follows.

The model is trained on the positive labeled character aligned examples. For every source word of the labeled data, it generates a list of m-best transliterations. The result is compared with the target side of the examples. Based on the errors made by the system, it updates the weights of the features. The process is iterated with new values of the features.

Jiampojamarn et al. (2010) use the trained transliteration model for transliteration mining. Given a test word pair, they generate a transliteration of the source word and the target word using the transliteration system. A transliteration score function is defined from the edit distance of the source word and its generated transliteration, and the target word and its generated transliteration. A word pair is considered a transliteration pair if its score is greater than a threshold.

My unsupervised mining systems are based on generative models. They are built using only unlabeled data while the discriminative approaches generally require both positive and negative labeled data for training.

2.5. Summary

The major limitation of the previous transliteration mining systems is the use of labeled data for training. The generative approaches require positive labeled data and the discriminative approaches generally require both positive and negative labeled examples for training. The labeled data is not available for most language pairs. This limits the applicability of these systems to only a few language pairs.

I overcome this limitation by presenting the first unsupervised transliteration mining algorithm (See Chapter 4 for detail). I build a joint sequence model

using g2p on unlabeled data. The training is limited to one-to-one alignment which makes it similar to PHMM and identical to Ristad and Yianilos (1998). The model scores the unlabeled data and filters out a few word pairs from the unlabeled data which have the least score, assuming that they are less likely to be transliterations. The joint sequence model is again built on the reduced set of unlabeled data. This is an iterative process, cleaning the unlabeled data in every step until a clean list of transliteration pairs is left. This procedure of mining is close but opposite to self training where in every iteration, a few word pairs that are likely to be transliterations are added to the training data. A second difference is the training which is done using labeled data in self training.

In Chapter 5, I present an unsupervised transliteration mining model. It is a generative model that models the unlabeled data. In contrast to previous generative systems, it consists of two sub models – a transliteration model and a non-transliteration model. The transliteration model is a joint probability model as presented by Li et al. (2004). The non-transliteration model consists of two unigram character models which generate the source and target words independently.

My transliteration model is a simplified form of g2p. I restrict character alignments to a combination of 0–1,1–1,1–0 character alignments. This is equivalent to the stochastic transducer model of Ristad and Yianilos (1998). I train my model using EM. The difference to other models that also use EM for training is that I use EM to learn parameters maximizing the likelihood of the interpolation of both sub-models rather than only the transliteration model. This helps the system to model the complete unlabeled data that consists of both transliterations and non-transliterations, in contrast to the previous systems which train on only labeled data or on a mixture of labeled and unlabeled data.

In the next chapter, I present an application of transliteration by incorporating it into machine translation. I show that it is helpful in more than translating OOVs. I use a rule-based system for transliteration mining which restricts the applicability of this machine translation system to only those language pairs

for which transliteration rules are available. This limitation motivates my work on unsupervised transliteration mining.

3. Machine Translation Through Transliteration

Transliteration has been previously used in machine translation only as a back-off measure to translate named entities (NEs) and out-of-vocabulary (OOV) words in a pre- or post-processing step as outlined in Section 3.2. In this chapter, I present a novel approach to integrate transliteration into statistical machine translation and show that transliteration is helpful for more than just translating out-of-vocabulary words. I use it to increase the weight of transliterations observed infrequently in the word aligned data and to transliterate source language words which are not OOV, but whose target language transliteration is incorrectly not observed.

3.1. Introduction

The work on the machine translation system is joint work with Nadir Durrani. In this chapter, I will use the pronoun "we" to represent the work of two persons. We propose a machine translation model that considers both transliteration and translation when translating a source word. The transliteration system is a joint sequence model. For the training, we mine transliteration pairs using a rule-based mining system which uses the edit distance metric and hand-crafted transliteration rules. The translation model is defined as a word-based joint probability model. At decoding time, the system has to decide whether to translate or transliterate and which translation or transliteration to choose based on the context. Closely related language pairs like Hindi/Urdu

and Lao/Thai are interesting cases in this scenario. In these language pairs, the choice of choosing a translation or transliteration is dependent on the context and there are more than one correct transliterations of a word which are disambiguated using the context. We conduct experiments on the Hindi/Urdu language pair and show that transliteration is helpful in machine translation.

A complete report on the end-to-end system can be found in Durrani et al. (2010). An extended description of the transliteration system used can be found in Sajjad et al. (2011a).

3.2. Previous Work

There has been a significant amount of work on using transliteration to deal with out-of-vocabulary (OOV) words for SMT systems. These systems focus on name transliteration, which is largely independent of context. So, the choice of a transliteration is not dependent on its neighboring words. Figure 3.1 shows an example of an MT system that uses transliteration in a post-processing step for translating only OOVs. A second type of MT system uses transliteration inside of decoding as shown in Figure 3.2. So transliterations compete with translations on the fly and the decoder has to decide based on the language model context which translation to choose or which transliteration to choose. The previous work on these systems is limited to handling OOVs and NEs. In this chapter, we present a novel model for machine translation which uses transliteration in decoding like in Figure 3.2. Here, transliterations compete with translations for all source language words. The system shows an improvement in translation quality and we conclude that transliteration is useful for more than just translating OOVs. Following is a summary of the previous MT systems that use transliteration.

Zhao et al. (2007) present a log-linear block transliteration model to transliterate unseen named entities in Arabic to English machine translation system. Their system shows an improvement in machine translation quality. The limitation of their work is that they are transliterating only NEs and not doing

any disambiguation. The best method proposed by Kashani et al. (2007a) integrates translations provided by external sources such as transliteration or rule-base translation of numbers and dates, for an arbitrary number of entries within the input text. The limitation of Kashani et al. (2007a) is that transliterations do not compete with the internal phrase table and are only used for OOV words. They only compete amongst themselves during a second pass of decoding. Hermjakob et al. (2008) add transliterations to the SMT phrase table dynamically such that they can directly compete with translations during decoding. They first use a named-entity tagger to identify good candidates for transliteration from the Arabic source text and apply a transliterator to words that occur up to 50 times in the training corpus. The assumption here is that frequent words tend to translate properly so there is no need to transliterate them. They add the transliterations to the phrase table where they compete with translations during decoding. This is closer to our approach except that we use transliteration as an alternative to translation for all source language words. So, for every source word the decoder has to disambiguate between a number of translation and transliteration options.

Our focus is disambiguation of Hindi homonyms whereas Hermjakob et al. (2008) are concentrating only on transliterating NE's. Moreover, they are working with a large bitext so they can rely on their translation model and only need to transliterate NEs and OOVs. Our translation model is based on data which is both sparse and noisy. Therefore we pit transliterations against translations for every input word. Sinha (2009) presents a rule-based MT system that uses Hindi as a pivot to translate from English to Urdu. This work also uses transliteration only for the translation of unknown words.

3.3. Hindi and Urdu Language

Hindi is an official language of India and is written in Devanagari script. Urdu is the national language of Pakistan, and also one of the state languages in India, and is written in Perso-Arabic script. Hindi inherits its vocabulary from

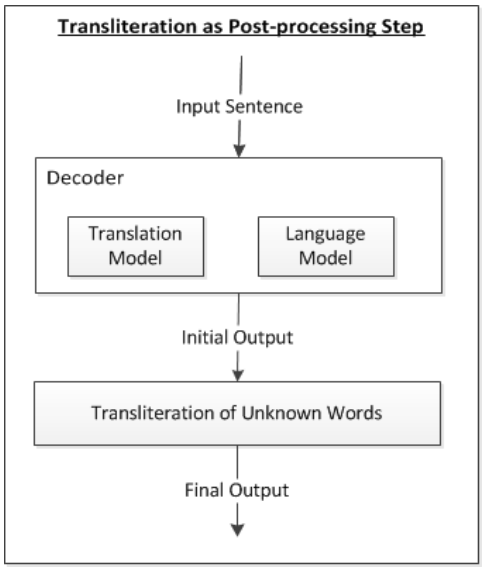


Figure 3.1.: Transliteration as post-processing step in machine translation

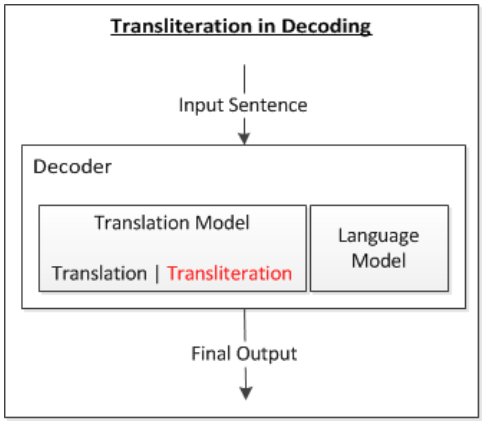


Figure 3.2.: Transliteration in decoding

Sanskrit while Urdu descends from several languages including Arabic, Farsi (Persian), Turkish and Sanskrit. Hindi and Urdu share grammatical structure and a large proportion of vocabulary that they both inherited from Sanskrit. Most of the verbs and closed-class words (pronouns, auxiliaries, case-markers, etc) are the same. Because both languages have lived together for centuries, some Urdu words which originally came from Arabic and Farsi have also mixed into Hindi and are now part of the Hindi vocabulary. The spoken form of the two languages is very similar.

The extent of overlap between Hindi and Urdu vocabulary depends upon the domain of the text. Text coming from the literary domain like novels or history tend to have more Sanskrit (for Hindi) and Persian/Arabic (for Urdu) vocabulary. However, news wire that contains text related to media, sports and politics, etc., is more likely to have common vocabulary.

In an initial study on a small news corpus of 5000 words, randomly selected

from BBC[1] News, we found that approximately 62% of the Hindi types are also part of the Urdu vocabulary and thus can be transliterated while only 38% have to be translated. This provides a strong motivation to implement an end-to-end translation system which strongly relies on high quality transliteration from Hindi to Urdu.

Hindi and Urdu have similar sound systems but transliteration from Hindi to Urdu is still very hard. Urdu text is generally written without diacritics whereas Hindi text is written with diacritics. There are some phonemes in Hindi that have several orthographic equivalents in Urdu. For example the "z" soundcan only be written as ज़ whenever it occurs in a Hindi word but can be written as ذ, ز, ض and ظ in an Urdu word. Transliteration becomes non-trivial in cases where the multiple orthographic equivalents for a Hindi word are all valid Urdu words. Context is required to resolve ambiguity in such cases. Our transliterator (described in Section 3.4.1 and Section 3.5.1) gives an accuracy of 81.6% and a 25-best accuracy of 92.3% when tested on a test corpus of 820 words.

The problem we are solving here is more difficult than techniques aimed at handling OOV words, which focus primarily on name transliteration, because we need different transliterations in different contexts; in their case context is irrelevant. For example: consider the problem of transliterating the English word "read" to a phoneme representation in the context "I will read" versus the context "I have read". An example of this for Hindi to Urdu transliteration: the two Urdu words صورت (face/condition) and سورت (chapter of the Koran) are both written as सूरत (surat) in Hindi. The two are pronounced identically in Urdu but written differently. In such cases we hope to choose the correct transliteration by using context. Some other examples are shown in Table 3.1.

Sometimes there is also an ambiguity of whether to translate or transliterate a particular word. The Hindi word शान्ती, for example, will be translated to سکون (peace, sakun) when it is a common noun but transliterated to شانتی (shanti,

[1]http://www.bbc.co.uk/hindi/index.shtml

Hindi	Urdu	Roman	Gloss
आम	عام / آم	Aam	Mango/Ordinary
जाली	جعلی / جالی	Jaali	Fake/Net
शेर	شیر / شعر	Shaer	Lion/Verse

Table 3.1.: Hindi words that can be transliterated differently in different contexts

Hindi	Urdu	Roman	Gloss
सीमा	سیما / سرحد	Seema	Border/Seema
अंबर	امبر / آسمان	Ambar	Sky/Ambar
विजय	وِجے / جیت	Vijay	Victory/Vijay

Table 3.2.: Hindi words that can be translated or transliterated in different Contexts

shanti) when it is a proper name. We try to model whether to translate or transliterate in a given situation. Some other examples are shown in Table 3.2.

3.4. Our Approach

We present a conditional probability model and a joint probability model for translation. Both of our models combine a character-based transliteration model with a word-based translation model. Our models look for the most probable Urdu token sequence u_1^n for a given Hindi token sequence h_1^n. We assume that each Hindi token is mapped to exactly one Urdu token and that there is no reordering. The assumption of no reordering is reasonable given the fact that Hindi and Urdu have identical grammar structure and the same

word order. An Urdu token might consist of more than one Urdu word.[2] The following sections give a mathematical formulation of our two models, Model-1 and Model-2.

3.4.1. Model-1 : Conditional Probability Model

Applying a noisy channel model to compute the most probable translation $\hat{u}_1^n$, we get:

$$\arg\max_{u_1^n} p(u_1^n|h_1^n) = \arg\max_{u_1^n} p(u_1^n)p(h_1^n|u_1^n) \tag{3.1}$$

Language Model

The language model (LM) $p(u_1^n)$ is implemented as an n-gram model using the SRILM-Toolkit (Stolcke, 2002) with Kneser-Ney smoothing. The parameters of the language model are learned from a monolingual Urdu corpus. The language model is defined as:

$$p(u_1^n) = \prod_{i=1}^{n} p_{LM}(u_i|u_{i-k}^{i-1}) \tag{3.2}$$

where k is a parameter indicating the amount of context used (e.g., $k = 4$ means 5-gram model). The token sequence u_1^n is converted to a word sequence w_1^m in order to compute its LM probability. u_i can be a single or a multi-word token. A multi-word token consists of two or more Urdu words. For a multi-word u_i we do multiple language model look-ups, one for each u_{i_x} in $u_i = u_{i_1}, \ldots, u_{i_m}$ and take their product to obtain the value $p_{LM}(u_i|u_{i-k}^{i-1})$.

Language Model for Unknown Words: Our model generates transliterations that can be known or unknown to the language model and the translation

[2] This occurs frequently in case markers with nouns, derivational affixes and compounds etc. These are written as single words in Hindi as opposed to Urdu where they are written as two words. For example खूबसूरत (beautiful ; khobsurat) and आपका (your's ; apka) are written as خوب صورت (khob surat) and آپ کا (ap ka) respectively in Urdu.

model. We refer to the words known to the language model and to the translation model as *LM-known* and *TM-known* words respectively and to words that are unknown as *LM-unknown* and *TM-unknown* respectively.

We assign a special value ψ to the LM-unknown words. If one or more u_{i_x} in a multi-word u_i are LM-unknown we assign a language model score $p_{LM}(u_i|u_{i-k}^{i-1}) = \psi$ for the entire u_i, meaning that we consider partially known transliterations to be as bad as fully unknown transliterations. The parameter ψ controls the trade-off between LM-known and LM-unknown transliterations. It does not influence translation options because they are always LM-known in our case. This is because our monolingual corpus also contains the Urdu part of the translation corpus. The optimization of ψ is described in Section 3.5.2.

Transliteration Model

We define a character-based transliteration model $p_c(h|u)$ in terms of $p_c(h, u)$, a joint character model, which is also used for Chinese-English back-transliteration (Li et al., 2004) and Bengali-English name transliteration (Ekbal et al., 2006). The character-based transliteration probability is defined as follows:

$$\begin{aligned} p_c(h,u) &= \sum_{a_1^n \in align(h,u)} p(a_1^n) \\ &= \sum_{a_1^n \in align(h,u)} \prod_{i=1}^{n} p(a_i|a_{i-j}^{i-1}) \end{aligned} \tag{3.3}$$

where a_i is a pair consisting of the i-th Hindi character h_i and the sequence of 0 or more Urdu characters that it is aligned with. A sample alignment is shown in Table 3.4(b) in Section 3.5.1. Our best results are obtained with a 5-gram model. The parameters $p(a_i|a_{i-j}^{i-1})$ are estimated from a small transliteration corpus which we automatically extracted from the translation corpus. The extraction details are also discussed in Section 3.5.1. Because our overall model is a conditional probability model, joint-probabilities are marginalized using character-based prior probabilities:

$$p_c(h|u) = \frac{p_c(h,u)}{p_c(u)} \tag{3.4}$$

The prior probability $p_c(u)$ of the character sequence $u = c_1^m$ is defined with a character-based language model:

$$p_c(u) = \prod_{i=1}^{m} p(c_i|c_{i-j}^{i-1}) \tag{3.5}$$

The parameters $p(c_i|c_{i-j}^{i-1})$ are estimated from the Urdu part of the character-aligned transliteration corpus.

Translation Model

The translation model (TM) $p(h_1^n|u_1^n)$ is approximated with a context-independent model:

$$p(h_1^n|u_1^n) = \prod_{i=1}^{n} p(h_i|u_i) \tag{3.6}$$

where h_i and u_i are Hindi and Urdu tokens respectively. Our combined model estimates the conditional probability $p(h_i|u_i)$ by interpolating a word-based (translation) model and a character-based (transliteration) model.

$$p(h_i|u_i) = \lambda p_w(h_i|u_i) + (1-\lambda)p_c(h_i|u_i) \tag{3.7}$$

The parameters of the word-based translation model $p_w(h|u)$ are estimated from the word alignments of a small parallel corpus. We only retain 1-1/1-N (1 Hindi word, 1 or more Urdu words) alignments and throw away N-1 and M-N alignments for our models. This is further discussed in Section 3.5.1.

Replacing Equation (3.4) in Equation (3.7) we get:

$$p(h_i|u_i) = \lambda p_w(h_i|u_i) + (1-\lambda)\frac{p_c(h_i,u_i)}{p_c(u_i)} \tag{3.8}$$

Having all the components of our model defined we insert (3.8) and (3.2) in

(3.1) to obtain the final equation:

$$\hat{u}_1^n = \arg\max_{u_1^n} \prod_{i=1}^{n} p_{LM}(u_i|u_{i-k}^{i-1})[\lambda p_w(h_i|u_i) + (1-\lambda)\frac{p_c(h_i,u_i)}{p_c(u_i)}] \tag{3.9}$$

The optimization of the interpolating factor λ is discussed in Section 3.5.2.

3.4.2. Model-2 : Joint Probability Model

This section briefly defines a variant of our model where we interpolate joint probabilities instead of conditional probabilities. The motivation behind using joint probability $p(h,u)$ instead of conditional probability $p(h|u)$ is that the joint probability $p_w(h,u)$ is easily interpolated with the character-based translation model, which is also a joint model. Again, the translation model $p(h_1^n|u_1^n)$ is approximated with a context-independent model:

$$p(h_1^n|u_1^n) = \prod_{i=1}^{n} p(h_i|u_i) = \prod_{i=1}^{n} \frac{p(h_i,u_i)}{p(u_i)} \tag{3.10}$$

The joint probability $p(h_i,u_i)$ of a Hindi and an Urdu word is estimated by interpolating a word-based model and a character-based model.

$$p(h_i,u_i) = \lambda p_w(h_i,u_i) + (1-\lambda)p_c(h_i,u_i) \tag{3.11}$$

and the prior probability $p(u_i)$ is estimated as:

$$p(u_i) = \lambda p_w(u_i) + (1-\lambda)p_c(u_i) \tag{3.12}$$

The parameters of the translation model $p_w(h_i,u_i)$ and the word-based prior probabilities $p_w(u_i)$ are estimated from the 1-1/1-N word-aligned corpus (the one that we also used to estimate translation probabilities $p_w(h_i|u_i)$ previously).

The character-based transliteration probability $p_c(h_i,u_i)$ and the character-based prior probability $p_c(u_i)$ are defined by (3.3) and (3.5) respectively in the

previous section. Putting (3.11) and (3.12) in (3.10) we get

$$p(h_1^n|u_1^n) = \prod_{i=1}^{n} \frac{\lambda p_w(h_i, u_i) + (1-\lambda) p_c(h_i, u_i)}{\lambda p_w(u_i) + (1-\lambda) p_c(u_i)} \tag{3.13}$$

The idea is to interpolate joint probabilities and divide them by the interpolated marginals. The final equation for Model-2 is given as:

$$\begin{aligned} \hat{u}_1^n = \arg\max_{u_1^n} \prod_{i=1}^{n} & p_{LM}(u_i|u_{i-k}^{i-1}) \times \\ & \frac{\lambda p_w(h_i, u_i) + (1-\lambda) p_c(h_i, u_i)}{\lambda p_w(u_i) + (1-\lambda) p_c(u_i)} \end{aligned} \tag{3.14}$$

3.4.3. Search

The decoder performs a stack-based search using a beam-search algorithm similar to the one used in Pharaoh (Koehn, 2004a). It searches for an Urdu string that maximizes the product of translation probability and the language model probability (Equation3.1) by translating one Hindi word at a time. It is implemented as a two-level process. At the lower level, it computes n-best transliterations for each Hindi word h_i according to $p_c(h, u)$. The joint probabilities given by $p_c(h, u)$ are marginalized for each Urdu transliteration to give $p_c(h|u)$. At the higher level, transliteration probabilities are interpolated with $p_w(h|u)$ and then multiplied with language model probabilities to give the probability of a hypothesis. We use 20-best translations and 25-best transliterations for $p_w(h|u)$ and $p_c(h|u)$ respectively and a 5-gram language model.

We start with an empty hypothesis. Input is read from left to right and the Hindi words are translated one by one. A hypothesis is expanded by picking a possible translation/transliteration of the next Hindi word. The parameters $p_w(h|u)$ and $p_c(h|u)$ are both non-zero only for options that occur in the intersection of the two sets otherwise $p_c(h|u)$ is zero for translations and $p_w(h|u)$ is zero for transliterations. The language model probabilities are calculated and the hypothesis probability is updated. A Hindi word is marked as translated after its n-best translation and transliteration lists are exhausted.

The hypotheses are maintained in stacks. Each stack S_i stores hypotheses for possible translations up to Hindi word h_i. To keep the search space manageable and time complexity polynomial we apply pruning and recombination. Since our model uses monotonic decoding we only need to recombine hypotheses that have the same context (last n-1 words). Next we do histogram-based pruning, maintaining the 100-best hypotheses for each stack.

3.5. Evaluation

3.5.1. Training

This section discusses the training of the different model components - translation model, language model and transliteration model. The translation probabilities are learned using a parallel corpus. For language model, we use monolingual data in addition to the target side of the parallel corpus. The transliteration model requires a list of transliteration pairs for training. We use an edit distance metric with the hand-crafted transliteration rules to extract transliteration pairs from word-aligned parallel corpus. The following subsections describe each corpus in detail.

Translation Corpus

We used the freely available EMILLE Corpus as our bilingual resource which contains roughly 13,000 Urdu and 12,300 Hindi sentences. From these we were able to sentence-align 7000 sentence pairs using the sentence alignment algorithm given by Moore (2002).

The word alignments for this task were created by using GIZA++ (Och and Ney, 2003) in both directions. We extracted a total of 107323 alignment pairs (5743 N-1 alignments, 8404 M-N alignments and 93176 1-1/1-N alignments). Of these alignments M-N and N-1 alignment pairs were ignored. We manually inspected a sample of 1000 instances of M-N/N-1 alignments and found that more than 70% of these were (totally or partially) wrong. Of the 30% cor-

rect alignments, roughly one-third constitute N-1 alignments. Most of these are cases where the Urdu part of the alignment actually consists of two (or three) words but was written without space because of lack of standard writing convention in Urdu. For example جا سکتے (can go ; ja saktay) is alternatively written as جاسکتے (can go ; jasaktay) i.e. without space. We learned that these N-1 translations could be safely dropped because we can generate a separate Urdu word for each Hindi word. For valid M-N alignments we observed that these could be broken into 1-1/1-N alignments in most of the cases. We also observed that we usually have coverage of the resulting 1-1 and 1-N alignments in our translation corpus. Looking at the noise in the incorrect alignments we decided to drop N-1 and M-N cases. We do not model deletions and insertions so we ignored null alignments. Also 1-N alignments with gaps were ignored. Only the alignments with contiguous words were kept.

Monolingual Corpus

Our monolingual Urdu corpus consists of roughly 114K sentences. This comprises 108K sentences from the data made available by the University of Leipzig[3] + 5600 sentences from the training data of each fold during cross validation.

Transliteration Corpus

The training corpus for transliteration is extracted from the 1-1/1-N word-alignments of the EMILLE corpus discussed in Section 3.5.1. We use an edit distance algorithm to align this training corpus at the character level and we eliminate word pairs with high edit distance which are unlikely to be transliterations.

We used our knowledge of the Hindi and Urdu scripts to define the initial character mapping. The mapping was further extended by looking into avail-

[3]http://corpora.informatik.uni-leipzig.de/

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 3.3.: Hindi/Urdu handcrafted equivalence rules

able Hindi/Urdu transliteration systems[4,5] and other resources (Gupta, 2004; Malik et al., 2008; Jawaid and Ahmed, 2009). Each pair in the character map 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 3.3. Likewise, I assign a cost of 0.2 to the mappings generated from an Hindi character mapped to two Urdu characters.

The edit distance metric allows insert, delete and replace operations. We manually defined the cost of edit operations. We set a cost of 0.6 for deletions and insertions, except the deletion of Hindi diacritics where the cost of deletion is zero. These costs are optimized on held out data.

We now discuss two special phenomena: 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

[4]CRULP: http://www.crulp.org/software/langproc.htm
[5]Malerkotla.org: http://translate.malerkotla.co.in

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 hand-crafted 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 extracted from the aligned training corpus 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 two ambiguous replacements in the Hindi/Urdu pairs (Table 3.3). If we decrease the filtering threshold or increase the replacement cost, the number of types extracted reduces significantly.

We align the list of word pairs at the character level using the same hand-crafted equivalence rules and the edit distance algorithm. We get four kinds of alignments of Hindi characters to Urdu characters i.e. $1-N$, $\emptyset-N$, $N-\emptyset$ and $N-1$. The alignments are modified by merging unaligned $\emptyset-N$ alignments (no character on source side, N character on target side) with the preceding alignment pair. If there is no preceding alignment pair then it is merged with the following pair. Table 3.4 gives an example showing initial alignment (a) and the final alignment (b) after applying the merge operation. Our model retains $N-\emptyset$ alignments as deletion operations.

We apply a threshold on the edit distance cost of the character aligned word pairs and extract character aligned transliteration pairs. The parameters $p_c(h,u)$ and $p_c(u)$ are trained on the aligned transliteration pairs using

a)	Hindi	∅	b	c	∅	e	f
	Urdu	A	XY	C	D	∅	F
b)	Hindi		b	c		e	f
	Urdu		AXY	CD		∅	F

Table 3.4.: Alignment (a) before (b) after merge

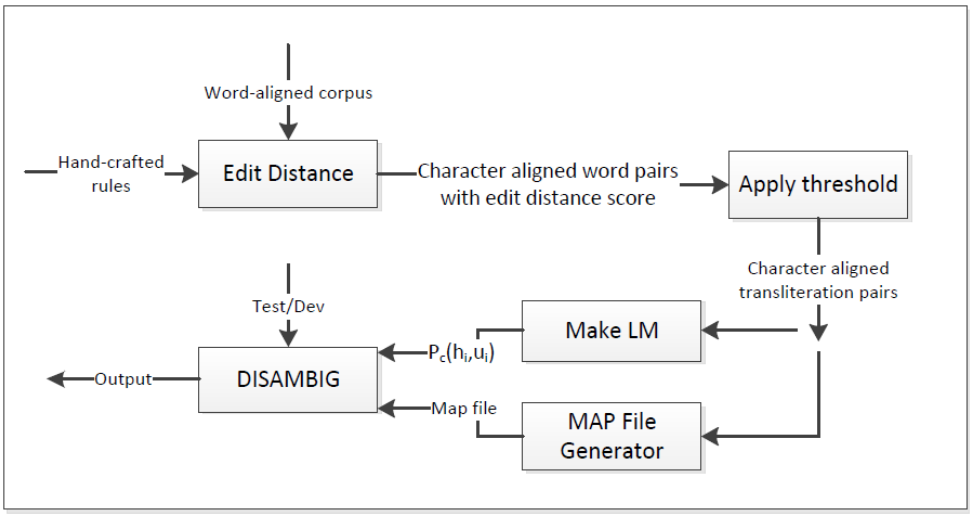


Figure 3.3.: Complete procedure of the transliteration system

the SRILM toolkit. We use Add-1 smoothing for unigrams and Kneser-Ney smoothing for higher n-grams. The DISAMBIG command of the SRILM toolkit uses $p_c(h, u)$ and a map file (source to target language character mappings with their probability) for training and generates N-best transliterations of the input source words. We use probability of one for all character mappings. These probabilities are later automatically learned from the training data. Figure 3.4 shows an example of a map file of Hindi/Urdu. The complete training procedure of the transliteration system starting from the 1-1/1-N word alignments to decoder is summarized in Figure 3.3.

ज	ज-1.0 ز ज-1.0 ج ज-1.0 جو ज-1.0 ض ज-1.0 جی
ी	ी-1.0 ی ी-1.0 ے ी-1.0 ا ी-1.0 و ी-1.0 ڈ ी-1.0 یو ी-1.0 یں ी-1.0 ید ी-1.0 یک ी-1.0 ین ी-1.0 یہ ी-1.0 یا ी-1.0 ी- 1.0 ी-ُ 1.0
द	द-1.0 د द-1.0 ت द-1.0 دی द-1.0 دو
ु	ु- 1.0 ु-ِ 1.0 ु-ُ 1.0
न	न-1.0 ن न-1.0 نی न-1.0 س न-1.0 ل न-1.0 و न-1.0 نر न-1.0 نے न-1.0 न-1.0 ت न-1.0 نا न-1.0 ا न-1.0 (ن न-1.0 ڈ न-1.0 د न-1.0 ئ न-1.0 نع न-1.0 نک न-1.0 نگ न-1.0 نو न-1.0 م न-1.0
क	क-1.0 ک क-1.0 ق क-1.0 کا क-1.0 کی क-1.0 ق क-1.0 ب क-1.0 س क-1.0 کھ क-1.0 قی क-1.0 क- 1.0 ن क-1.0 کر क-1.0 ک ©1.0-क کو क-1.0 كت क-1.0 ك
ल	ल-1.0 ل ल-1.0 لس ल-1.0 لی ल-1.0 لڈ ल-1.0 لک ल-1.0 لہ ल-1.0 (ل ल-1.0 لو ल-1.0 ن ल-1.0 س ल-1.0 لد ल-1.0 لا
त	त-1.0 ت त-1.0 ٹ त-1.0 تی त-1.0 تع त-1.0 تے त-1.0 ن त-1.0 ط त-1.0 تو त-1.0 تج त-1.0 تھ

Figure 3.4.: An example of a map file used in the DISAMBIG module of Figure 3.3. A Hindi character h is mapped with one or more Urdu characters u with probability 1.0. Due to the right to left text direction of Urdu, the character mappings look a bit different. An example of character mapping is "h-u 1.0"

Diacritic Removal and Normalization

In Urdu, short vowels are represented with diacritics but these are rarely written in practice. In order to keep the data consistent, all diacritics are removed. This loss of information is not harmful when transliterating/translating from Hindi to Urdu because undiacritized text is equally readable to native speakers as its diacritized counterpart. However leaving occasional diacritics in the corpus can worsen the problem of data sparsity by creating spurious ambiguity.[6]

There are a few Urdu characters that have multiple equivalent Unicodes. All such forms are normalized to have only one representation.[7]

3.5.2. Experimental Setup

We perform a 5-fold cross validation taking 4/5 of the data as training and 1/5 as test data. Each fold comprises roughly 1400 test sentences and 5600 training sentences.

Parameter Optimization

Our model contains two parameters λ (the interpolating factor between translation and transliteration modules) and ψ (the factor that controls the trade-off between LM-known and LM-unknown transliterations). Both of these parameters are optimized as described below.

Because our training data is very sparse we do not use held-out data for parameter optimization. Instead we optimize these parameters by performing a 2-fold optimization for each of the 5 folds. Each fold is divided into two halves. The parameters λ and ψ are optimized on the first half and the other half is used for testing, then optimization is done on the second half and the

[6] It should be noted though that diacritics play a very important role when transliterating in the reverse direction because these are virtually always written in Hindi as dependent vowels.

[7] www.crulp.org/software/langproc/urdunormalization.htm

first half is used for testing. The optimal value for parameter λ occurs between 0.7-0.84 and for the parameter ψ between $1e^{-5}$ and $1e^{-10}$.

Results

Baseline Pb_0**:** We ran Moses (Koehn et al., 2007) using Koehn's training scripts,[8] doing a 5-fold cross validation with no reordering.[9] For the other parameters we use the default values i.e. 5-gram language model and maximum phrase-length= 6. Again, the language model is implemented as an n-gram model using the SRILM-Toolkit with Kneser-Ney smoothing. Each fold comprises roughly 1400 test sentences. From the data in the other folds, we used 5000 in training and 600 in dev.[10] We also used two methods to incorporate transliterations in the phrase-based system:

Post-process Pb_1**:** All the OOV words in the phrase-based output are replaced with their top-candidate transliteration as given by our transliteration system. This is similar to the method proposed by (Kashani et al., 2007a) other than that they give n-best transliterations to their system along with their probabilities and let the model choose the best one based on internal decoder features whereas we are feeding in the transliteration given the phrase-based system is unable to produce translation.

Pre-process Pb_2**:** Instead of adding transliterations as a post process we do a second pass by adding the unknown words with their top-candidate transliteration to the training corpus and rerun Koehn's training script with the new training corpus and run the decoder again.

[8] http://statmt.org/wmt08/baseline.html

[9] Results are worse with reordering enabled.

[10] After having the MERT parameters, we add the 600 dev sentences back into the training corpus, retrain GIZA, and then estimate a new phrase table on all 5600 sentences. We then use the MERT (Och, 2003) parameters obtained before together with the newer (larger) phrase-table set.

Systems	$\mathbf{F_1}$	$\mathbf{F_2}$	$\mathbf{F_3}$	$\mathbf{F_4}$	$\mathbf{F_5}$	Avg
$\mathbf{Pb_0}$	21.8	14.3	10.6	10.8	14.0	14.3
$\mathbf{Pb_1}$	26.9	15.7	11.6	11.7	15.4	16.25
$\mathbf{Pb_2}$	26.3	15.7	11.5	12.0	15.2	16.13
$\mathbf{M_1}$	35.1	18.5	11.9	12.6	15.0	18.6
$\mathbf{M_2}$	31.7	17.4	11.0	11.5	13.7	17.05

Table 3.5.: Comparing Model-1 and Model-2 with phrase-based systems where F_i is the ith fold

Table 3.5 shows results (taking arithmetic average over 5 folds) from Model-1 and Model-2 in comparison with three baselines discussed above. Both our systems (Model-1 and Model-2) beat the baseline phrase-based system with a BLEU point difference of 4.30 and 2.75 respectively. The transliteration aided phrase-based systems Pb_1 and Pb_2 are closer to our Model-2 results but are way below Model-1 results. The difference of 2.35 BLEU points between M_1 and Pb_1 indicates that transliteration is useful for more than only translating OOV words for language pairs like Hindi/Urdu. Our models choose between translations and transliterations based on context unlike the phrase-based systems Pb_1 and Pb_2 which use transliteration only as a tool to translate OOV words.

The results from Fold 1 are much higher than other folds especially Fold 3 and Fold 4 where they are roughly 3 times lower. We observed that the data in EMILLE is not uniform. In Fold 1 translators have preferred to transliterate where ever it is possible. Fold 2 is a good mix of translations and transliterations whereas Fold 3, 4 and 5 are largely translated. This explains the fact that our results are roughly the same as phrase-based results for these folds whereas transliteration gives us a massive gain in fold 1 and a reasonable advantage in Fold 2. The reason for this is that the Urdu part of EMILLE

was partially translated directly from English to Urdu and partially translated from Hindi to Urdu. We suspect that due to the similar sentence structure of Hindi/Urdu, translators unintentionally stick to the same sentence structure as that of the source sentence and prefer to transliterate Hindi words to Urdu words where possible. That is why Fold 1 and Fold 2 have a higher number of transliterations than other folds.

3.6. Error Analysis

Based on preliminary experiments we found three major flaws in our initial formulations. This section discusses each one of them and provides some heuristics and modifications that we employ to try to correct deficiencies we found in the two models described in Section 3.4.1 and Section 3.4.2.

3.6.1. Heuristic-1

A lot of errors occur because our translation model is built on very sparse and noisy data. The motivation for this heuristic is to counter wrong alignments at least in the case of verbs and functional words (which are often transliterations). This heuristic favors translations that also appear in the n-best transliteration list over only-translation and only-transliteration options. We modify the translation model for both the conditional and the joint model by adding another factor which strongly weighs translation+transliteration options by taking the square-root of the product of the translation and transliteration probabilities. Thus modifying equations (3.8) and (3.11) in Model-1 and Model-2 we obtain equations (3.15) and (3.16) respectively:

$$p(h_i|u_i) = \lambda_1 p_w(h_i|u_i) + \lambda_2 \frac{p_c(h_i, u_i)}{p_c(u_i)} + \lambda_3 \sqrt{p_w(h_i|u_i) \frac{p_c(h_i, u_i)}{p_c(u_i)}} \tag{3.15}$$

$$p(h_i, u_i) = \lambda_1 p_w(h_i, u_i) + \lambda_2 p_c(h_i, u_i) + \lambda_3 \sqrt{p_w(h_i, u_i) p_c(h_i, u_i)} \quad (3.16)$$

For the optimization of lambda parameters we hold the value of the translation coefficient λ_1[11] and the transliteration coefficient λ_2 constant (using the optimized values as discussed in Section 3.5.2) and optimize λ_3 again using 2-fold optimization on all the folds as described above. After optimization, we rescale the lambdas to make their sum equal to 1.

3.6.2. Heuristic-2

When an unknown Hindi word occurs for which all transliteration options are LM-unknown then the best transliteration should be selected. The problem in our original models is that a fixed LM probability ψ is used for LM-unknown transliterations. Hence our model selects the transliteration that has the best $\frac{p_c(h_i,u_i)}{p_c(u_i)}$ score i.e. we maximize $p_c(h_i|u_i)$ instead of $p_c(u_i|h_i)$ (or equivalently $p_c(h_i, u_i)$). The reason is an inconsistency in our models. The language model probability of unknown words is uniform (and equal to ψ) whereas the translation model uses the non-uniform prior probability $p_c(u_i)$ for these words. There is another reason why we can not use the value ψ in this case. Our transliterator model also produces space inserted words. The value of ψ is very small because of which transliterations that are actually LM-unknown, but are mistakenly broken into constituents that are LM-known, will always be preferred over their counter parts. An example of this is अमेरिका (America) for which two possible transliterations as given by our model are امیرکا (america, without space) and امیر کا (ameri ca, with space). The latter version is LM-known as its constituents are LM-known. Our models always favor the latter version. Space insertion is an important feature of our transliteration model. We want our transliterator to tackle compound words, derivational affixes, case-markers

[11] The translation coefficient λ_1 is same as λ used in previous models and the transliteration coefficient $\lambda_2 = 1 - \lambda$

with nouns that are written as one word in Hindi but as two or more words in Urdu. Examples were already shown in Section 3.4's footnote.

We eliminate the inconsistency by using $p_c(u_i)$ as the 0-gram back-off probability distribution in the language model. For an LM-unknown transliteration we now get in Model-1:

$$\begin{aligned}
& p(u_i|u_{i-k}^{i-1})[\lambda p_w(h_i|u_i) + (1-\lambda)\frac{p_c(h_i,u_i)}{p_c(u_i)}] \\
&= p(u_i|u_{i-k}^{i-1})[(1-\lambda)\frac{p_c(h_i,u_i)}{p_c(u_i)}] \\
&= [\prod_{j=0}^{k}\alpha(u_{i-j}^{i-1})]p_c(u_i)[(1-\lambda)\frac{p_c(h_i,u_i)}{p_c(u_i)}] \\
&= [\prod_{j=0}^{k}\alpha(u_{i-j}^{i-1})][(1-\lambda)p_c(h_i,u_i)]
\end{aligned}$$

where $\prod_{j=0}^{k}\alpha(u_{i-j}^{i-1})$ is just the constant that SRILM returns for unknown words. The last line of the calculation shows that we simply drop $p_c(u_i)$ if u_i is LM-unknown and use the constant $\prod_{j=0}^{k}\alpha(u_{i-j}^{i-1})$ instead of ψ. A similar calculation for Model-2 gives $\prod_{j=0}^{k}\alpha(u_{i-j}^{i-1})p_c(h_i,u_i)$.

3.6.3. Heuristic-3

This heuristic discusses a flaw in Model-2. For transliteration options that are TM-unknown, the $p_w(h,u)$ and $p_w(u)$ factors become zero and the translation model probability as given by Equation(3.13) becomes:

$$\frac{(1-\lambda)p_c(h_i,u_i)}{(1-\lambda)p_c(u_i)} = \frac{p_c(h_i,u_i)}{p_c(u_i)}$$

In such cases the λ factor cancels out and no weighting of word translation vs. transliteration occurs anymore. As a result of this, transliterations are sometimes incorrectly favored over their translation alternatives.

In order to remedy this problem we assign a minimal probability β to the word-based prior $p_w(u_i)$ in case of TM-unknown transliterations, which pre-

Systems	F_1	F_2	F_3	F_4	F_5	Avg
M_1	35.1	18.5	11.9	12.6	15.0	18.6
M_2	31.7	17.4	11.0	11.5	13.7	17.05
M_1H_1	35.9	18.6	12.0	12.7	15.1	18.86
M_2H_1	33.7	17.4	11.2	11.7	13.8	17.56
M_1H_2	36.2	18.6	11.9	12.6	15.5	18.97
M_2H_2	34.2	18.0	11.1	11.7	14.3	17.85
M_1H_{12}	37.2	18.7	12.4	12.8	15.6	19.35
M_2H_{12}	36.7	18.0	11.2	11.7	14.2	18.34
M_2H_3	33.9	18.7	12.0	12.7	15.3	18.52
M_2H_{13}	36.3	18.7	12.1	12.5	15.0	18.93
M_2H_{23}	34.4	18.6	11.9	12.6	15.3	18.55
M_2H_{123}	36.9	18.7	12.0	12.4	15.1	19.00

Table 3.6.: Applying heuristics 1 and 2 and their combinations to Model-1 and Model-2

vents it from ever being zero. Because of this addition the translation model probability for LM-unknown words becomes:

$$\frac{(1-\lambda)p_c(h_i,u_i)}{\lambda\beta+(1-\lambda)p_c(u_i)} \text{ where } \beta = \frac{1}{\text{Urdu Types in TM}}$$

3.7. Final Results

This section shows the improvement in BLEU score by applying heuristics and combinations of heuristics in both the models. Tables 3.6 and 3.8 show the improvements achieved by using the different heuristics and modifications discussed in Section 3.6. We refer to the results as M_xH_y where x denotes the model number, 1 for the conditional probability model and 2 for the joint probability model and y denotes a heuristic or a combination of heuristics

	$\mathbf{H_1}$	$\mathbf{H_2}$	$\mathbf{H_{12}}$
$\mathbf{M_1}$	18.86	18.97	19.35
$\mathbf{M_2}$	17.56	17.85	18.34

Table 3.7.: Summary of results of applying Heuristic 1 and Heuristic 2 and their combinations to Model-1 and Model-2

	$\mathbf{H_3}$	$\mathbf{H_{13}}$	$\mathbf{H_{23}}$	$\mathbf{H_{123}}$
$\mathbf{M_2}$	18.52	18.93	18.55	19.00

Table 3.8.: Summary of results of applying Heuristic 3 and its combinations with other heuristics to Model-2

applied to that model.[12]

Both heuristics (H_1 and H_2) show improvements over their base models M_1 and M_2. Heuristic-1 shows notable improvement for both models in parts of test data which has high number of common vocabulary words. Using heuristic 2 we were able to properly score LM-unknown transliterations against each other. Using these heuristics together we obtain a gain of 0.75 over M-1 and a gain of 1.29 over M-2.

Heuristic-3 remedies the flaw in M_2 by assigning a special value to the word-based prior $p_w(u_i)$ for TM-unknown words which prevents the cancellation of interpolating parameter λ. M_2 combined with heuristic 3 (M_2H_3) results in a 1.47 BLEU point improvement and combined with all the heuristics (M_2H_{123}) gives an overall gain of 1.95 BLEU points and is close to our best results (M_1H_{12}).

We concatenate all the fold results to perform significance test. Both our

[12]For example M_1H_1 refers to the results when heuristic-1 is applied to model-1 whereas M_2H_{12} refers to the results when heuristics 1 and 2 are together applied to model 2.

best systems M_1H_{12} and M_2H_{123} are statistically significant ($p < 0.05$)[13] over all the baselines discussed in Section 3.5.2.

One important issue that has not been investigated yet is that BLEU has not yet been shown to have good performance in morphologically rich target languages like Urdu, but there is no metric known to work better. We observed that sometimes on data where the translators preferred to translate rather than doing transliteration our system is penalized by BLEU even though our output string is a valid translation. For other parts of the data where the translators have heavily used transliteration, the system may receive a higher BLEU score. We feel that this is an interesting area of research for automatic metric developers, and that a large scale task of translation to Urdu which would involve a human evaluation campaign would be very interesting.

3.8. Sample Output

This section gives two examples showing how our model ($M1H2$) performs disambiguation. Given below are some test sentences that have Hindi homonyms (underlined in the examples) along with Urdu output given by our system. In the first example (given in Figure 3.5) the Hindi word शेर can be transliterated to شیر (lion) or شعر (verse) depending upon the context. Our model correctly identifies which transliteration to choose given the context.

In the second example (shown in Figure 3.6) the Hindi word शान्ती can be translated to سکون (peace, sakun) when it is a common noun but transliterated to شانتی (shanti, shanti) when it is a proper name. Our model successfully decides whether to translate or transliterate given the context.

[13] We used Kevin Gimpel's tester (http://www.ark.cs.cmu.edu/MT/) which uses bootstrap resampling (Koehn, 2004b), with 1000 samples.

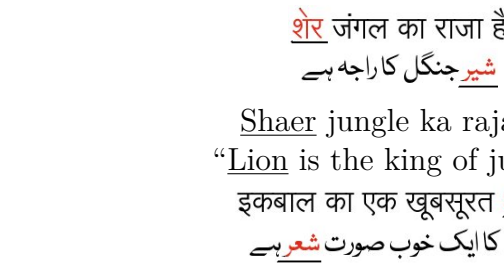


Figure 3.5.: Different transliterations in different contexts

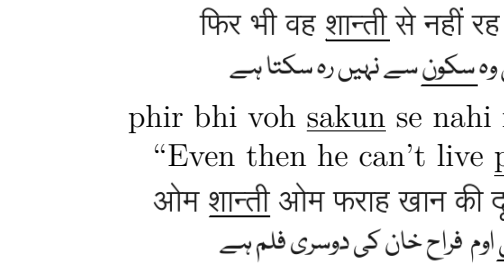


Figure 3.6.: Translation or transliteration

3.9. Summary

We have presented a way to integrate transliterations into machine translation. In closely related language pairs such as Hindi/Urdu with a significant amount of vocabulary overlap, transliteration can be very effective in machine translation. We have addressed two problems. First, transliteration helps overcome the problem of data sparsity and noisy alignments. We are able to generate word translations that are unseen in the translation corpus but known to the language model. Additionally, we can generate novel transliterations (that are LM-Unknown). Second, generating multiple transliterations for homograph Hindi words and using language model context helps us solve the problem of disambiguation.

The transliteration system used in the machine translation system is based on hand-crafted transliteration rules and the edit distance metric. In order to use this machine translation system for other language pairs, one needs transliteration rules. Hand-crafted rules (with weights which work well with the edit distance algorithm) are only available for few language pairs and making these rules is an expensive process in terms of time and effort.

In the next chapter, I will present an unsupervised method to mine transliteration pairs from a parallel corpus which can later be used to automatically learn transliteration units.

3.10. Research Contribution

We presented a novel model that incorporates transliteration into machine translation system. Our experimental results showed that transliteration is helpful for more than just translating OOVs and named-entities. I use it as a tool for disambiguation of homonyms which can be translated or transliterated or transliterated differently based on different contexts.

4. Algorithm for Unsupervised Transliteration Mining

In the previous chapter, I presented a machine translation model that incorporates a transliteration model. The system showed an improvement in performance over the baseline system when tested on the Hindi/Urdu parallel corpus. The transliteration system used in the MT system is based on hand-crafted multigrams. In order to replicate the similar experiment for other language pairs, I need either hand-crafted multigrams or a list of transliteration pairs to build the transliteration system. I presented the previous work on transliteration systems in Section 2.3. Most of the methods are supervised, requiring language dependent resources that are not available for all language pairs. Using transliteration mining, the list of transliteration pairs can be mined automatically from a parallel corpus. However, all previous mining systems are either supervised or semi-supervised as mentioned in Section 2.4. There is no unsupervised system in the literature.

In this chapter, I show that it is possible to extract transliteration pairs from a parallel corpus using an unsupervised method. The automatically extracted transliteration pairs can then be used to build a transliteration system.

4.1. Introduction

I propose an unsupervised transliteration mining algorithm to mine transliteration pairs from parallel corpora. It takes a list of word pairs for training and iteratively mines transliteration pairs from it. For the list of word pairs, I first

align a bilingual corpus at the word level using GIZA++ and create a word-aligned list containing a mix of non-transliterations and transliterations. The non-transliteration is defined as a word pair which is not a transliteration pair. It contains misalignments, translations, etc. I train a statistical transliterator on the word-aligned list. The transliterator is able to learn some transliteration information from the noisy data. I then filter out a few word pairs (those which have the lowest transliteration probabilities according to the trained transliterator) from the word-aligned list which are likely to be non-transliterations. I retrain the transliterator on the filtered list. This process is iterated, filtering out more and more non-transliteration pairs until a nearly clean list of transliteration word pairs is left. The optimal number of iterations is automatically determined by a novel algorithm for stopping criterion. The published report of this work can be found in Sajjad et al. (2011b).

4.2. Models

My algorithms use two different models. The first model is a joint character sequence model which I apply to transliteration mining. I use the grapheme-to-phoneme converter g2p to implement this model. The other model is a standard phrase-based MT model which I apply to transliteration (as opposed to transliteration mining). I build it using the Moses toolkit. Figure 4.2 and Figure 4.3 show the complete procedure of my algorithms with the models.

4.2.1. Joint Sequence Model Using g2p

Here, I briefly describe g2p using notation from Bisani and Ney (2008). The details of the model, its parameters and the utilized smoothing techniques can be found in Bisani and Ney (2008). The training data is a word-aligned list (a set of word pairs consisting of a source word $\mathbf{e}$ and its presumed transliteration $\mathbf{f}$) extracted from a word-aligned parallel corpus.

g2p builds a joint sequence model on the character sequences of the word pairs and infers m-to-n alignments between source and target characters with

Expectation Maximization (EM) training. The m-to-n character alignment units are referred to as multigrams q. The model built on multigrams consisting of source and target character sequences greater than one learns too much noise (non-transliteration information) from the training data and performs poorly. In my experiments, I use multigrams with a maximum of one character on the source and one character on the target side (i.e., 0–1, 1–1, 1–0 character alignment units).

The joint probability $p(e, f)$ of a word pair can be defined as a sum over all multigram sequences:

$$p(e, f) = \sum_{a \in Align(e,f)} p(a) \tag{4.1}$$

where $Align(e, f)$ is the set of all possible multigram sequences producing pair (e, f). The N-gram approximation of the joint probability can be defined in terms of multigrams q_j as:

$$p(a) = p(q_1^k) \approx \prod_{j=1}^{k+1} p(q_j | q_{j-N+1}^{j-1}) \tag{4.2}$$

where q_0, q_{k+1} are set to a special boundary symbol.

N-gram models of order > 1 did not work well because these models tended to learn noise (information from non-transliteration pairs) in the training data. For my experiments, I only trained g2p with the unigram model.

The model is trained with the EM algorithm. In test mode, I look for the best sequence of multigrams given a fixed source and target string and return the probability of this sequence.

For the transliteration mining process, I trained g2p on lists containing both transliteration pairs and non-transliteration pairs.

4.2.2. Statistical Machine Transliteration System

I build a phrase-based machine translation system for transliteration using the Moses toolkit (Koehn et al., 2003). I also tried using g2p for implementing

the transliteration decoder but found Moses to perform better. Moses has an advantage of using a large language model (LM). I build the LM on the target word types in the data to be filtered. Secondly, it uses Minimum Error Rate Training (MERT) which optimizes transliteration accuracy rather than the likelihood of the training data as g2p does. The training data contains more non-transliteration pairs than transliteration pairs. I don't want to maximize the likelihood of the non-transliteration pairs. Instead I want to optimize the transliteration performance for test data.

For training Moses as a transliteration system, I treat each word pair as if it were a parallel sentence, by putting spaces between the characters of each word. The model is built with the default settings of the Moses toolkit. Except that the distortion limit d is set to zero (no reordering). The LM is implemented as a five-gram model using the SRILM-Toolkit (Stolcke, 2002), with Add-1 smoothing for unigrams and Kneser-Ney smoothing for higher n-grams.

Using the log linear formulation, the best target word given a source word can be described as:

$$e_{best} = argmax_e \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i)^{\lambda_\phi} \prod_{i=1}^{|e|} p_{LM}(e_i|e_1...e_{i-1})^{\lambda_{LM}} \tag{4.3}$$

The distortion feature is dropped from the formulation as transliteration involves no reordering. $\bar{f}_1^I$ and $\bar{e}_1^I$ are the I phrase pairs which together form to word pair (f, e). $\phi(\bar{f}_i|\bar{e}_i)$ is the translation probability of the i-th source phrase given the i-th target phrase. p_{LM} is the language model probability. λ_ϕ and λ_{LM} are the weights of the translation model and the language model respectively.

4.3. Extraction of Transliteration Pairs

Training of a supervised transliteration system requires a list of transliteration pairs which is expensive to create. Such lists are usually either built manually or extracted using a classifier trained on manually labeled data and using other

Aamir	आमिर/aamir
Aapne	आपने/aapne
Ability	रकम/rakam
Behave	करने/karne
Chin	चिन/chin

Figure 4.1.: Example of an English/Hindi word-aligned list

language dependent information. In this section, I present an iterative method for the extraction of transliteration pairs from parallel corpora which is fully unsupervised and language pair independent.

Initially, I extract a word-aligned list from a word-aligned parallel corpus using GIZA++. I extract all word pairs which occur as 1-to-1 alignments in the word-aligned corpus and call it the *word-aligned list*. The extracted word pairs are either transliterations, other kinds of translations, or misalignments. I call the word pairs which are not transliterations as *non-transliterations*. Figure 4.1 shows an example of an English/Hindi word-aligned list.

In each iteration, I first train g2p on the word-aligned list. Then I delete those 5% of the (remaining) training data which are least likely to be transliterations according to g2p.[1] I determine the best iteration according to my stopping criterion and return the filtered dataset from this iteration. The stopping criterion uses unlabeled held-out data to predict the optimal stopping point. The following sections describe the transliteration mining method in detail. I will first describe the iterative filtering algorithm (Algorithm 1) and then the algorithm for the stopping criterion (Algorithm 2). In practice, I first run Algorithm 2 for 100 iterations to determine the best number of iterations. Then, I run Algorithm 1 for that many iterations.

[1]Since I delete 5% from the filtered data, the number of deleted data items decreases in each iteration.

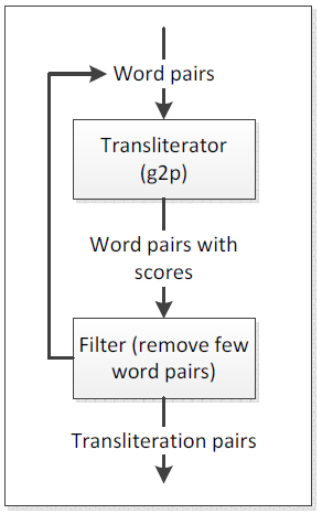


Figure 4.2.: Procedure of Algorithm 1

4.3.1. Algorithm: Mining of Transliteration Pairs

Algorithm 1 builds a joint sequence model using g2p on the training data and computes the joint probability of all word pairs according to g2p. The word pairs with longer source and target strings get lower probability as compare to the smaller strings. I normalize the probabilities by taking the Nth square root where N is the average length of the source and the target string. The training data contains mostly non-transliteration pairs and a few transliteration pairs. Therefore the training data is initially very noisy and the joint sequence model is not very accurate. However it can successfully be used to eliminate a few word pairs which are very unlikely to be transliterations. On the filtered training data, I can train a model which is slightly better than the previous model. Using this improved model, I can eliminate further non-transliterations. The process is iterated. Ideally, the iterative process of Algorithm 1 should stop when the training data contains only transliterations. I propose Algorithm 2 to determine the stopping iteration for Algorithm 1. My results show that at the iteration determined by my stopping criterion, the filtered set mostly contains

Algorithm 1 Mining of transliteration pairs

1: training data ←word-aligned list
2: I ← 0
3: **repeat**
4: Build a joint source channel model on the training data using g2p and compute the joint probability of every word pair.
5: Remove the 5% word pairs with the lowest length-normalized probability from the training data. {and repeat the process with the filtered training data}
6: I ← I+1
7: **until** I = Stopping iteration from Algorithm 2

transliterations and only a small number of transliterations have been mistakenly eliminated (see Section 4.5). Figure 4.2 shows the step-by-step procedure of Algorithm 1.

4.3.2. Algorithm: Selection of Stopping Criterion

In every step of Algorithm 1, the training data is cleaned by a small percentage. Therefore, the percentage of transliterations in the training data increases for every iteration. The idea for selecting the stopping iteration is to build a transliteration system on the training data of every iteration and evaluate its quality on a development set. The assumption here is that the transliteration system built on the training data of iteration j should perform better than the system built on the training data of iteration $j-1$. If this assumption does not hold for an iteration, I select the training data of the previous iteration as the filtered data.

The problem here is that I do not have a development set to find the stopping iteration of Algorithm 1. Therefore, Algorithm 2 automatically determines the best stopping point of the iterative transliteration mining process by using held-out data from the word-aligned list. It is an extension of Algorithm 1. It runs the iterative process of Algorithm 1 on half of the word-aligned list

(training data) for 100 iterations. For every iteration, it builds a transliteration system on the filtered data. The transliteration system is tested on the source side of the other half of the word-aligned list (held-out). The output of the transliteration system is matched against the target side of the held-out data. (These target words are either transliterations, translations or misalignments.) I match the target side of the held-out data under the assumption that all matches are transliterations. The iteration where the output of the transliteration system best matches the held-out data is chosen as the stopping iteration of Algorithm 1.

I will now describe Algorithm 2 in detail. Algorithm 2 initially splits the word pairs into training and held-out data. This could be done randomly, but it turns out that this does not work well for some tasks. The reason is that the parallel corpus contains inflectional variants of the same word. If two variants are distributed over training and held-out data, then the one in the training data may cause the transliteration system to produce a correct translation (but not transliteration) of its variant in the held-out data. This problem is further discussed in Section 4.5.3. Instead of randomly splitting the data, I first create clusters of word pairs which have a common prefix of length 2 both on the source and target language side. I randomly add each cluster either to the training data or to the held-out data.

I repeat the mining process (described in Algorithm 1) to eliminate non-transliteration pairs from the training data. For each iteration of Algorithm 2, i.e., steps 4 to 9, I build a transliteration system on the filtered training data and test it on the source side of the held-out. I collect statistics on how well the output of the system matches the target side of the held-out. The matching scores on the held-out data often make large jumps from iteration to iteration. I take the median of the results from 9 consecutive iterations (the 4 iterations before, the current and the 4 iterations after the current iteration) to smooth the scores. I call this median9. I choose the iteration with the best smoothed score as the stopping point for the mining process. In my tests, the median9 heuristic indicated an iteration close to the optimal iteration. Figure 4.3 shows the procedure of Algorithm 2.

Algorithm 2 Selection of the stopping iteration for the transliteration mining algorithm

1: Create clusters of word pairs from the word-aligned list which have a common prefix of length 2 both on the source and target language side.
2: Randomly add each cluster either to the training data or to the held-out data.
3: I $\leftarrow$ 0
4: **while** I $<$ 100 **do**
5: Build a joint sequence model on the training data using g2p and compute the length-normalized joint probability of every word pair in the training data.
6: Remove the 5% word pairs with the lowest probability from the training data. {The training data will be reduced by 5% of the rest in each iteration}
7: Build a transliteration system on the filtered training data and test it using the source side of the held-out and match the output against the target side of the held-out.
8: I $\leftarrow$ I+1
9: **end while**
10: Collect statistics of the matching results and take the median from 9 consecutive iterations (median9).
11: Choose the iteration with the best median9 score for the transliteration mining process.

Sometimes several nearby iterations have the same maximal smoothed score. In that case, I choose the one with the highest unsmoothed score. Section 4.5 explains the median9 heuristic in more detail and presents experimental results showing that it works well.

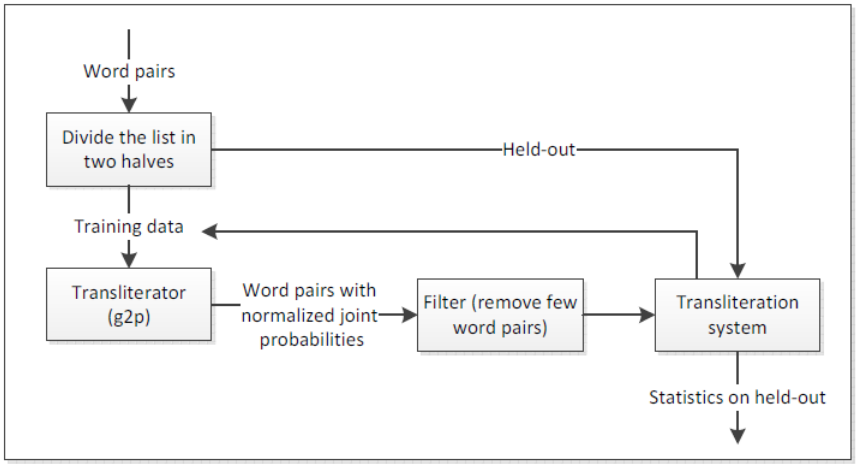


Figure 4.3.: Procedure of Algorithm 2

4.4. Transliteration Mining Using the NEWS10 Dataset

I evaluate my transliteration mining algorithm on the dataset from NEWS 2010 shared task on transliteration mining (NEWS10). These are pairs of titles of Wikipedia pages on the same topic written in different languages. I evaluate my unsupervised system on four language pairs:[2] English/Arabic, English/Hindi, English/Tamil and English/Russian, and compare it with the semi-supervised systems presented at the NEWS10 (Kumaran et al., 2010). My unsupervised system shows an F-measure of up to 92% and outperformed most of the semi-supervised systems on three language pairs.

[2]I do not evaluate on the English/Chinese data because the Chinese data requires word segmentation which is beyond the scope of my work. Another problem is that my extraction method was developed for alphabetic languages and probably needs to be adapted before it is applicable to logographic languages such as Chinese.

4.4.1. Training

The NEWS10 dataset contains training set, reference set and seed set. The training data is a list of parallel phrases. The reference data is a subset of the training data which is annotated with positive and negative examples. The seed data is a small list of positive examples used by the semi-supervised system for initial training. A few examples of the training, reference and seed data is mentioned in Section 5.7.1. I make no use of seed data since my system is fully unsupervised.

I align the parallel phrases of the training data using GIZA++ (Och and Ney, 2003). The alignment is done in both directions i.e. source to target and target to source. For each direction, I run GIZA++ with 5 iterations of Model 1, 4 iterations of the HMM model, and 4 iterations of Model 4. The two alignments are symmetrized using the grow-diag-final-and heuristic (Koehn et al., 2003). I extract all word pairs which occur as 1-to-1 alignments in the word-aligned corpus. I ignore non-1-to-1 alignments because they are less likely to be transliterations for most language pairs. I remove numbers from the list as they are non-transliterations according to the definition of NEWS10. The source language words occurring on the target language side and vice versa are removed. The extracted set of word pairs will be called *word-aligned list* later on. I use it as the training data for Algorithm 1.

4.4.2. Results

I run Algorithm 2 for 100 iterations using the word-aligned list to determine the stopping iteration. Then, I run Algorithm 1 up to the iterations returned by Algorithm 2. I calculate the F-measure of my filtered transliteration pairs against the supplied gold standard using the supplied evaluation tool. Table 4.1 shows the precision and recall of my system. On three language pairs, my system achieves high precision and high recall with an F-measure of up to 92.2% on the English/Hindi dataset. It also maintains a good balance between the values of precision and recall.

	P	R	F
English/Arabic	87.9	86.9	87.4
English/Hindi	90.7	93.9	92.2
English/Tamil	90.9	89.3	90.1
English/Russian	76.8	75.3	76.0

Table 4.1.: Precision, Recall and F-measure of my unsupervised transliteration mining system on the NEWS10 dataset

	Unsupervised	**Semi-supervised**		**Statistics**	
	My	S-Best	S-Worst	Systems	Rank
English/Arabic	87.4	91.5	70.2	16	3
English/Hindi	92.2	94.4	71.4	14	3
English/Tamil	90.1	91.4	57.5	14	3
English/Russian	76.0	87.5	44.4	15	14

Table 4.2.: Summary of results on the NEWS10 dataset where *My* shows the F-measure of my unsupervised mining system against the gold standard using the supplied evaluation tool, *S-Best* is the best result from NEWS10 and *S-Worst* is the worst result from the NEWS10. *Systems* is the total number of participants in the subtask, and *Rank* is the rank I would have obtained if my system had participated

I compare my results with the semi-supervised systems that participated at NEWS10. For English/Arabic, English/Hindi and English/Tamil, my system is better than most of the semi-supervised systems presented at the NEWS 2010 shared task for transliteration mining. It achieves an F-measure of 87.4%, 90.7% and 90.9% when evaluated on English/Arabic, English/Hindi and English/Tamil datasets respectively. Table 4.2 summarizes the F-measures on these datasets.

On the English/Russian dataset, my system achieves 76% F-measure with 76.8% precision and 75.3% recall which is not good compared with the systems

English	Russian	English	Russian
Studio	Студия/Studiya	Catalonia	Каталонии/Katalonii
Estonia	Эстонии/Estonii	Monastery	Монастырь/Monastyr
Tartary	Тартария/Tartariya	Geography	География/Geografiya

Figure 4.4.: Cognates from the English/Russian corpus extracted by my mining system as transliteration pairs. None of them are correct transliteration pairs according to the gold standard

that participated in NEWS10. The English/Russian corpus contains many close transliterations which – according to the NEWS10 definition – are not transliterations of each other. My system learns them and extracts them as transliterations. Figure 4.4 shows a few examples of close transliterations. All these pairs differ by one or two ending characters. They are non-transliterations according to the NEWS10 gold standard and my unsupervised system mines them as transliterations. The issue of close transliteration is discussed in detail in Section 5.7.7.

The two best teams on the English/Russian task presented various extraction methods (Jiampojamarn et al., 2010; Darwish, 2010). Their systems behave differently on English/Russian than on other language pairs. Their best systems for English/Russian are only trained on the seed data and the use of unlabeled data does not help the performance. Since my system is fully unsupervised, and the unlabeled data is not useful, I perform badly.

4.5. Transliteration Mining Using Parallel Corpora

The Wikipedia InterLanguage Links shared task data contains a much larger proportion of transliterations than a parallel corpus. In order to examine how well my method performs on parallel corpora, I apply it to parallel corpora of English/Hindi and English/Arabic. This is the first transliteration mining

evaluation on this task. I manually build gold standard data to compare and evaluate my system.

Algorithm 2 uses two heuristics – median9 and splitting method. The splitting method is used to avoid early peaks in the held-out statistics, and the median9 heuristic smooths the held-out statistics in order to obtain a single peak.[3] In the following sections, I also describe the median9 heuristic and the splitting method of Algorithm 2.

4.5.1. Training

I use the English/Hindi corpus from the shared task on word alignment, organized as part of the ACL 2005 Workshop on Building and Using Parallel Texts (WA05) (Martin et al., 2005). For English/Arabic, I use a freely available parallel corpus from the United Nations (UN) (Eisele and Chen, 2010). I randomly take 200,000 parallel sentences from the UN corpus of the year 2000. I word-aligned the parallel sentences of English/Hindi and English/Arabic using GIZA++ and apply the grow-diag-final-and heuristic as described in Section 4.4.1. I extract word pairs with 1-to-1 alignments in the word-aligned list. I remove numbers from the word-aligned list. If a source language character occurs on the target language side or vice versa, I remove that word pair.

I create gold standards for both language pairs by randomly selecting a few thousand word pairs from the word-aligned lists extracted from the two corpora. I manually tag them as either transliterations or non-transliterations. The English/Hindi gold standard [4] contains 180 transliteration pairs and 2084 non-transliteration pairs and the English/Arabic gold standard contains 288

[3] I do not use the seed data in my system. However, to check the correctness of the stopping point, we tested the transliteration system on the seed data (available with NEWS10) for every iteration of Algorithm 2. I verified that the median9 held-out statistics and accuracy on the seed data have their peaks at the same iteration.

[4] For English/Hindi gold standard, I take a subset of the word-aligned list which consists of m-to-n alignments broken down to 1-to-1 word pairs. So, the gold standard might contain more transliterations and non-transliterations then the word-aligned list built using only the 1-to-1 alignment.

transliteration pairs and 6639 non-transliteration pairs. The gold standards are available to the research community.

4.5.2. Motivation for Median9 Heuristic

Algorithm 2 collects statistics from the held-out data (step 10) and selects the stopping iteration. Due to the noise in the held-out data, the transliteration accuracy on the held-out data often jumps from iteration to iteration. The dotted line in Figure 4.7 shows the held-out prediction accuracy for the English/Hindi parallel corpus. The curve is very noisy and has two peaks. It is difficult to see the effect of the filtering. I take the median of the results from 9 consecutive iterations to smooth the scores. The solid line in Figure 4.7 shows a smoothed curve built using the median9 held-out scores. A comparison with the gold standard (Section 4.5.4) shows that the stopping point (peak) reached using the median9 heuristic is better than the stopping point obtained with unsmoothed scores.

4.5.3. Motivation for Splitting Method

Algorithm 2 initially splits the word-aligned list into training and held-out data. A random split worked well for the WIL data, but failed on the parallel corpora. The reason is that parallel corpora contain inflectional variants of the same word. Figure 4.5 (left) shows a few English/Hindi example word pairs which are morphologically related to other word pairs either in the source language (2nd) or the target language (5th) or in both (1st, 3rd, 4th). If these variants are randomly distributed over training and held-out data, then a non-transliteration word pair such as the English-Hindi pair "change – badlao" may end up in the training data and the related pair "changes – badlao" in the held-out data. The Moses system used for transliteration will learn to "transliterate" (or actually translate) "change" to "badlao". From other examples, it will learn that a final "s" can be dropped. As a consequence, the Moses transliterator may produce the non- transliteration "badlao" for the English word "changes"

in the held-out data. Such matching predictions of the transliterator which are actually translations lead to an overestimate of the transliteration accuracy and may cause Algorithm 2 to predict a stopping iteration which is too early.

I could try to solve this problem by reducing the phrase length of the transliteration system. However, this might also decrease the ability of the transliteration system to produce correct transliterations. Therefore I instead solve it in such a way that inflectional variants of a word are placed either in the training data, or in the held-out, but not in both. In this way, the problem can be solved.[5]

I first create clusters of word pairs which have a common prefix of length 2 or more, both on the source language side and on the target language side. In Figure 4.5, the 1st, 3rd and 4th word pair belong to the same cluster as they are inflectional variants of each other. I randomly add each cluster either to the training data or to the held-out data. The word pairs which are not in any cluster are also randomly added to the training and held-out data (see Algorithm 2).

The graph in Figure 4.6 shows that the median9 held-out statistics obtained after a random data split of the Hindi/English corpus contains two peaks which occur too early. These peaks disappear in the graph of Figure 4.7 which shows the results obtained after a split with the clustering method.

The overall trend of the smoothed curve in Figure 4.7 is very clear. I start by filtering out non-transliteration pairs from the data, so the results of the transliteration system go up. When no more non-transliteration pairs are left, I start filtering out transliteration pairs and the results of the system go down. I use this stopping criterion for all language pairs and achieve consistently good results.

[5]This solution is appropriate for all of the language pairs used in my experiments, but should be revisited if there is inflection realized as prefixes, etc.

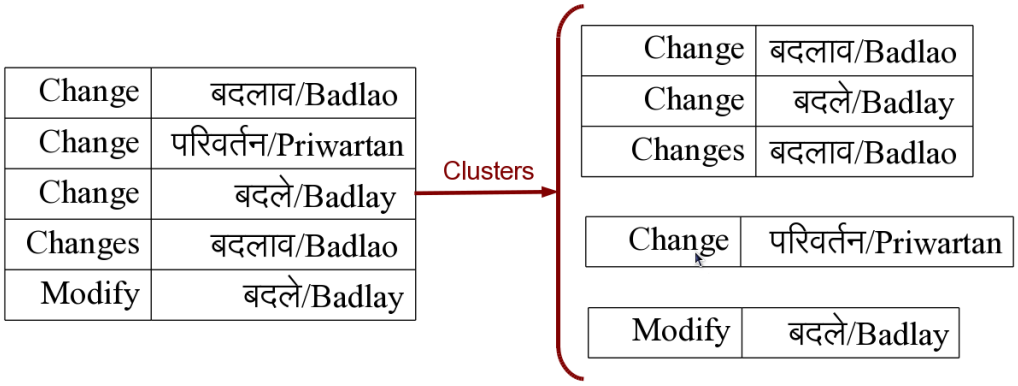


Figure 4.5.: A working example of splitting method

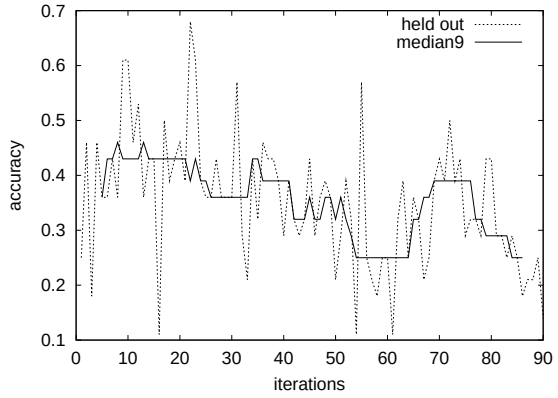


Figure 4.6.: Statistics of held-out prediction of English/Hindi data using modified Algorithm 2 with random division of the word-aligned list. The dotted line shows unsmoothed held-out scores and solid line shows median9 held-out scores

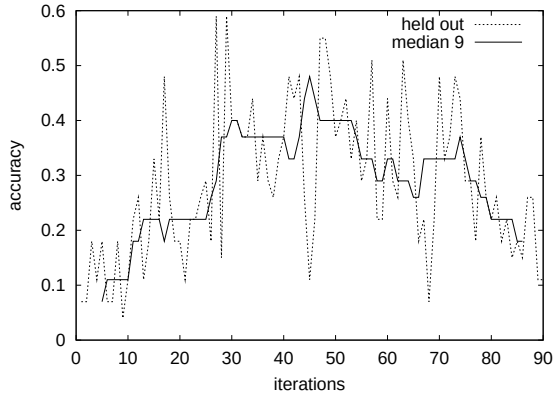


Figure 4.7.: Statistics of held-out prediction of English/Hindi data using Algorithm 2. The dotted line shows unsmoothed held-out scores and solid line shows median9 held-out scores

4.5.4. Results

According to the gold standard, the English/Hindi and English/Arabic datasets contain 8% and 4% transliteration pairs respectively. I repeat the same mining procedure – run Algorithm 2 up to 100 iterations and return the stopping iteration. Then, I run Algorithm 1 up to the stopping iteration returned by Algorithm 2 and obtain the filtered data.

In the English/Hindi experiments, I automatically stopped at iteration 45, where 170 transliteration pairs out of 180 and 98 non-transliteration pairs out of 2084 are left. Table 4.3 shows the precision, recall and F-measure of different iterations of Algorithm 1. At 50th iteration, the number of non-transliteration pairs reduces from 2084 to 46 and there is only a loss of 16 transliteration pairs.

The English/Arabic mining task is harder than the English/Hindi task as it contains only 4% transliteration pairs. I run Algorithm 2 on the English/Arabic word-aligned list. It returns iteration 79. Algorithm 1 is then run up to that iteration. Table 4.4 shows the variation in results on different values of stopping iteration. At 79th iteration, the system filters out 6580 non-transliteration pairs

Iteration	P	R	F
0	8.0	100.0	14.7
10	12.5	96.7	22.2
20	20.3	94.4	33.4
30	32.4	94.4	48.3
40	50.3	94.4	65.5
50	78.1	91.1	84.1

Table 4.3.: Transliteration mining results on different values of stopping iteration using the English/Hindi parallel corpus against the gold standard

Iteration	P	R	F
0	4.16	100.0	8.0
10	9.6	96.5	17.5
30	27.9	94.8	43.1
50	49.9	94.8	64.9
70	67.6	91.3	77.7
90	84.8	44.4	58.3

Table 4.4.: Transliteration mining results on different values of stopping iteration using the parallel corpus of English/Arabic against the gold standard

from 6639 non-transliteration pairs and there is a loss of only 91 transliteration pairs.

Table 4.5 shows the mining results on the English/Hindi and English/Arabic corpora. The mining system has wrongly identified a few non-transliteration pairs as transliterations (see Table 4.5, last column). Most of these word pairs are close transliterations and differ by only one or two characters from perfect transliteration pairs. The word-aligned list contains very few transliteration pairs and might not contain all multigrams of source and target language characters. The close transliterations provide many valid multigrams and are

	TP	FN	TN	FP	P	R	F
English/Hindi	170	10	2039	45	79.1	94.4	86.1
English/Arabic	197	91	6580	59	77.0	68.4	72.5

Table 4.5.: Transliteration mining results using the parallel corpus of English/Hindi (EH) and English/Arabic (EA) against the gold standard

helpful for the mining system. A few examples of close transliteration pairs is shown in Figure 5.27.

4.6. Summary

I have proposed a method to automatically extract transliteration pairs from parallel corpora without supervision or linguistic knowledge. I evaluated it against the supervised and semi-supervised systems of NEWS10 and achieved high F-measure and performed better than most of the semi-supervised systems. I also evaluated my method on parallel corpora. This is a more difficult task than NEWS10 dataset as there are very few transliterations in the word-aligned list. My unsupervised system achieved high F-measure on both language pairs.

On English/Russian NEWS10 dataset, my system learned close transliterations and classified them as transliterations. The number of close transliterations is very high in the English/Russian data. The unsupervised system learns to delete the ending characters of these word pairs at high probability and extracts them as transliterations. This shows that the unlabeled data is not providing enough evidence to correctly classify close transliterations. A semi-supervised system that uses a combination of labeled and unlabeled data might solve this problem. In the next chapter, I present a model-based ap-

proach to transliteration mining. The model is flexible enough to be used for supervised, semi-supervised and unsupervised transliteration mining.

4.7. Research Contribution

I showed that transliteration mining can be done in an unsupervised fashion. The system has competitive results with the state-of-the-art supervised and semi-supervised systems.

I observed that the unsupervised mining system has problem of wrongly classifying close transliterations as transliterations when trained on a dataset with large number of close transliterations. For these datasets, the transliteration pairs in the unlabeled data are not sufficient for the correct classification of close transliterations.

5. Transliteration Mining Model

I call the transliteration mining system described in the previous chapter the *heuristic-based system* from now on.

The heuristic-based system is computationally expensive. Algorithm 2 runs EM training 100 times and each EM training runs for 100 iterations. For every EM, it also builds a phrase-based machine translation system (used as a transliteration model) on the training data. Another weakness of the heuristic-based system is that it is not extendable to use the available labeled data, which could help solve the problem of wrongly classifying close transliterations as transliterations.

In this chapter, I present a novel model of unsupervised transliteration mining. It uses only unlabeled data for training. The system is language pair independent and very efficient. It requires only a single EM training run to mine transliteration pairs. I also propose an extension of the unsupervised model that is able to use the labeled data resources and work both in a semi-supervised and in a supervised fashion.

5.1. Introduction

I propose an unsupervised transliteration mining model. It models unlabeled data which consists of transliterations and non-transliterations. I define the transliteration mining model as a mixture of a transliteration sub-model and a non-transliteration sub-model. The intuition in dividing the model into two sub-models is that if there is a good transliteration model, then a transliteration pair should have high probability according to the transliteration model

and a low probability according to the non-transliteration model. Likewise, a non-transliteration pair should have a low transliteration probability according to the transliteration model and a high non-transliteration probability.

I propose a semi-supervised extension of my unsupervised model which uses small labeled data and is able to learn close transliterations correctly. The basic idea of the semi-supervised system is to rely on the probability estimates calculated from labeled data and to use unlabeled data probability distribution as a backoff distribution. An extended presentation and evaluation of my unsupervised and semi-supervised systems can be found in Sajjad et al. (2012).

I also present a fully supervised model for transliteration mining. It uses an identical model as that of the unsupervised mining. The only difference is the training data which consists of only positive labeled examples in contrast to unlabeled data used in the unsupervised system.

I compare the results of my three systems and show that if labeled data is available, it is always good to build a semi-supervised system than an unsupervised system or a supervised system.

I define the transliteration model using a joint source channel model (Li et al., 2004). The character alignments of a source word and a target word are deduced using Expectation Maximization (EM). The non-transliteration model is a fixed joint probability model of randomly seeing a source word with a target word. At test time, a word pair is classified as transliteration if it has a higher probability assigned by the transliteration sub-model than by the non-transliteration sub-model.

For semi-supervised system, I modify EM training and add a new S-step to it. The S-step takes the probability estimates from the unlabeled data (computed in the M-step) and uses them as a backoff distribution to smooth probabilities which were estimated from the labeled data. The smoothed probabilities are then used in the next E-step. In this way, the parameters learned by EM are constrained to values which are close to those estimated from the labeled data.

All three transliteration mining systems use the same basic model for transliteration mining. They differ in the training data. The mathematical formula-

tion, model estimation and implementation details of the systems are described in the following sections.

This chapter is organized as follows. Section 5.2, 5.3 and 5.4 present my unsupervised, semi-supervised and supervised transliteration mining models. In Section 5.5, I describe my higher order transliteration mining models. I discuss the smoothing to deal with unknowns in testing in Section 5.6. I evaluate my transliteration mining systems on the dataset provided at NEWS 2010 shared task on transliteration mining (Kumaran et al., 2010) (NEWS10) and on parallel corpora. Section 5.7 presents an analysis of my unsupervised, semi-supervised and supervised systems with varying ngram order on the NEWS10 dataset. It also compares the results with the state-of-the-art semi-supervised and supervised systems. In Section 5.8, I evaluate my mining model on parallel corpora. Section 5.9 summarizes the chapter and Section 5.10 presents the contributions made in this chapter.

5.2. Unsupervised Transliteration Mining Model

A source word and its corresponding target word can be character-aligned in many ways. I refer to a possible alignment sequence which aligns a source word e and a target word f as a. The function $Align(e, f)$ returns the set of all valid alignment sequences a of a word pair (e, f). The joint transliteration probability $p_1(e, f)$ of a word pair is the sum of the probabilities of all alignment sequences:

$$p_1(e, f) = \sum_{a \in Align(e,f)} p_1(a) \tag{5.1}$$

Transliteration systems are trained on a list of transliteration pairs. The alignment between the pairs is learned with Expectation Maximization (EM). I use a simple unigram character alignment model, so an alignment sequence from function $Align(e, f)$ is a combination of 0–1, 1–1, and 1–0 character alignments between a source word e and its transliteration f. I refer to a

Source word	∅	c	∅	e	f
Target word	A	C	D	∅	F
Multigrams	∅-A	c-C	∅-D	e-∅	f-F

Table 5.1.: One possible alignment of a word pair $(cef, ACDF)$

character alignment unit as *multigram* later on and represent it by the symbol q.

There can be more than one multigram sequences that represent an alignment of a transliteration pair. Table 5.1 shows one possible sequence of multigrams of the transliteration pair $(cef, ACDF)$. In a unigram model, the probability of a sequence of multigrams a is the product of the probabilities of the multigrams it contains.

$$p_1(a) = p_1(q_1, q_2, ..., q_{|a|}) = \prod_{j=1}^{|a|} p_1(q_j) \tag{5.2}$$

For an M-gram model, the probability of a sequence of multigrams a is defined as follows:

$$p_1(a) = \prod_{j=1}^{|a|} p_1(q_j | q_{j-M+1}^{j-1}) \tag{5.3}$$

where q_{j-M+1}^{j-1} is the preceding context.

For $M = 2$ and $|a| = 4$, the formulation would be:

$$p_1(a) = p_1(q_1|\hat{s})p_1(q_2|q_1)p_1(q_3|q_2)p_1(q_4|q_3)p_1(/\hat{s}|q_4)$$

where $\hat{s}$ and $/\hat{s}$ are boundary symbols.

The transliteration model $p_1(e, f)$ handles only the transliteration pairs. The transliteration mining system has to learn from data containing both transliterations and non-transliterations. I propose a second model $p_2(e, f)$ to

deal with non-transliteration pairs (the *non-transliteration model*). In a non-transliteration word pair, the characters of the source and target words are unrelated. Therefore, I model them as randomly seeing a source word and a target word together. The non-transliteration model is a fixed model and uses random generation of characters from two unigram models (Gale and Church, 1993). It is defined as follows:

$$p_2(e, f) = p_E(e)\ p_F(f) \tag{5.4}$$

$p_E(e) = \prod_{i=1}^{|e|} p_E(e_i)$ and $p_F(f) = \prod_{i=1}^{|f|} p_F(f_i)$. Likewise Equation 5.3, the (monolingual) higher order source and target language models can be defined as $p_E(e) = \prod_{i=1}^{|e|} p_E(e_i|e_{i-M+1}^{i-1})$ and $p_F(f) = \prod_{i=1}^{|f|} p_F(f_i|f_{i-M+1}^{i-1})$.

The transliteration mining model is an interpolation of the transliteration model $p_1(e, f)$ and the non-transliteration model $p_2(e, f)$:

$$p(e, f) = (1 - \lambda)p_1(e, f) + \lambda p_2(e, f) \tag{5.5}$$

λ is the prior probability of non-transliteration.

Interpolation with the non-transliteration model allows the transliteration model to concentrate on modeling transliterations during EM training. After EM training, transliteration word pairs are assigned a high probability by the transliteration submodel and a low probability by the non-transliteration submodel, and vice versa for non-transliteration pairs. This property is exploited to identify transliterations.

5.2.1. Model Estimation

In this section, I discuss the estimation of the parameters of the transliteration model $p_1(e, f)$ and the non-transliteration model $p_2(e, f)$.

The non-transliteration model captures the random generation of the source and target word characters from two unigram character models. These are fixed models and their parameters are estimated from the source and target

words of the training data, respectively, and the parameters do not change during EM training.

For the transliteration model, I implement a simplified form of the grapheme-to-phoneme converter, g2p (Bisani and Ney, 2008). In the following, I use notations from Bisani and Ney (2008). g2p learns m-to-n character alignments between a source and a target word. I restrict myself to 0–1,1–1,1–0 character alignments. In preliminary experiments, using more than one character on the source side or the target side or both sides of the multigram caused the transliteration model to incorrectly learn non-transliteration information from the training data.

Given a training corpus of N word pairs, the log-likelihood ll can be calculated as the sum of the log probabilities of all training items:

$$\begin{aligned} ll &= \sum_{i=1}^{N} logp(e_i, f_i) \\ &= \sum_{i=1}^{N} log\Big((1-\lambda)p_1(e_i, f_i) + \lambda p_2(e_i, f_i)\Big) \end{aligned}$$

The Expectation Maximization (EM) algorithm is used to train the model. It maximizes the log-likelihood ll of the training data. In the E-step the EM algorithm computes expected counts for the multigrams and in the M-step the multigram probabilities are reestimated from these counts. These two steps are iterated.

The expected count of a multigram q is computed by multiplying the posterior probability of each alignment a with the frequency of q in a and summing these weighted frequencies over all alignments of all word pairs.

$$c(q) = \sum_{i=1}^{N} \sum_{a \in Align(e_i, f_i)} \frac{(1-\lambda)p_1(a, e_i, f_i)}{p(e_i, f_i)} n_q(a) \tag{5.6}$$

$n_q(a)$ is here the number of times the multigram q occurs in the sequence a,

$1-\lambda$ is the prior probability of transliteration, $p_1(a, e_i, f_i)$ is the transliteration probability of a particular alignment a, and $p(e_i, f_i)$ is defined in Equation 5.5.

The new estimate of the probability of a multigram is given by:

$$p(q) = \frac{c(q)}{\sum_{q'} c(q')} \tag{5.7}$$

If I extend the model to a higher order model, the expected count of a multigram sequence q_1^M is defined as:

$$c(q_1^M) = \sum_{i=1}^{N} \sum_{a \in Align(e_i, f_i)} \frac{(1-\lambda) p_1(a, e_i, f_i)}{p(e_i, f_i)} n_{q_1^M}(a) \tag{5.8}$$

$n_{q_1^M}(a)$ is the number of times the multigram occurs in the sequence a.

The conditional probability of the multigram q_i in an M-order model is estimated by:

$$p(q_i | q_{i-M+1}^{i-1}) = \frac{c(q_{i-M+1}^{i})}{\sum_{q'} c(q_{i-M+1}^{i-1}, q')} \tag{5.9}$$

The posterior probability of non-transliteration $p_{ntr}(e, f)$ is calculated using Bayes's rule by multiplying the prior non-transliteration probability with the non-transliteration probability $p_2(e, f)$ and normalizing it by dividing it with the total probability of the word pair $p(e, f)$ (see Equation 5.5).

$$p_{ntr}(e, f) = \frac{\lambda p_2(e, f)}{p(e, f)} \tag{5.10}$$

I calculate the expected count of non-transliterations by summing the posterior probabilities of non-transliteration given each word pair:

$$c_{ntr} = \sum_{i=1}^{N} p_{ntr}(e_i, f_i) = \sum_{i=1}^{N} \frac{\lambda p_2(e_i, f_i)}{p(e_i, f_i)} \tag{5.11}$$

λ is then reestimated by dividing the expected count of non-transliterations by the number of word pairs N.

$$\lambda = \frac{c_{ntr}}{N} \tag{5.12}$$

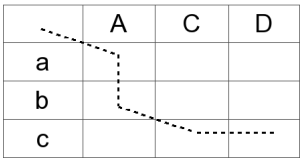
For the first EM iteration, the multigram probabilities are uniformly initialized with the inverse of the number of all possible multigrams that can be built from source and target language characters. After the training, the prior probability of non-transliteration is an approximation of the number of non-transliteration pairs in the training data. If the system uses different data for training and test, I re-estimate the prior probability of non-transliteration in the test mode on the test data using Equation 5.12. The procedure of reestimation of the prior probability is iterated until there is no change in the value of the prior.

5.2.2. Implementation Details

In this section, I discuss the implementation details of my unsupervised mining system. I represent the character alignments of a word pair as a graph $G(N, E)$ with a set of nodes N and edges E. A node $n(i, j) \in N$ represents the coverage vector of the character alignment of a source and a target word. A node $n(i, j)$ means that the source word is aligned up to the ith character and the target word is aligned up to the jth character with $0 < i < |e|$ and $0 < j < |f|$. The transitions between the nodes (called edges) are labeled with multigrams. I allow 0–1,1–1,1–0 character alignments. Every node, except the last node, has three possible transitions to next nodes which are defined as follows:

E is the transition between a set of node pairs $((i, j), (i', j'))$
if $i' = i + 1$ & $j' = j + 1$
or $i' = i$ & $j' = j + 1$
or $i' = i + 1$ & $j' = j$

A node $n(i, j)$ can be connected to nodes with coverage vector $(i, j + 1)$, $(i + 1, j + 1)$ and $(i + 1, j)$. Table 5.2 shows an example of the character alignments of a word pair $(abc\ ACD)$. The first alignment $\mathbf{a} - \mathbf{A}$ in Table



a	b	c	∅
A	∅	C	D

Table 5.2.: Character alignment of a word pair (*abc ACD*)

5.2 shows $(i+1, j+1)$ character alignment scenario where $(i+1)th$ source character and $(j+1)th$ target character are aligned to each other. The second character alignment $\mathbf{b} - \emptyset$ is a case of $(i+1, j)$ alignment as no new target side character is covered as shown in the left side of Table 5.2. The third alignment is again $(i+1, j+1)$ case and the last character alignment is an example of $(i, j+1)$ alignment.

I use the Forward-Backward algorithm to estimate the counts of the multigrams. The algorithm has a forward variable α and a backward variable β which are calculated in the standard way (Deligne and Bimbot, 1995). $\alpha(s)$ is defined as the sum of the product of incoming edges to node s with the value of α at the start node of the incoming edge:

$$\alpha(s) = \sum_{r:(r,s)\in E} \alpha(r)p(q_{rs}) \tag{5.13}$$

r is the start node of the incoming edge to node s and q_{rs} is the label of the edge (r, s)

The backward probability is defined in the similar way but in opposite direction starting from the end node of the graph to the first node:

$$\beta(s) = \sum_{s:(s,t)\in E} \beta(t)p(q_{st}) \tag{5.14}$$

s and t are the start and end nodes of the edge labeled q_{st}. $\beta((i, j), (|e|, |f|))$ and $\alpha(0)$ are equal to one.

Consider a node r connected to a node s via an edge labeled with the

multigram q_{rs}. The expected count of a transition between r and s is calculated using the forward and backward probabilities as follows:

$$\gamma'_{rs} = \frac{\alpha(r)\ p(q_{rs})\ \beta(s)}{\alpha(E)} \tag{5.15}$$

where $E = (|e|, |f|)$ is the final node of the graph.

The expected count of a transition is multiplied by the posterior probability of transliteration $(1 - p_{ntr}(e, f))$ which indicates how likely the string pair is to be a transliteration.

$$\gamma_{rs} = \gamma'_{rs}(1 - p_{ntr}(e, f)) \tag{5.16}$$

The counts γ_{rs} are then summed for all multigram types q over all training pairs to obtain the frequencies $c(q)$:

$$c(q) = \sum_{i=1}^{N} \sum_{(r,s)\in E_i} \gamma_{rs}$$

N is the set of nodes. The re-estimation of the probability of a multigram is calculated using Equation 5.9

The unsupervised mining system can be trained on one unlabeled dataset and tested on different data. In test mode, it is possible that a few multigrams are unknown to the trained model. I use Witten-Bell smoothing to assign a probability to unknown characters and unknown multigrams. This is explained in Section 5.6.

5.3. Semi-supervised Transliteration Mining Model

Our unsupervised transliteration mining system does not require labeled data for training and learns the transliteration information from unlabeled data. The benefit of an unsupervised mining system is that it can be applied to language pairs for which no labeled data is available. However, the unsupervised system

is prone to errors. It focuses on high recall and also mines close transliterations (see Section 5.7.4 for details). In a task dependent scenario, it is difficult for the unsupervised system to mine transliteration pairs according to a particular definition of what is considered a transliteration (which may vary somewhat with the task). In this section, I propose an extension of my unsupervised model which overcomes this shortcoming by using labeled data. The idea is to rely on probabilities from labeled data where they can be estimated reliably and to use probabilities from unlabeled data where the labeled data is sparse. This is achieved by smoothing the labeled data probabilities using the unlabeled data probabilities as a backoff. The following subsections describe the model and the implementation details.

5.3.1. Model

The semi-supervised model uses an identical model to the unsupervised transliteration mining. However, it is trained on both labeled and unlabeled data. The transliteration model trained on the labeled data has better multigram estimates for frequent multigrams than the model trained on the unlabeled data but suffers from sparse data problems. The system can rely on the probability estimates of the labeled data and use the unlabeled data estimates only where the labeled data is either less confident or has no information. I accomplish this by introducing a smoothing mechanism motivated from Witten-Bell smoothing. The labeled data estimates are smoothed using the unlabeled data probability distribution as a backoff distribution. The smoothed probability estimate $\hat{p}(q)$ is defined as follows:

$$\hat{p}(q) = \frac{c_s(q) + \eta_s p(q)}{N_s + \eta_s} \tag{5.17}$$

where $c_s(q)$ is the labeled data count of the multigram q, $p(q)$ is the unlabeled data probability estimate, and $N_s = \sum_q c_s(q)$, and η_s is the number of different multigram types observed in the Viterbi alignment of the labeled data. The

smoothing formulation is motivated from Witten-Bell smoothing (Witten and Bell, 1991).

5.3.2. Model Estimation

I calculate the unlabeled data probabilities in the E-step using Equation 5.5. For labeled data (containing only transliterations) I redefine our transliteration mining model as consisting of only the transliteration sub-model, i.e. set $\lambda = 0$. The transliteration mining model for labeled data is defined as follows:

$$p(e, f) = \sum_{a \in Align(e,f)} p_1(e, f, a) \tag{5.18}$$

In every EM iteration, I smooth the probability distribution using Equation 5.17 in such a way that the estimates of the multigrams of the unlabeled data that do not occur in the labeled data would be used as a backoff.

For different training and test data, the prior probability of non-transliteration is reestimated on the test data using Equation 5.12. The other parameters are kept fixed. The procedure is iterated until convergence.

5.3.3. Implementation Details

I divide the training process of semi-supervised mining in two steps as shown in Figure 5.1. The first step creates a reasonable alignment of the labeled data from which multigram counts can be obtained. The labeled data is a small list of transliteration pairs and it might be too small to learn good character alignments. Therefore I use the unlabeled data to help correctly align it. I combine the labeled and unlabeled data and train the unsupervised transliteration mining system on them. In the expectation step, the prior probability of non-transliteration λ is set to zero on the labeled data since it contains only transliterations. The first step passes the resulting multigram probability distribution to the second step.

I start the second step with the probability estimates from the first step and run the E-step separately on labeled and unlabeled data. The E-step on the

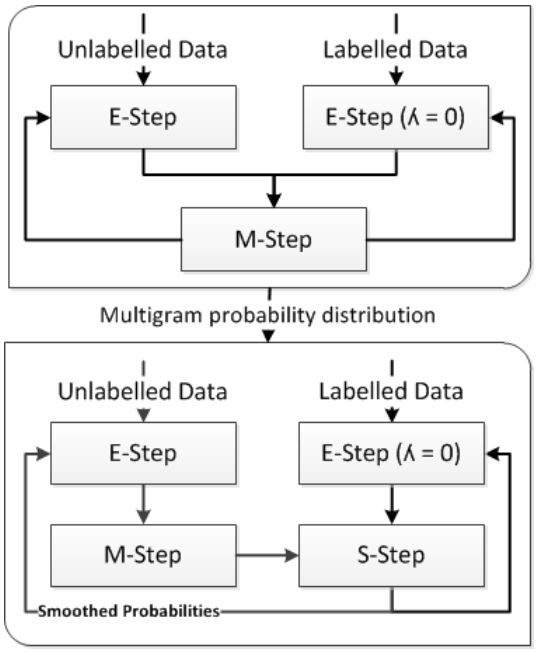


Figure 5.1.: Semi-supervised training

labeled data is done using Equation 5.18, which forces the posterior probability of non-transliteration to zero, while the E-step on the unlabeled data uses Equation 5.5. After the two E-steps, I estimate a probability distribution from the counts obtained from the unlabeled data (M-step) and use it as a back-off distribution in computing smoothed probabilities from the labeled data counts (S-step). Figure 5.1 shows a complete procedure of the semi-supervised training.

5.4. Supervised Transliteration Mining Model

On English/Russian, the unlabeled data contains many close transliterations and the semi-supervised system achieved low precision (See Section 5.7.4). The unlabeled information might be harmful for the system and it might make sense

to use only labeled data for training. To achieve this, I implement a supervised model of transliteration mining.

The supervised transliteration mining system uses an identical model as described in Section 5.2. However, here the training data consists of only transliteration pairs. The prior probability of non-transliteration will be zero for the training. The test data consists of both transliterations and non-transliterations. I train the non-transliteration model using the training data and estimate the prior probability of non-transliteration using the test data. This is similar to the re-estimation of the prior described in Section 5.2.1 for cases when training and test data are different.

5.4.1. Model Estimation

In this section, I describe the estimation of the parameters of the supervised model. The training consists of estimating only the transliteration model. In the test mode, both transliteration and non-transliteration models are used.

Given a training corpus of N word pairs, the log-likelihood $ll_{supervised}$ can be calculated as the sum of the log probabilities of all training items:

$$ll_{supervised} = \sum_{i=1}^{N} log\Big(p_1(e_i, f_i)\Big)$$

The Expectation Maximization (EM) algorithm maximizes the log-likelihood $ll_{supervised}$ of the training data. The E-step computes expected counts for the multigrams and the M-step reestimates the multigram probabilities from these counts. These two steps are iterated.

The expected count of a multigram q (E-step) is computed by multiplying the transliteration probability of each alignment a with the frequency of q in a and summing them over all alignments of all word pairs.

$$c(q) = \sum_{i=1}^{N} \sum_{a \in Align(e_i, f_i)} p_1(a, e_i, f_i) n_q(a) \tag{5.19}$$

For the first iteration of EM, I initialize the multigrams with a uniform distribution which is an inverse of the total number of multigrams in the training set. The new estimate of the probability of a multigram is calculated using Equation 5.7.

The training data consists of only labeled examples. The test data is a mixture of transliteration and non-transliteration word pairs. I build a non-transliteration model on the labeled data using Equation 5.4. The prior probability of non-transliteration is reestimated on the test data using Equation 5.12. The process is iterated until convergence. Every word pair in the test data is scored using the trained transliteration and non-transliteration model. The posterior probability of non-transliteration is calculated using Equation 5.10.

Due to the insufficient training data, there are a few characters and multigrams in test data which are unseen to the training data. I apply the Witten-Bell smoothing to avoid zero probability. I present the higher-order details of the supervised system in Section 5.4.2. The higher order multigram and character sequences from test which are unknown to the trained model are smoothed also using the Witten-Bell smoothing. It is described in Section 5.6.

5.4.2. Implementation Details

I use a similar implementation as described in Section 5.2.2. The character alignments of a word pair are implemented as a graph. The Forward-Backward algorithm is used to estimate the parameters. The training of the supervised mining system involves the labeled data. I did not apply Equation 5.16 on the expected counts of a transition (calculated using Equation 5.15). Instead I directly calculate the counts of multigrams, since in the supervised model I assume all word pairs are transliterations.

5.5. Higher Order Transliteration Mining Models

The unsupervised, semi-supervised and supervised systems described in previous sections build a unigram model of transliteration mining. I also calculate a higher order model from the unigram model to learn contextual information from the training data. In this section, I present a higher order transliteration mining procedure, built on the unigram model which is identical for all unsupervised, semi-supervised and supervised mining systems.

For ngram order $M > 1$, I did not reestimate the counts of a higher order model during the EM training. I instead train only the unigram model and generate the Viterbi alignments of the word pairs. I learn the context information from the Viterbi alignments of the unigram model to build higher order models. The assumption here is that the unigram model is good enough to generate good alignments of test data. The reestimation of higher order models is computationally expensive. Using only the unigram model for training is very efficient and the system is able to learn contextual information from the Viterbi alignments.

The transliteration mining procedure of higher order models is as follows:

I train a unigram transliteration mining model on the training data, generate its Viterbi alignments and compute the higher order counts. The higher order probabilities are then calculated using the higher order counts of the training data using Equation 5.8. Table 5.3 shows the unigram Viterbi alignment of a word pair *abc ACD* with bigram and trigram context.

In the test mode, for every word pair I compute the Viterbi alignment which is the most probable multigram sequence according to the model. The transliteration probability of a test word pair (e, f) is calculated as:

$$p_1(e, f, \hat{a}) = p(q_1, q_2, ...q_{|\hat{a}|}) = \prod_{i=1}^{|\hat{a}|+1} p(q_i|q_1^{i-1}) \tag{5.20}$$

where $q^{|\hat{a}|+1} = /\hat{s}$. $\hat{a}$ is the most probable multigram sequence of the word pair (e, f) which is calculated using the Viterbi algorithm. For $M = 2$, the bigram transliteration probability of the word pair (e, f) is calculated as:

Unigram–	a-A	b-∅	c-C	∅-D	
Bigram–	$\hat{s}$, a-A	a-A, b-∅	b-∅, c-C	c-C, ∅-D	∅-D, /$\hat{s}$
Trigram–	$\hat{s}\hat{s}$, $\hat{s}$, a-A	(s), a-A, b-∅	...	c-C, ∅-D, (/s)	∅-D, /$\hat{s}$, /$\hat{s}\hat{s}$

Table 5.3.: Bigram and trigram multigram context deduced from unigram Viterbi alignment of a word pair "abc" and "ACD" where $\hat{s}$, /$\hat{s}$, $\hat{s}\hat{s}$ and /$\hat{s}\hat{s}$ are the boundary symbols

$$p_1(e, f, \hat{a}) = p(q_1|\hat{s})p(q_1|q_2), ..., p(/\hat{s}|q_{|\hat{a}|})$$

5.6. Smoothing to Deal with Unknowns in Testing

The transliteration mining system is capable of training on one set of unlabeled data and testing on a different set of unlabeled data.[1] There are cases when a test multigram is unknown to the trained model. I apply Witten-Bell smoothing (Witten and Bell, 1991) to assign a small probability to unknown characters and unknown multigrams. The Witten-Bell formulation for the smoothed probability of a multigram q_i is given by:

$$\hat{p}_1(q_i) = \frac{c(q_i) + \eta(.)p_1(q_i)}{\sum_{q'} c(q') + \eta(.)} \tag{5.21}$$

$\eta(.)$ is the number of observed multigram types. For an unknown multigram the probability $p_1(q_i)$ is calculated as the inverse of the product of the total number of source and target language character types plus one i.e. $p(unknown) = 1/(S+1)(T+1)$ where S and T are source and target language character types respectively.

[1]For supervised system, training and test data are always different.

In case of higher order models, there are higher order multigrams of the test data that are unknown to the trained model. I also implement Witten-Bell smoothing to calculate the probability for the unknown higher order multigrams:

$$\hat{p}_1(q_i|q_1^{i-1}) = \frac{c(q_1^i) + \eta(q_1^{i-1})\hat{p}_1(q_i|q_1^{i-1})}{c(q_1^{i-1}) + \eta(q_1^{i-1})} \tag{5.22}$$

q_1^{i-1} is the preceding context of multigram q_i. $c_(q_1^i)$ is the count of the multigram sub-sequence calculated on the training data, $c(q_1^{i-1})$ is the count of the context irrespective of the current multigram and $\eta(q_1^{i-1})$ is the number of different multigram sub-sequence types that follow q_i in the training data.

For a bigram model, $c(q_{i-1}, q_i)$ is the frequency of the bigram in the training data and $c(q_{i-1})$ is the count of the previous multigram in the training data. The smoothed probability estimate of the multigram q_i given the previous multigram q_{i-1} is defined as follows:

$$\hat{p}_1(q_i|q_{i-1}) = \frac{c(q_{i-1}, q_i) + \eta(q_{i-1})\hat{p}_1(q_i)}{c(q_{i-1}) + \eta(q_{i-1})}$$

$\eta(q_{i-1})$ is the number of multigram types that follow q_{i-1} in the training data.

The non-transliteration model is also built on the training data and is formulated in Equation 5.4. I also use Witten-Bell smoothing to assign probabilities to unknown characters and to unknown character sequences of the test data. The smoothed non-transliteration probability of the source language is defined as:

$$\hat{p}_E(e_i|e_1^{i-1}) = \frac{c(e_1^i) + \eta(e_1^{i-1})\hat{p}_E(e_i|e_1^{i-1})}{c(e_1^{i-1}) + \eta(e_1^{i-1})} \tag{5.23}$$

where e_1^{i-1} is the preceding context of character e_i, $c(e_1^i)$ is the count of character e_i with context e_1^{i-1} and $c(e_1^{i-1})$ is the frequency of the context. $\eta(e_1^{i-1})$ is the number of character types that follow the context e_1^{i-1}. The values of all variables are calculated on the training data.

The unigram smoothed probability $\hat{p}_E(e_i)$ is defined in similar way:

$$\hat{p}_E(e_i) = \frac{c(e_i) + \eta(.)p_E(e_i)}{\sum_{e'} c(e') + \eta(.)} \tag{5.24}$$

The number of unique characters in the test data can at most be the twice the number of characters in the training data. I take this assumption and define the probability of unknown characters $p_E(e_i)$ as an inverse of twice the total number of character types in the training data. This can be written as $\frac{1}{2\eta(.)}$ where $\eta(.)$ is the total number of characters in the source language. This smoothing formulation is similar to the Add-λ Smoothing with an additive constant of 0.5. Equation 5.23 and Equation 5.24 are also used for the smoothing of target language characters.

The smoothing formulation slightly differs for the semi-supervised system as the training data consists of both labeled and unlabeled data. The semi-supervised model relies on the labeled data. For a bigram system, when a bigram multigram does not exist in the labeled data distribution, the model backs off to the labeled data unigram distribution which is calculated using Equation 5.17.

$$\hat{p}(q_i|q_{i-1}) = \frac{c_s(q_{i-1}, q_i) + \eta_s(q_{i-1})\hat{p}(q_i)}{c_s(q_{i-1}) + \eta_s(q_{i-1})} \tag{5.25}$$

$c_s(q_{i-1}, q_i)$ is the count of the bigram multigram in the labeled data and $c_s(q_{i-1})$ is the count of some multigram q_{i-1} in the labeled data. $\eta_s(q_{i-1})$ is the number of multigram pair types in the labeled data where q_{i-1} is the first element.

The generalized smoothing equation for the semi-supervised system is:

$$\hat{p}(q_i|q_1^{i-1}) = \frac{c_s(q_1^i) + \eta_s(q_1^{i-1})\hat{p}(q_i|q_1^{i-1})}{c_s(q_1^{i-1}) + \eta_s(q_1^{i-1})} \tag{5.26}$$

5.7. Transliteration Mining Using the NEWS10 Dataset

I evaluate my transliteration mining systems using the dataset provided at NEWS 2010 shared task on transliteration mining (Kumaran et al., 2010) (NEWS10). NEWS10 is a standard task on transliteration mining from Wikipedia InterLanguage Links (WIL), which link the titles of Wikipedia pages that are on the same topic but in different languages. I compare my results on the NEWS10 dataset with the best supervised and semi-supervised systems presented at NEWS10 (Kumaran et al., 2010) and the best supervised and semi-supervised results reported in the literature for the NEWS10 task.

I conduct experiments on the datasets of four language pairs: English/Arabic, English/Hindi, English/Tamil and English/Russian. Each dataset contains training data, seed data and reference data. Figure 5.2 shows a few examples of the English/Hindi WIL phrases taken from the training data. The WIL phrases contain "_" to separate English words while the words of other languages (target languages) are separated by a space. The reference data is a small subset of the training data which is manually annotated with positive and negative examples of transliterated words. A WIL phrase which does not contain any transliteration is represented with an empty title tag. The "Title ID = 5614" in Figure 5.3 is an example of such a phrase. If a phrase is partially a transliteration and partially a non-transliteration, the reference data only contains the transliteration pair (see "Title ID = 6736" in Figure 5.2 and Figure 5.3). The seed data is a list of 1000 transliteration pairs provided to semi-supervised systems or supervised systems for initial training. I use the seed data only in our supervised and semi-supervised system, and not in the unsupervised system. Figure 5.4 shows examples of seed data.

5.7.1. Training Data

The transliteration mining system is trained and tested on a list of word pairs. Every entry in the list consists of a word pair which is a source language

```
<Title ID="6734">
        <SourceEntity>Mughal_Empire</SourceEntity>
        <TargetEntity>मुग़ल साम्राज्य</TargetEntity>
</Title>

<Title ID="6735">
        <SourceEntity>Sari</SourceEntity>
        <TargetEntity>साड़ी</TargetEntity>
</Title>

<Title ID="6736">
        <SourceEntity>Sherwood_number</SourceEntity>
        <TargetEntity>शेरवुड संख्या</TargetEntity>
</Title>

<Title ID="6737">
        <SourceEntity>William_Lawrence_Bragg</SourceEntity>
        <TargetEntity>विलियम लारेन्स ब्राग</TargetEntity>
</Title>
```

Figure 5.2.: Example of English/Hindi NEWS10 training data

```
<Title ID = "5614">
</Title>

<Title ID = "6736">
        <MinedPair ID = "1">
                <SourceName>sherwood</SourceName>
                <TargetName>शेखुड</TargetName>
        </MinedPair>
</Title>

<Title ID = "7465">
        <MinedPair ID = "1">
                <SourceName>indian</SourceName>
                <TargetName>इंडियन</TargetName>
        </MinedPair>
        <MinedPair ID = "2">
                <SourceName>corps</SourceName>
                <TargetName>काप्र्स</TargetName>
        </MinedPair>
        <MinedPair ID = "3">
                <SourceName>of</SourceName>
                <TargetName>आफ</TargetName>
        </MinedPair>
        <MinedPair ID = "4">
                <SourceName>engineers</SourceName>
                <TargetName>इंजिनयर्स</TargetName>
        </MinedPair>
</Title>
```

Figure 5.3.: A few examples of English/Hindi WIL reference data. The empty title tag represents a negative example like “Title ID = 5614” where the word pair corresponds to it is a non-transliteration pair

```
<Name ID="1">
        <SourceName>kosi</SourceName>
        <TargetName>कोसी</TargetName>
</Name>

<Name ID="2">
        <SourceName>yogavati</SourceName>
        <TargetName>योगवति</TargetName>
</Name>

<Name ID="3">
        <SourceName>jeevankala</SourceName>
        <TargetName>जीवनकला</TargetName>
</Name>

<Name ID="4">
        <SourceName>rebellion</SourceName>
        <TargetName>रिबेलियन</TargetName>
</Name>
```

Figure 5.4.: A few examples of English/Hindi NEWS10 seed data

word and its corresponding target language word. A word pair can be either a transliteration pair or a non-transliteration pair. A non-transliteration pair is defined as a word pair which is not a transliteration pair. I generated a list of word pairs from the NEWS10 dataset in the following way:

I word-aligned the parallel phrases of the training data using GIZA++ (Och and Ney, 2003), and symmetrized the alignments using the grow-diag-final-and heuristic (Koehn et al., 2003). I extracted all word pairs which occur as 1-to-1 alignments and I will later refer to them as the *word-aligned list* (this list will be used in some experiments). I compared the word-aligned list with the NEWS10 reference data and found that the word-aligned list is missing some transliteration pairs because of word-alignment errors. I built another list by adding a word pair for every source word that cooccurs with a target word in a parallel phrase/sentence and call it the *cross-product list* later on. The cross-product list is noisier but contains almost all transliteration pairs in the corpus. Table 5.4 shows the statistics of the word-aligned list and the cross-product list calculated using the NEWS10 reference data.[2]

I preprocess both lists and automatically remove numbers from source and target language side as they are defined as non-transliterations (Kumaran et al., 2010). I also automatically remove the source language words that occur on the target language side or vice versa.

5.7.2. Experimental Setup

The unsupervised transliteration mining system is trained on the cross-product list.[3] The semi-supervised system is trained on the cross-product list and the

[2]Due to an inconsistent word definition used in the reference data, I did not achieve 100% recall in the cross-product list. For example, the underscore is defined as a word boundary for English WIL phrases. This assumption is not followed for certain phrases like "New_York" and "New_Mexico". There are 16, 9, 4, and 3 transliteration pairs missing out of 884, 858, 982 and 690 transliteration pairs in the cross- product list of English/Arabic, English/Russian, English/Hindi and English/Tamil respectively.

[3]In Section 5.7.3, I trained the unsupervised system on the word-aligned list to compare the results with my heuristic-based system presented in Chapter 4.

	Word-aligned			Cross-product		
	P	R	F	P	R	F
EA	27.8	97.1	43.3	14.3	98.0	25.0
EH	42.5	98.7	59.4	20.5	99.6	34.1
ET	32.0	98.1	48.3	17.2	99.6	29.3
ER	25.5	95.6	40.3	12.8	99.0	22.7

Table 5.4.: Statistics of word-aligned and cross-product list calculated from the NEWS10 dataset, before mining. *EA* is English/Arabic, *EH* is English/Hindi, *ET* is English/Tamil and *ER* is English/Russian

seed data. The supervised system is trained on only the seed data. The multigrams are initialized with a uniform distribution which is the inverse of the sum of all possible multigrams of the source and target language characters. The prior probability of non-transliteration λ is initialized with value 0.5. For every iteration of EM, the prior probability is reestimated using Equation 5.12.

In the test mode, the trained model is applied to the test data. The prior probability is estimated on the training data. If the training data is identical to the test data then the estimated value of λ in the training is used at test. If they are different, as in the case of the supervised system, I reestimate the prior probability on the test data. The estimated value of the prior probability approximates the number of non-transliteration pairs in the data. The word pairs with a posterior probability of non-transliteration of less than 0.5 are classified as transliteration pairs. Section 5.7.8 discusses the effect of varying threshold on the mining systems in detail.

The word-aligned list calculated from the NEWS10 dataset is used to compare the unsupervised transliteration mining system with my unsupervised heuristic-based system presented in Chapter 4. I could not train the heuristic-based system on the cross-product list because of time complexity. The model-based system trained on the word-aligned list performs better than the heuristic-based system and the model-based system trained on the cross-product list

performs better than the model-based system trained on the word-aligned list. Therefore, for comparison with other systems that evaluated on NEWS10 data, I use only the model-based system trained on the cross-product list.

5.7.3. Unsupervised Model-based System vs. Heuristic-based System

In this section, I compare my heuristic-based system trained on the word-aligned list with my model-based system trained on the word-aligned list and the cross-product list. The heuristic-based system first runs Algorithm 2 to select a stopping iteration for the transliteration mining algorithm. The mining algorithm (Algorithm 1) is then run up to the number of iterations returned by Algorithm 2. Table 5.5 shows the results of the heuristic-based and model-based systems. The model-based system clearly outperformed the heuristic-based system irrespective of the training data. The model-based mining system trained on the word-aligned list shows F-measures 4.3%, 3.3%, 2.8% and 1.7% better than the heuristic-based system on English/Arabic, English/Hindi, English/Tamil and English/Russian respectively. The good thing about the heuristic-based system is that it tends to balance precision and recall. The model-based system focuses on high recall with slightly lower precision. A lower value of the threshold on the posterior probability of non-transliteration would increase precision and lower recall as shown in Section 5.7.8.

The heuristic-based system is computationally expensive. Algorithm 2 runs EM 100 times and for every time it runs EM training, it trains a phrase-based machine translation system for transliteration. The transliteration mining algorithm (Algorithm 1) also runs EM for the number of times returned by Algorithm 2. In contrast, the model-based system runs only one EM training to learn character alignments of the training data.

The model-based transliteration mining system built on the cross-product list consistently performed better than the one built on the word-aligned list. Later, I consider only the model-based systems built on the cross-product list for comparison. I call them the *transliteration mining systems* later on. So, the

	Heuristic-based (WA)			Model-based (WA)			Model-based (CP)		
	P	R	F	P	R	F	P	R	F
EA	87.9	86.9	87.4	87.6	96.2	91.7	89.2	95.7	**92.4**
EH	90.7	93.9	92.2	92.6	98.5	95.5	92.6	99.0	**95.7**
ET	90.9	89.3	90.1	88.5	97.7	92.8	88.3	98.6	**93.2**
ER	76.8	75.3	76.0	66.0	94.6	77.7	67.2	97.1	**79.4**

Table 5.5.: Result of my unsupervised heuristic-based system trained on the word-aligned list (WA) and my unsupervised model-based system trained on the word-aligned list (WA) and on the cross-product list (CP). *EA* is English/Arabic, *EH* is English/Hindi, *ET* is English/Tamil and *ER* is English/Russian. The bolded numbers show the best F-measure

unsupervised model-based system is called the unsupervised transliteration mining system.

5.7.4. Comparison of My Unigram Transliteration Mining Systems

In this section, I compare my unigram unsupervised, semi-supervised and supervised transliteration mining systems. All systems use similar initialization of parameters. The word pairs with a posterior probability of non-transliteration of less than 0.5 are mined as transliteration pairs. Table 5.6 summarizes the results of the unsupervised, semi-supervised and supervised systems on four language pairs.

The unsupervised system achieves high recall with somewhat lower precision. The datasets contain a few non-transliteration pairs that get high probability from the transliteration model. Most of these word pairs only differ by one or two characters from an exact transliteration. There are various phenomena that motivate such constructions. Section 5.7.7 describes them in detail. I call word pairs which differ by one or two characters from an exact transliteration

pair *close transliterations*. The unsupervised mining system is trained only on the unlabeled data. It is difficult for the system to learn the difference between close transliteration pairs and true transliteration pairs. The unsupervised mining system is able to achieve a reasonable precision for English/Arabic, English/Hindi and English/Tamil. In the English/Russian dataset the percentage of close transliterations is very high. These pairs have one or two additional characters from exact transliteration pairs. The unsupervised system learns to delete the additional characters with a high probability and incorrectly mines these word pairs.

On three language pairs, the semi-supervised system achieves only a small gain in F-measure over the unsupervised mining system. This shows that the unlabeled training data is already providing most of the transliteration information. The labeled data is used to help the transliteration mining system to learn the right definition of transliteration. On the English/Russian dataset, the semi-supervised system achieves almost 7% increase in precision with a 2.2% drop in recall compared to the unsupervised system. This provides a 3.7% gain on F-measure. The increase in precision shows that the seed data is helping the system in disambiguating transliteration pairs from close transliterations. However, precision achieved by the semi-supervised system on English/Russian is still less than for other language pairs. In order to learn the contextual information, I build bigram and trigram semi-supervised models. Results are discussed in Section 5.7.5.

The supervised system is built only on the labeled data. It has higher precision than the unsupervised system but recall is lower and for most language pairs overall F-measure is lower than for both the unsupervised and semi-supervised systems. The reason for the low recall is that the seed data consists of only 1000 transliteration pairs which is not enough for the model to learn good estimates. There are various multigrams of the test data that are unknown to the trained model. The smoothed estimate of the unknown multigrams is very low. That helps the supervised system to achieve high precision but at the cost of recall. Another reason might be the reestimation of the prior on the test data which is different from the unsupervised and semi-supervised systems

Unigram	Unsupervised			Semi-supervised			Supervised		
	P	R	F	P	R	F	P	R	F
EA	89.2	**95.7**	**92.4**	**92.6**	92.2	**92.4**	84.9	95.1	89.7
EH	92.6	**99.0**	95.7	**95.5**	97.0	**96.3**	94.2	94.6	94.4
ET	88.3	**98.6**	93.2	**93.4**	95.8	**94.6**	90.4	95.8	93.0
ER	67.1	**97.1**	79.4	**74.0**	94.9	**83.1**	70.0	95.3	80.7

Table 5.6.: Results of the unsupervised, semi-supervised and supervised transliteration mining systems trained on the cross-product list and used the **unigram** model for transliteration and non-transliteration. *EA* is English/Arabic, *EH* is English/Hindi, *ET* is English/Tamil and *ER* is English/Russian. The bolded values show the best precision, recall and F-measure for each language pair

where the prior is reestimated during EM training. The value of the prior has a direct effect on the posterior probability based on which the system generates a mined list of transliteration pairs. In Section 5.7.8, I show the variation of the threshold on the posterior probability of transliteration. The supervised system achieved better F-score at lower values of the posterior than 0.5 which works fine for most of the unsupervised and semi-supervised systems.

On the English/Russian dataset, the supervised system is better than the unsupervised system. The unsupervised system has the problem of classifying close transliterations incorrectly as described before. The supervised system is trained only on the labeled data. It is able to achieve about 3% better precision than the unsupervised system with a recall drop of 1.8%. The phenomenon of close transliterations is described in Section 5.7.7.

The semi-supervised system has the best results for all four language pairs. The unsupervised system has the second best results on three language pairs. It uses only unlabeled data for training, thus could not differentiate between close transliterations and transliterations. The supervised system uses a small labeled dataset for the training which is not enough for the model to learn good estimates of all the multigrams. From the results of Table 5.6, it can be

concluded that if a small labeled dataset is available, it is always good to build a semi-supervised system and if no labeled data is available, an unsupervised system is able to mine transliterations but that there may be a problem with erroneous inclusion of close transliterations.

5.7.5. Comparison of My Higher Order Transliteration Mining Systems

I extend the unigram model to a bigram and trigram model. The unigram model is first trained on training data using the same initialization of parameters as used for the unigram unsupervised, semi-supervised and supervised systems. The model returns Viterbi alignments of word pairs of the training data. The bigram and trigram probabilities of the training pairs are then estimated from the unigram Viterbi alignments. This is an approximation of the higher order model probabilities.

In the test mode, the unigram probability distribution estimated from the training data is used to get the Viterbi alignments of the test data. The bigram and trigram transliteration probabilities are calculated from the Viterbi alignments. A word pair with a posterior probability of non-transliteration less than 0.5 is chosen as a transliteration pair. If the training data and the test data are identical, then the system does not smooth multigram probability distribution and uses the prior probability estimated on the training data. However, there are options provided in the tool to turn on the smoothing and to estimate the prior probability in the test mode.

Table 5.7 summarizes the results of my bigram unsupervised, semi-supervised and supervised systems on four language pairs. The semi-supervised system achieves the best F-measure for all language pairs. The unsupervised system shows lower precision as compared to the semi-supervised and supervised system. I see a similar behavior of the unsupervised system when using a trigram model for transliteration and non-transliteration as shown in Table 5.8. The unsupervised system uses only unlabeled data for training. When used with a higher order model it tends to learn noise from the training data and performs

Bigram	Unsupervised			Semi-supervised			Supervised		
	P	R	F	P	R	F	P	R	F
EA	79.2	**96.4**	86.9	**93.4**	91.5	**92.5**	87.9	95.4	91.5
EH	81.6	**99.2**	89.5	**96.9**	95.3	**96.1**	95.3	94.1	94.7
ET	73.7	**98.8**	84.5	**95.2**	92.5	**93.8**	93.6	91.9	92.8
ER	58.9	**98.0**	73.6	**77.7**	94.3	**85.2**	74.6	94.3	83.3

Table 5.7.: Results of the unsupervised, semi-supervised and supervised transliteration mining systems trained on the cross-product list and using the **bigram** model for transliteration and non-transliteration. *EA* is English/Arabic, *EH* is English/Hindi, *ET* is English/Tamil and *ER* is English/Russian. The bolded values show the best precision, recall and F-measure for each language pair

poorly with a low precision and high recall. The supervised system on the other hand learns only from the labeled data. Thus it shows an increase in precision and a drop in recall from the unigram to bigram model. The trigram model shows a decrease in both precision and recall which is due to data sparsity.

The English/Hindi, English/Arabic, English/Tamil and English/Russian unigram, bigram and trigram results of the unsupervised, semi-supervised and supervised transliteration mining systems are shown in Table 5.9, Table 5.10, Table 5.11 and Table 5.12 respectively.[4] The unsupervised results of different ngram order are consistent on all language pairs. The unigram unsupervised system achieves the best F-measure among all unsupervised systems with a decent balance of precision and recall. For unsupervised transliteration mining, it can be concluded that it is best to use unigram models for transliteration and non-transliteration.

On two language pairs, English/Hindi and English/Tamil, the unigram semi-supervised system shows the best results as compared to the bigram and tri-

[4]These are the identical results presented in Tables 5.6, 5.7 and 5.8. Here, I clustered them in terms of language pair.

Trigram	Unsupervised			Semi-supervised			Supervised		
	P	R	F	P	R	F	P	R	F
EA	58.4	**95.0**	72.4	**95.8**	83.4	**89.2**	82.0	94.6	87.9
EH	45.7	**97.5**	62.2	**97.8**	88.6	92.9	94.5	94.0	**94.2**
ET	33.5	**99.4**	50.1	**97.2**	84.3	90.3	92.6	90.7	**91.7**
ER	38.1	**97.1**	54.7	**81.8**	88.0	**84.8**	72.7	95.3	82.2

Table 5.8.: Results of the unsupervised, semi-supervised and supervised transliteration mining systems trained on the cross-product list and using the **trigram** model for transliteration and non-transliteration. *EA* is English/Arabic, *EH* is English/Hindi, *ET* is English/Tamil and *ER* is English/Russian. The bolded values show the best precision, recall and F-measure for each language pair

gram semi-supervised systems. The bigram and trigram systems show high precision. However, the F-measure drops because of a decrease in recall. The bigram English/Arabic semi-supervised system shows a small F-measure improvement of 0.1% over the unigram semi-supervised system. On the English/Russian dataset, the bigram semi-supervised system shows an F-measure improvement of 2.1% over the unigram semi-supervised system with a 3.7% gain in precision and only a 0.6% drop in recall. The English/Russian dataset contains many close transliterations. This is because of Russian nouns that are marked and their English counterparts do not contain this information. The unigram model is unable to fully learn to differentiate them from transliterations. The contextual information used in the bigram and trigram systems help to learn this information. The F-measure of the trigram system drops 0.4% from the bigram system. It achieves an improvement of 4% in precision from the bigram semi-supervised system but recall drops by 6.3% which causes an overall drop in F-measure.

The bigram and trigram supervised systems show an improvement over the unigram model. On three language pairs, the bigram system obtained high precision and shows the best F-measure for the supervised system.

English/Hindi	Unsupervised			Semi-supervised			Supervised		
	P	R	F	P	R	F	P	R	F
Unigram	**92.6**	99.0	**95.7**	95.5	**97.0**	**96.3**	94.2	**94.6**	94.4
Bigram	81.6	**99.2**	89.5	96.9	95.3	96.1	**95.3**	94.1	**94.7**
Trigram	45.7	97.5	62.2	**97.8**	88.6	92.9	94.5	94.0	94.2

Table 5.9.: Results of the unsupervised, semi-supervised and supervised transliteration mining systems trained on the cross-product list of English/Hindi. The bolded values show the best precision, recall and F-measure for the unigram, bigram and trigram systems

English/Arabic	Unsupervised			Semi-supervised			Supervised		
	P	R	F	P	R	F	P	R	F
Unigram	**89.2**	95.7	**92.4**	92.6	**92.2**	92.4	84.9	95.1	89.7
Bigram	79.2	**96.4**	86.9	93.4	91.5	**92.5**	**87.9**	**95.4**	**91.5**
Trigram	58.4	95.0	72.4	**95.8**	83.4	89.2	82.0	94.6	87.9

Table 5.10.: Results of the unsupervised, semi-supervised and supervised transliteration mining systems trained on the cross-product list of English/Arabic. The bolded values show the best precision, recall and F-measure for the unigram, bigram and trigram systems

English/Tamil	Unsupervised			Semi-supervised			Supervised		
	P	R	F	P	R	F	P	R	F
Unigram	**88.3**	98.6	**93.2**	93.4	**95.8**	**94.6**	90.4	**95.8**	**93.0**
Bigram	73.7	98.8	84.5	95.2	92.5	93.8	**93.6**	91.9	92.8
Trigram	33.5	**99.4**	50.1	**97.2**	84.3	90.3	92.6	90.7	91.7

Table 5.11.: Results of the unsupervised, semi-supervised and supervised transliteration mining systems trained on the cross-product list of English/Tamil. The bolded values show the best precision, recall and F-measure for the unigram, bigram and trigram systems

English/Russian	Unsupervised			Semi-supervised			Supervised		
	P	R	F	P	R	F	P	R	F
Unigram	**67.1**	97.1	**79.4**	74.0	**94.9**	83.1	70.0	**95.3**	80.7
Bigram	58.9	**98.0**	73.6	77.7	94.3	**85.2**	**74.6**	94.3	**83.3**
Trigram	38.1	97.1	54.7	**81.8**	88.0	84.8	72.7	**95.3**	82.2

Table 5.12.: Results of the unsupervised, semi-supervised and supervised transliteration mining systems trained on the cross-product list of English/Russian. The bolded values show the best precision, recall and F-measure for the unigram, bigram and trigram systems

5.7.6. Comparison with the State-Of-The-Art Systems

In this section, I compare my transliteration mining systems with the state-of-the-art semi-supervised and supervised transliteration mining systems that participated in NEWS10 or evaluated on the NEWS10 dataset. There is no unsupervised system in the literature. However, I compare the results of my unsupervised system and semi-supervised system with the semi-supervised and supervised systems. Table 5.13 shows the result of my unigram unsupervised transliteration mining system, unigram and bigram semi-supervised transliteration mining systems in comparison with the best supervised systems presented at NEWS10 (S_{BEST}) and the best supervised/semi-supervised results reported on the NEWS10 dataset (GR, DBN). All systems that evaluated on NEWS10 dataset are supervised or semi-supervised. On three language pairs, my unigram unsupervised mining system performed better than all semi-supervised/supervised systems which participated in NEWS10. It also has competitive results with the best supervised results reported on NEWS10 data sets. On English/Hindi, my unsupervised system outperforms the state-of-the-art supervised and semi-supervised systems.

Kahki et al. (2011) (GR) achieved the best results on the English/Arabic, English/Tamil and English/Russian dataset. For the English/Arabic task, they

	My Systems			State-of-the-art systems		
	Unsuper	Semisupervised		Semisupervised/supervised		
	Unigram	*Unigram*	*Bigram*	S_{Best}	*GR*	*DBN*
EA	92.4	92.4	92.5	91.5	94.1	-
EH	95.7	96.3	96.1	94.4	93.2	95.5
ET	93.2	94.6	93.8	91.4	95.5	93.9
ER	79.4	83.1	85.2	87.5	92.3	82.5

Table 5.13.: Comparison of my system with the state-of-the-art semi-supervised and supervised systems where S_{Best} is the best NEWS10 system, *GR* is the supervised system of kahki11 and *DBN* is the semi-supervised system of nabende11

normalized the data using language dependent heuristics. They applied an Arabic word segmenter which uses language dependent information. Arabic long vowels which have identical sound but are written differently were merged to one form. English characters were normalized by dropping accents. They also used a non-standard evaluation method (discussed in Section 5.7.7).

On the English/Russian dataset, my unsupervised system faces the problem that it extracts close transliterations as transliterations. I have the same problem with my heuristic-based system. These close transliterations are discussed in Section 5.7.7.

There are two English/Russian supervised systems, (Kahki et al., 2011; Jiampojamarn et al., 2010), which are better than my bigram semi-supervised system. Jiampojamarn et al. (2010)'s best system on English/Russian is based on the edit distance method. It achieved 87.5% F-measure with a precision of 88.0% and a recall of 86.9% (in Table 5.13 S_{Best} result on the English/Russian dataset). The Kahki et al. (2011) system is built on seed data only. Both of these systems are focused on high precision. My systems are focused on high recall at the cost of lower precision.

English	Hindi	English	Hindi
Lanthanum	लाञ्थनम/Laanthanum	Sailendra	शैलेन्द्र/Shalendra
January	जनवरी/Janvary	August	अगस्त/Aagast

Table 5.14.: Word pairs with pronunciation differences

5.7.7. Error Analysis

The general errors made by my transliteration mining systems can be classified into the following categories.

Pronunciation Differences

Proper names may be pronounced differently in two languages. Sometimes, English short vowels are converted to long vowels in Hindi such as the English word "Lanthanum" which is pronounced "Laanthanum" in Hindi. A similar case is the English/Hindi word pair "Donald/Donaald". Sometimes two languages use different vowels to produce a similar sound like in the English word "January" which is pronounced as "Janvary" in Hindi. All these words only differ by one or two characters from an exact transliteration. They are non-transliterations according to the gold standard but my unsupervised system extracts them as transliterations. The semi-supervised system is able to learn them as non-transliterations. Table 5.14 shows a few examples of such word pairs.

Inconsistencies in the Gold Standard

There are several inconsistencies in the gold standard where my semi-supervised transliteration mining system correctly identifies a word pair as a transliteration but it is marked as a non-transliteration or vice versa. Consider the example of the English word "George" which is pronounced as "Jaarj" in Hindi. My semi-supervised system learns this as a non-transliteration but it is wrongly annotated as a transliteration in the gold standard.

English	Arabic	English	Arabic
Basrah	البصرة/Albasrah	Nasr	النصر/Alnasr
Kuwait	الكويت/Alkuwait	Riyadh	الرياض/Alriyadh

Table 5.15.: Examples of word pairs which are wrongly annotated as transliterations in the gold standard

There are a few word segmentation inconsistencies in the gold standard. The underscore "_" is used to separate the target language words in WIL phrases (consider English as the source language). This assumption is not followed for a few word pairs like "New_York" and "New_Mexico". In the reference data, these phrases are included with the "_" sign while all other phrases are word segmented on "_". I did not get these words in my training data as I tokenize English words on "_" and get words "New", "York" and "Mexico" from the above phrases.

Arabic nouns have an article "al" attached to them which is translated in English as "the". There are various cases in the training data where an English noun such as "Quran" is matched with an Arabic noun "alQuran". My semi-supervised mining system correctly classifies such cases as non-transliterations, but 24 of them are incorrectly annotated as transliterations in the gold standard. I did not correct this as I use the standard gold standard with the standard evaluation method, and therefore my system is unfairly penalized by the wrong annotation. Kahki et al. (2011) preprocessed such Arabic words and separated "al" from the noun "Quran" before mining. They report a match if the version of the Arabic word with "al" appears with the corresponding English word in the gold standard. Table 5.15 shows examples of word pairs which are wrongly annotated as transliterations in the gold standard.

Cognates

Sometimes a word pair differs by only one or two ending characters from a

English	Russian	English	Russian
Studio	Студия/Studiya	Catalonia	Каталонии/Katalonii
Estonia	Эстонии/Estonii	Monastery	Монастырь/Monastyr
Tartary	Тартария/Tartariya	Geography	География/Geografiya

Table 5.16.: Cognates from the English/Russian corpus extracted by my systems as transliteration pairs. None of them are correct transliteration pairs according to the gold standard

true transliteration. Such word pairs are very common in the WIL data. For example in the English/Russian training data, the Russian nouns are marked with case and when they are translated/transliterated to English, the case marking of the noun is omitted or translated because their English counterparts do not mark the case or translate it as a separate word. Often the Russian word differs only by the last character from a correct transliteration of the English word. Due to the large amount of such word pairs in the English/Russian data, my unsupervised transliteration mining system learns to delete the final case marking characters from the Russian words. It assigns a high transliteration probability to these word pairs and extracts them as transliterations. In the English/Hindi dataset, such word pairs are mostly English words that are borrowed in Hindi. Sometimes the base form of a word is borrowed from the language of origin and then it gets morphology from the target language like the word "calls" which is translated in Hindi as "callain". Table 5.16 shows some examples from the training data of English/Russian. All these pairs are mined by my systems as transliterations but they are non-transliterations in the gold standard.

5.7.8. Additional Experiments

In this section, I show the effect of my systems on different values of parameters. Later, I present an analysis on the English/Russian dataset. I use a few

variations of the seed data for the semi-supervised system and show that the seed data might contain a few wrong transliterations that are harmful for the semi-supervised mining system.

Variation of Parameters

I used a single setting of parameters for my transliteration mining systems. For the first iteration of EM, a uniform probability distribution is used and the prior probability of non-transliteration is set to 0.5. EM training maximizes the likelihood of the training data. After the training, the word pairs with a posterior probability of non-transliteration $p_{ntr}(e, f) = \lambda p_2(e_i, f_i)/p(e_i, f_i)$ less than 0.5 are selected as transliteration pairs. I do not have development data to find the best value for the threshold. The threshold value of 0.5 means that I am assuming that the system is able to classify transliteration and non-transliteration correctly. In this section, I show the effect of the variation of parameters on the results of my mining systems.

Unsupervised Transliteration Mining System

The unsupervised transliteration mining system is trained on the cross-product list. Table 5.17 shows the precision, recall and F-measure of the unsupervised mining system on different values (θ) of the posterior probability of non-transliteration. θ values close to zero tend to produce high precision lists of transliteration pairs as the system selects only highly probable transliteration pairs. The value of $\theta = 0.5$ works fine for three language pairs – English/Hindi, English/Arabic and English/Tamil and is either equal or close to the best F-measure the system is able to achieve. On the English/Russian dataset, the system achieves low precision at $\theta = 0.5$. The F-measure increases with the decrease in the value of θ. At $\theta = 0.1$, the system shows the best F-measure. I tried different values of $\theta < 0.1$. It balances precision and recall but the system could not achieve an F-measure better than 80.0%. The nature of the English/Russian dataset is different from others which is discussed under cognates in Section 5.7.7. I also tried different initial values of the prior probability

θ	English/Arabic P	English/Arabic R	English/Arabic F	English/Hindi P	English/Hindi R	English/Hindi F	English/Tamil P	English/Tamil R	English/Tamil F	English/Russian P	English/Russian R	English/Russian F
0.1	92.2	89.8	91.0	94.0	96.4	95.2	91.0	95.1	93.0	69.2	94.6	**80.0**
0.2	91.0	93.1	92.1	93.4	98.1	**95.7**	89.6	96.5	93.0	68.4	95.5	79.7
0.3	90.3	94.6	92.4	93.1	98.6	**95.7**	89.0	97.5	93.1	67.8	95.9	79.5
0.4	90.2	95.0	**92.6**	92.7	98.8	**95.7**	89.1	97.8	**93.2**	67.7	96.5	79.6
0.5	89.2	95.7	92.4	92.6	99.0	**95.7**	88.3	98.6	**93.2**	67.1	97.1	79.4
0.6	88.3	96.0	92.0	92.4	99.1	95.6	87.7	99.0	93.0	66.9	97.1	79.2
0.7	87.8	96.5	91.9	92.0	99.1	95.4	87.5	99.1	92.9	66.6	97.2	79.1
0.8	86.9	96.7	91.5	91.6	99.1	95.2	87.2	99.3	92.8	66.3	97.4	78.9
0.9	85.8	96.8	91.0	91.4	99.2	95.1	85.9	99.6	92.2	65.5	98.3	78.6

Table 5.17.: Results of the unigram unsupervised transliteration mining system on the posterior probability of non-transliteration less than θ

of non-transliteration to see its effect on the final mined transliteration pairs. The mining system is not very sensitive to λ. On different values of λ, it shows only a small variation in F-measure.

Semi-supervised Transliteration Mining System

The semi-supervised mining system consists of two steps. The first step trains using standard EM. The second step runs two E-steps separately on the labeled and unlabeled data. It estimates a probability distribution from the unlabeled data which is used as a backoff distribution for smoothing the estimates from the labeled data.

The smoothing technique used in the semi-supervised system is motivated from the Witten-Bell smoothing which is normally applied to integer frequencies obtained by simple counting (see Equation 5.17). I apply it to fractional counts obtained during EM training. I automatically choose the value of the smoothing parameter η_s as the number of different multigram types observed in the Viterbi alignment of the labeled data. This value works fine for all language pairs. Table 5.18 shows the variation of results on different values of η_s. The system is trained on the English/Hindi and English/Russian cross-product

list and the seed data. All results in Table 5.18 are calculated using $\theta = 0.5$. The $\eta_s = 0$ means that the model is not giving any weight to the unlabeled data and relies only on the smoothed labeled data probability distribution. The result of the system from $\eta_s = 0$ to $\eta_s = 50$ shows a gain in F-measure which leads to the conclusion that the unlabeled information is useful for the mining system. For higher values of η_s like 1800, the mining model gives more weight to the multigram probability distribution calculated from the unlabeled data which decreases precision and increases recall. At the automatically calculated value of $\eta_s = 165$ and $\eta_s = 178$ for English/Hindi and English/Russian, the mining system shows the best or close to best F-measure and shows a good balance in precision and recall.

The semi-supervised system uses the same initialization of the parameters as is used for the unsupervised system. The prior non-transliteration probability λ initializes to 0.5 and a threshold $\theta = 0.5$ is used on the posterior probability of non-transliteration $p_{ntr}(e, f)$ to decide between transliterations and non-transliterations. Table 5.19 shows the variation in the result of the semi-supervised system on different thresholds θ. It shows a similar behavior like the unsupervised system on different threshold. The threshold $\theta = 0.5$ works for all language pairs including English/Russian. The system is able to achieve more balanced precision and recall and higher F-measure than the unsupervised system.

Supervised Transliteration Mining System

The supervised transliteration mining system uses only the transliteration model and the labeled data for training. In the test mode, the non-transliteration model is used with the transliteration model to estimate the prior probability on the test data and to calculate the posterior probability of non-transliteration. The mining system later uses the posterior to separate out transliterations from non-transliterations. Table 5.20 and Table 5.21 show the results of English/Hindi, English/Arabic, English/Tamil and English/Russian supervised mining systems using the threshold θ on the posterior probability of non-transliteration. The value $\theta = 0.5$ does not work perfectly. However, the

η_s	English/Hindi			English/Russian		
	P	R	F	P	R	F
0	97.0	92.3	94.6	75.6	91.4	82.7
50	96.1	96.3	96.2	**74.6**	**94.2**	**83.3**
100	95.9	96.6	96.2	74.4	94.4	83.2
150	95.6	96.8	96.2	74.2	94.8	83.2
AUTO	**95.5**	**97.0**	**96.3**	74.0	94.9	83.1
200	95.5	97.1	96.3	74.0	94.9	83.1
250	95.3	97.1	96.2	73.7	94.9	82.9
300	95.1	97.1	96.1	73.5	94.9	82.8
400	95.1	97.1	96.1	73.4	95.0	82.8
500	95.1	97.3	96.2	73.2	95.3	82.8
600	94.9	97.3	96.1	72.9	95.5	82.7
700	94.8	97.5	96.1	72.6	95.6	82.5
800	94.7	97.5	96.0	72.4	95.6	82.4
900	94.7	97.5	96.0	72.4	95.6	82.4
1000	94.6	97.5	96.0	72.2	95.6	82.3
1100	94.6	97.7	96.1	72.1	95.7	82.3
1200	94.6	97.7	96.1	72.1	95.7	82.2
1300	94.6	97.7	96.1	71.2	95.9	81.7
1400	94.6	97.7	96.1	71.1	95.9	81.7
1500	94.6	97.7	96.1	71.1	95.9	81.7
1600	94.2	97.7	95.9	71.0	95.9	81.6
1700	94.2	97.7	95.9	70.9	95.9	81.6
1800	94.2	97.8	96.0	70.9	96.0	81.5

Table 5.18.: Results of the semi-supervised mining system trained on the English/Hindi and English/Russian data using different values of η_s. AUTO is the automatically calculated value of η_s as number of multigram tokens in the Viterbi of the seed data. The value of η_s is 165 and 178 for English/Hindi and English/Russian respectively

θ	English/Arabic P	English/Arabic R	English/Arabic F	English/Hindi P	English/Hindi R	English/Hindi F	English/Tamil P	English/Tamil R	English/Tamil F	English/Russian P	English/Russian R	English/Russian F
0.1	94.0	85.0	89.2	96.0	95.8	95.9	94.9	91.9	93.4	75.5	91.6	82.8
0.2	93.5	88.2	90.8	95.5	96.5	96.0	94.0	93.9	94.0	74.7	93.1	82.9
0.3	93.6	90.6	92.1	94.9	96.9	95.9	93.6	94.6	94.1	74.5	93.8	**83.1**
0.4	93.2	91.9	**92.5**	94.8	97.1	96.0	93.5	95.4	94.4	74.3	94.2	**83.1**
0.5	92.6	92.2	92.4	94.7	97.4	96.0	93.4	95.8	94.6	74.0	94.9	**83.1**
0.6	91.9	92.4	92.2	94.6	97.6	96.0	93.3	96.2	94.7	73.3	95.5	82.9
0.7	91.3	92.9	92.1	94.4	97.7	96.0	93.0	96.5	94.7	72.9	96.2	82.9
0.8	91.1	93.4	92.2	94.0	98.2	**96.1**	92.7	97.4	**95.0**	72.6	96.7	83.0
0.9	90.2	95.0	92.6	93.9	98.5	**96.1**	92.1	98.1	**95.0**	71.8	96.9	82.5

Table 5.19.: Results of the unigram semi-supervised mining system using different threshold θ on the posterior probability of non-transliteration

results of the system obtained at $\theta = 0.5$ are still close to the best F-measure the system is able to achieve.

English/Russian Dataset

Although my semi-supervised system is trained on labeled data, it also has low precision on the English/Russian dataset. The reason is that the seed data contains close transliteration pairs which are causing the mining system to learn them as transliterations. I propose a heuristic to filter out the wrong transliteration pair from the seed data which are likely to be close transliterations and then use the updated seed data in the training of the semi-supervised transliteration mining system. I did that in the following way:

I generate the Viterbi alignments of the seed data after the first step of the semi-supervised training and count the number of times a multigram occurs at the end of a Viterbi string. This gives a set of multigrams with counts where count is the number of times a multigram occurs at the end of the Viterbi alignment of the seed data. If a multigram count is less than a certain threshold, I consider them wrong transliteration pairs and drop from the seed data. The assumption here is that the seed data contains only few wrong pairs.

θ	English/Hindi P	R	F
0.01	96.0	93.4	94.7
0.02	95.9	94.0	**95.0**
0.03	95.9	94.1	**95.0**
0.04	95.9	94.1	**95.0**
0.05	95.8	94.2	**95.0**
0.06	95.6	94.3	94.9
0.07	95.5	94.4	94.9
0.08	95.4	94.4	94.9
0.09	95.3	94.4	94.8
0.1	95.2	94.4	94.8
0.2	95.0	94.5	94.7
0.5	94.2	94.6	94.4
0.8	93.3	94.9	94.1

θ	English/Arabic P	R	F
0.01	90.4	86.8	88.6
0.02	89.3	90.0	89.7
0.03	88.5	91.1	89.7
0.04	88.6	92.2	90.4
0.05	88.6	92.6	**90.6**
0.06	88.2	92.6	90.3
0.07	88.0	92.9	90.4
0.08	88.0	93.1	90.5
0.09	87.8	93.1	90.4
0.1	87.8	93.1	90.4
0.3	86.1	94.5	90.1
0.5	84.9	95.1	89.7
0.8	81.9	95.6	88.2

Table 5.20.: Results of the unigram supervised English/Hindi and English/Arabic mining system using different threshold θ on the posterior of non-transliteration

This heuristic would not work if there are many wrong pairs in the seed data. I use the new seed data for the training of the semi-supervised system. Table 5.22 shows the results of the semi-supervised system using the seed data filtered with different thresholds. All systems show a consistent increase in F-measure for an increase in threshold up to values 3 or 4 (afterwards F-score decreases). The removal of word pairs with infrequent endings help the system to achieve high precision. The trigram semi-supervised system at $threshold = 3$ shows the best F-measure of 86.1%. Threshold values of 2, 3, 4 reduce the size of the seed data by 53, 68 and 80 pairs respectively where the original seed data contains 1000 transliteration pairs. Due to data sparsity, the F-measure drops after $threshold > 4$.

θ	English/Tamil P	R	F
0.002	94.5	91.3	92.9
0.004	94.4	92	93.2
0.006	94	92.8	93.4
0.008	93.9	93.5	**93.7**
0.01	93.5	93.5	93.5
0.02	92.6	93.9	93.2
0.03	92.5	94.3	93.4
0.04	92.2	94.6	93.4
0.05	92.1	94.9	93.5
0.06	92.1	95.1	93.6
0.07	92.1	95.1	93.6
0.08	92.0	95.1	93.5
0.09	92.0	95.1	93.5
0.1	91.9	95.1	93.4
0.2	91.4	95.2	93.3
0.5	90.4	95.8	93
0.8	88.5	95.8	92.0

θ	English/Russian P	R	F
0.002	74.2	88.7	80.8
0.004	73.9	89.9	81.1
0.006	73.0	91.8	**81.4**
0.008	72.9	92.0	81.3
0.01	72.9	92.2	**81.4**
0.02	72.1	92.7	81.1
0.03	72.0	93.0	81.2
0.04	71.9	93.2	81.2
0.05	71.7	93.5	81.2
0.06	71.7	93.7	81.2
0.07	71.4	94.2	81.2
0.08	71.2	94.3	81.1
0.09	71.1	94.4	81.1
0.1	71.1	94.4	81.1
0.2	70.5	94.9	80.9
0.5	70.0	95.3	80.7
0.8	69.7	95.8	80.7

Table 5.21.: Result of the unigram supervised English/Tamil and English/Russian mining system using different threshold θ on the posterior of non-transliteration

ER	**Unigram** P	R	F	**Bigram** P	R	F	**Trigram** P	R	F
0	74.0	94.9	83.1	77.7	94.3	85.2	81.8	88.0	84.8
2	75.2	93.9	83.5	78.3	93.5	85.2	83.3	88.2	85.7
3	75.4	94.1	83.7	79.1	93.6	85.7	83.9	88.3	**86.1**
4	75.5	93.8	83.7	79.3	93.5	85.8	84.1	87.4	85.7

Table 5.22.: Result of the semi-supervised transliteration mining system for English/Russian using filtered seed data. 0, 2, 3, 4 is the value of the threshold on the count of the ending multigrams. 0 means no filtering.

5.8. Transliteration Mining Using Parallel Corpora

The percentage of transliteration pairs in the NEWS10 datasets is much larger than in a normal parallel corpus. I further check the effectiveness of my transliteration mining systems by evaluating them on parallel corpora with as few as 2% transliteration pairs. I conduct parallel corpus experiments using two language pairs, English/Hindi and English/Arabic. The English/Hindi corpus is from the shared task on word alignment organized as part of the ACL 2005 Workshop on Building and Using Parallel Texts (WA05) (Martin et al., 2005). For English/Arabic, I use 200,000 parallel sentences from the United Nations (UN) corpus (Eisele and Chen, 2010). I manually build a gold standard for the English/Hindi corpus and for the English/Arabic corpus by manually annotating a subset of the list of word pairs as either transliteration or non-transliteration.

5.8.1. Training

I follow the same procedure for creating the training data as described in Section 5.7.1. I align the parallel sentences using GIZA++ and refine them using the grow-diag-final-and heuristic. I extract a word-aligned list from the 1-to-1 alignments. Later, for every parallel source and target sentence, I make a pair of every source word that cooccurs with a target word and build a cross-product list.

The cross-product list is huge and it is computationally expensive to build a mining system on it. In order to keep the experiment computationally inexpensive, I train my transliteration mining systems on the word-aligned list and test them on the cross-product list. To compare the unsupervised model-based system with the heuristic-based system, I test it on the word-aligned list. The cross-product list is noisier than the word-aligned list but has almost 100% recall of transliteration pairs. The English-Hindi cross-product list has almost 130% more transliteration pairs (412 types) than the word-aligned list (180 types). I can not report these numbers on the English/Arabic cross-product

	Translit	Non-translit	Total
English/Hindi$_{\text{word-aligned}}$	180	2084	5612
English/Hindi$_{\text{cross-product}}$	412	12408	478443
English/Arabic$_{\text{word-aligned}}$	288	6639	178342
English/Arabic$_{\text{cross-product}}$	288	6639	26782146

Table 5.23.: Statistics of the word-aligned list and the cross-product list of the English/Hindi and English/Arabic parallel corpus

list since the English/Arabic gold standard is built on the word-aligned list. Table 5.23 shows the statistics of the word-aligned list and the cross-product list calculated using the gold standard of English/Hindi and English/Arabic. The "total" is the number of word pairs in the list. It is not equal to the sum of transliterations and non-transliterations in the list because the gold standard is only a subset of the training data.

5.8.2. Results

The unsupervised system is built on the word-aligned list. The semi-supervised system is trained on the word-aligned list and the English/Hindi and English/Arabic seed data provided by NEWS10. The supervised system is trained on the seed data only. All systems are then tested on the cross-product list. I initialize multigrams with a uniform probability distribution and set the prior probability of non-transliteration to 0.5. At test mode, the prior probability is reestimated on the test data. A threshold of 0.5 on the posterior probability of non-transliteration is used to decide between transliteration and non-transliteration.

I first train and test my unsupervised unigram model-based system on the word-aligned list and compare it with my heuristic-based system. Table 5.24 shows the results. On both languages, the model-based system shows high

	TP	FN	TN	FP	P	R	F
English/Hindi$_{\text{heuristic}}$	170	10	2039	45	79.1	94.4	86.1
English/Hindi$_{\text{model}}$	176	4	2034	50	77.9	97.8	86.7
English/Arabic$_{\text{heuristic}}$	197	91	6580	59	77.0	68.4	72.5
English/Arabic$_{\text{model}}$	288	0	6440	199	59.1	100.0	74.3

Table 5.24.: Transliteration mining results of the heuristic-based system and the unsupervised unigram model-based system trained and tested on the word-aligned list of the English/Hindi and English/Arabic parallel corpus

recall of up to 100% with lower precision and achieves 0.6% and 1.8% higher F-measure than the heuristic-based system.

Table 5.25 shows the transliteration mining results of unsupervised, semi-supervised and supervised systems trained on the word-aligned list and tested on the cross-product list. The unsupervised mining system for higher order performs poorly. It achieves very low precision and learns noise from the training data. The unigram semi-supervised system shows the best results of 85.6% F-measure with high precision and high recall. The higher order semi-supervised systems show a significant recall drop which resulted in a lower F-measure. The unigram supervised system performs better than the bigram and trigram supervised system (similar behavior to the unsupervised and semi-supervised system). The best F-measure achieved by the supervised system is 78.9% which is much lower than the best F-measure achieved by unsupervised and semi-supervised system. One reason is data sparsity. The seed data consists of only 1000 transliteration pairs. Secondly, the seed data and the training data used in the supervised systems are from different domains (Wikipedia and UN). Seed data extracted from the same domain is likely to work better, resulting in even higher scores than I have reported.

The mining systems show consistent behavior on the English/Arabic parallel corpus as well (see Table 5.26). The semi-supervised trigram system shows

English/Hindi	Unsupervised			Semi-supervised			Supervised		
	P	R	F	P	R	F	P	R	F
Unigram	72.9	**93.9**	82.1	**79.8**	92.2	**85.6**	80.4	77.4	78.9
Bigram	4.2	**97.6**	8.1	**88.4**	81.3	**84.7**	79.9	76.0	77.9
Trigram	4.9	**97.6**	9.3	**87.2**	62.6	72.9	72.6	77.2	**74.8**

Table 5.25.: Results of the unsupervised, semi-supervised and supervised mining systems trained on the word-aligned list and tested on the cross-product list of the English/Hindi parallel corpus. The bolded values show the best precision, recall and F-measure for the unigram, bigram and trigram systems

English/Arabic	Unsupervised			Semi-supervised			Supervised		
	P	R	F	P	R	F	P	R	F
Unigram	41.1	**100.0**	58.3	51.4	99.3	67.7	**54.1**	97.9	**69.7**
Bigram	4.2	**100.0**	8.1	**61.3**	98.3	**75.5**	61.0	98.3	75.3
Trigram	4.2	**100.0**	8.1	**63.8**	97.2	**77.0**	44.9	98.6	61.7

Table 5.26.: Results of the unsupervised, semi-supervised and supervised mining systems trained on the word-aligned list and tested on the cross-product list of the English/Arabic parallel corpus. The bolded values show the best precision, recall and F-measure for the unigram, bigram and trigram systems

the best F-measure. The parallel corpus contains only few transliterations. However, my unsupervised unigram system is able to mine transliterations with high recall. The higher order unsupervised system tends to learn noise from the training data. If labeled data is available, it is best to use the semi-supervised system.

The semi-supervised system performs better in correctly classifying close transliterations as non-transliteration. Table 5.27 shows a few word pairs from the English/Hindi experiment. These pairs are wrongly classified by the unigram unsupervised system and correctly classified by the unigram semi-supervised system. The unigram semi-supervised system is better than the unsupervised

English	Hindi
Also	एल/Al
Buses	बसों/Buson
Gas	जैसे/Gasain
Schools	स्कूलों/Schoolon
Trains	ट्रेनें/Trainain
'your	योर/Your

Table 5.27.: Examples of the English/Hindi close transliterations mined by the unigram unsupervised system and correctly classified as non-transliterations by the unigram semi-supervised system

system but there also a few close transliteration pairs which are wrongly classified by the unigram semi-supervised system. The bigram semi-supervised system uses the contextual information to correctly classify them. Table 5.28 shows a few word pairs from the English/Hindi experiment that are wrongly classified by the unigram semi-supervised system and correctly classified by the bigram semi-supervised system.

I looked into the errors made by my bigram semi-supervised system. The mined transliteration pairs still contain close transliterations. These are arguably better than other classes of non-transliterations like translations where source and target language words do not have a transliteration relationship between them. It is even possible that they provide transliteration information to system and it is very likely that they could be helpful when the available labeled data is very small.

English	Hindi
Appointments	अप्वाइंटमेंटों/Appointmemton
Brailled	ब्रैल/Brail
Chemicals	कैमीकल/Chemical
Consumers'	कंज़यूमर्ज़/Consumers
Companies	कंपनियाँ/Companiyan
Homes	होम/Home
Miles	मील/Mile
Parked	पार्क/Park
Volunteering	वालंटरी/Voluntary

Table 5.28.: Examples of the English/Hindi close transliterations mined by the unigram semi-supervised system and correctly classified as non-transliterations by the bigram semi-supervised system

5.9. Summary

I presented a novel model to automatically mine transliteration pairs. My approach is efficient, language pair independent (for alphabetic languages) and is flexible to use for unsupervised, semi-supervised and supervised transliteration mining. My best unsupervised system achieved high F-measure up to 95.7% and on three language pairs performed better than all supervised and semi-supervised systems that participated in NEWS10. On English/Russian dataset, it wrongly classified close transliterations as transliterations. The completely supervised system performed poorly due to data sparseness as labeled data is a small list of transliteration pairs.

The semi-supervised system resolved the limitations of my unsupervised and supervised systems. It has the best results and showed that labeled data helps the mining system to achieve high precision list of transliteration pairs and

unlabeled data helps to avoid data sparseness. The bigram semi-supervised system generally showed high F-measure and kept a good balance between precision and recall. The results showed that the semi-supervised system of order>2 focused on high precision and showed a significant drop in recall.

Transliteration information is helpful in major applications like machine translation and word alignment. In the next chapter, I incorporate the unsupervised transliteration mining model to unsupervised word alignment and show a significant improvement in the quality of word alignment.

5.10. Research Contribution

I presented a novel model for unsupervised transliteration mining. I also proposed a single framework that is able to mine transliteration pairs in an unsupervised, semi-supervised and supervised fashion.

In addition, I presented an analysis of the unsupervised, semi-supervised and supervised systems with varying ngram sizes and concluded the following points:

- If there is no labeled data available for a language pair, it is better to build a unigram unsupervised transliteration mining system. The higher order unsupervised systems tended to learn noise from the data and performed poorly.

- If there is some labeled data available, it is always better to build a semi-supervised system than a completely supervised or unsupervised system. The higher order semi-supervised systems learn the contextual information at the end of the word pairs and helped to achieve high precision. This is particularly helpful on training data with a large number of close transliterations.

6. Transliteration Mining to Improve Word Alignment

I presented an unsupervised model to automatically mine transliteration pairs from a parallel corpus. These transliteration pairs are helpful in major applications like cross language information retrieval, machine translation and word alignment. In this chapter, I show the applicability of transliteration in word alignment. The word alignment system works at word level and learns the translation correspondence in a parallel sentence. I integrate the unsupervised transliteration mining module into word alignment. So, the selection of a word pair to be a correct alignment is decided using both translation probabilities and transliteration probabilities. The new alignment method is fully unsupervised. The results show that transliteration information improves the quality of word alignment.

6.1. Introduction

The GIZA++ toolkit aligns parallel sentences at word level and builds a translation table (t-table) with the translation probability of every word pair. I define a transliteration sub-model and train it on the transliteration pairs mined by my unsupervised transliteration mining system using parallel corpora. I integrate it to the GIZA++ word aligner. The probability of a word pair is calculated as an interpolation of the transliteration probability and the translation probability stored in the t-table of the different alignment models used by

Algorithm 3 Estimation of transliteration probabilities, **e**-to-**f** direction

unfiltered data ←word-aligned list
filtered data ←transliteration pairs extracted using Algorithm 1
Train a transliteration system on the filtered data
for all e **do**
    $nbestTI(e) \leftarrow$ 10 best transliterations for e according to the transliteration system
    $cooc(e) \leftarrow$ set of all f that cooccur with e in a parallel sentence
    $candidateTI(e) \leftarrow cooc(e) \cup nbestTI(e)$
end for
for all f **do**
    $p_{moses}(f, e) \leftarrow$ joint transliteration probability of e and f according to the transliterator
    Calculate conditional transliteration probability $p_{ti}(f|e) \leftarrow \frac{p_{moses}(f,e)}{\sum_{f' \in CandidateTI(e)} p_{moses}(f',e)}$
end for

the GIZA++ aligner. The experiments show an improvement in both precision and recall over the baseline alignment.

6.2. Transliteration Module

Consider **e** is the source language and **f** is the target language. GIZA++ applies the IBM models (Brown et al., 1993) and the HMM model (Vogel et al., 1996) in both directions, i.e., e-to-f and f-to-e. The alignments are refined using the grow-diag-final-and heuristic (Koehn et al., 2003). GIZA++ generates a list of translation pairs with alignment probabilities, which is called the t-table. In this section, I present a method to estimate conditional transliteration probabilities in the e-to-f direction. An identical procedure provides the transliteration probabilities in the other direction. These transliteration probabilities are later interpolated with the conditional translation probabilities generated by GIZA++.

The transliteration module requires two things - a transliteration system and a candidate list that covers all possible word pairs that can be generated from the parallel corpus. For the transliteration system, I use the Moses toolkit. I take the filtered transliteration pairs extracted using my unsupervised transliteration mining system (Chapter 5) as training data for Moses. The characters of source words and target words in the filtered transliteration pairs are separated by space. So, every word pair behaves like a parallel sentence and every character of the word behaves like a word. The modified filtered transliteration pairs are used to train Moses.

For the candidate list, I first apply the transliteration system (Moses) to the e side of the list of word pairs (word-aligned list/cross-product list). For a source word, I generate the 10-best transliterations and call it *nbestTI(e)*. Then, I extract every f that cooccurs with e in a parallel sentence and add it to $nbestTI(e)$ which gives the list of candidate transliteration pairs $candidateTI(e)$.

The next step is to compute the transliteration probability of the word pairs in $candidateTI(e)$. I use the constraint decoding option of Moses to compute the joint probability of e and f. It divides the translation score of the best target sentence given a source sentence by the normalization factor. Moses with the constraint decoding option takes a pair of e and f, and calculates its joint probability. The joint probability is 0 if the decoder fails to produce the transliteration f for e.

GIZA++ generates conditional translation probabilities. In order to generate conditional transliteration probabilities, I use the sum of transliteration probabilities $\sum_{f' \in CandidateTI(e)} p_{moses}(f', e)$ as an approximation for the prior probability $p_{moses}(e) = \sum_{f'} p_{moses}(f', e)$ which is needed to convert the joint transliteration probability into a conditional probability. The conditional transliteration probability $p_{ti}(f|e)$ is defined as:

$$p_{ti}(f|e) = \frac{p_{moses}(f, e)}{\sum_{f'} p_{moses}(f', e)} \tag{6.1}$$

where $p_{moses}(f, e) = e^{score_{moses}(f,e)}$ and $(f, e) \in candidateTI$.

The complete procedure of the transliteration module is described in Algorithm 3.

6.3. Modified EM Training of the Word Alignment Models

In this section, I propose a method to modify the translation probabilities of the t-table by interpolating the translation counts with transliteration counts. The interpolation is done in both directions. In the following, I will only consider the **e**-to-**f** direction.

I combine the transliteration probabilities with the translation probabilities of the IBM models and the HMM model. The normal translation probability $p_{ta}(f|e)$ of the word alignment models is computed with relative frequency estimates:

$$p_{ta}(f|e) = \frac{f_{ta}(f,e)}{f_{ta}(e)} \tag{6.2}$$

I smooth the alignment frequencies by adding the transliteration probabilities weighted by the factor λ and get the following modified translation probabilities:

$$\hat{p}(f|e) = \frac{f_{ta}(f,e) + \lambda p_{ti}(f|e)}{f_{ta}(e) + \lambda} \tag{6.3}$$

where $f_{ta}(f,e) = p_{ta}(f|e)f_{ta}(e)$. $p_{ta}(f|e)$ is obtained from the original t-table of the alignment model. $f(e)$ is the total corpus frequency of $\mathbf{e}$. λ is the transliteration weight which is defined as the number of counts the transliteration model gets versus the translation model. It is optimized for every language pair (see Section 6.4). Apart from the definition of the weight λ, this smoothing method is equivalent to Witten-Bell smoothing.

I smooth after every iteration of the IBM models and the HMM model except the last iteration of each model. Algorithm 4 shows the smoothing for IBM Model4. IBM Model1 and the HMM model are smoothed in the same way. I

Algorithm 4 Interpolation with the IBM Model4, **e**-to-**f** direction

{I want to run four iterations of Model4}
$f(e) \leftarrow$ total frequency of e in the corpus
Run MGIZA++ for one iteration of Model4
$I \leftarrow 1$
while $I < 4$ **do**
    Look up $p_{ta}(f|e)$ in the t-table of Model4
    $f_{ta}(f, e) \leftarrow p_{ta}(f|e)f(e)$ for all (f, e)
    $\hat{p}(f|e) \leftarrow \frac{f_{ta}(f,e)+\lambda p_{ti}(f|e)}{f_{ta}(e)+\lambda}$ for all (f, e)
    Resume MGIZA++ training for 1 iteration using the modified t-table probabilities $\hat{p}(f|e)$
    $I \leftarrow I + 1$
end while

also apply Algorithm 3 and Algorithm 4 in the alignment direction **f** to **e**. The final alignments are generated using the grow-diag-final-and heuristic (Koehn et al., 2003).

6.4. Evaluation

In this section, I present the results of my word alignment model. I use MGIZA++ (Gao and Vogel, 2008) for incorporation. It is an extension of GIZA++ with the additional ability to resume training from any model rather than starting with Model1. Here, I use the term GIZA++ instead of MGIZA++.

6.4.1. Training Data

I evaluate my word alignment model on English/Hindi and English/Arabic. The English/Hindi corpus available from ACL 2005 Workshop on Building and Using Parallel Texts (WA05) (Martin et al., 2005) consists of training, development and test data. For English/Arabic, I use the parallel corpus from the United Nations (Eisele and Chen, 2010). I randomly pick 200,000 par-

allel sentences from the year 2000. For the development and test data for English/Arabic, I use manually created gold standard word alignments for 155 sentences, available from Fraser and Marcu (2007). I use 50 sentences for development and 105 sentences for test.

For the transliteration module, I generate a word-aligned list and a cross-product list for both English/Arabic and English/Hindi using the method described in Section 5.7.1. I build two versions of the transliteration system – one is trained and tested on the word-aligned list and the other is trained on the word-aligned list but tested on the cross-product list. I compare the effect of using the different lists for the transliteration module on the English/Hindi word alignment in the next section.

6.4.2. Experiments

I align the data sets using GIZA++ (Och and Ney, 2003). It runs with 5 iterations of Model1, 4 iterations of HMM and 4 iterations of Model4 and refines the alignments using the grow-diag-final-and heuristic (Koehn et al., 2003). I obtain the baseline F-measure by comparing the alignments of the test corpus with the gold standard alignments.

In my word alignment model, I also use GIZA++ with 5 iterations of Model1, 4 iterations of HMM and 4 iterations of Model4. I interpolate translation and transliteration probabilities at different iterations (and different combinations of iterations) of the three models and always observe an improvement in alignment quality. For the final experiments, I interpolate at every iteration of the IBM models and the HMM model except the last iteration of every model where I could not interpolate for technical reasons.[1] Algorithm 4 shows the interpolation of the transliteration probabilities with IBM Model4. I used the

[1] I had problems in resuming MGIZA++ training when training was supposed to continue from a different model, such as if I stopped after the 5th iteration of Model1 and then tried to resume MGIZA++ from the first iteration of the HMM model. In this case, I ran the 5th iteration of Model1, then the first iteration of the HMM and only then stopped for interpolation; so I did not interpolate in just those iterations of training where I was transitioning from one model to the next.

	P_b	R_b	F_b	P_{ti}	R_{ti}	F_{ti}
EH	49.1	48.5	51.2	58.5	52.8	55.5
EA	50.8	49.9	50.4	56.1	55.4	55.7

Table 6.1.: Word alignment results on the test data of English/Hindi (EH) and English/Arabic (EA) where P_b is the precision of baseline GIZA++ and P_{ti} is the precision of my word alignment system

same procedure with IBM Model1 and the HMM model. The parameter λ is optimized on development data for every language pair. The word alignment system is not very sensitive to λ. Any λ in the range between 20 and 100 works fine for all language pairs. The optimization helps to maximize the improvement in word alignment quality. For my experiments, I use $\lambda = 35$. For the transliteration system, I train the unsupervised transliteration mining system on the word-aligned list and mine transliteration pairs from the cross-product list. On test data, I achieve an improvement of approximately 9.6% and 5.3% in precision and 4.3% and 5.5% in recall on English/Hindi and English/Arabic[2] word alignment, respectively. Table 6.1 shows the scores of the baseline model and my word alignment model.

I compare my English/Hindi word alignment results with the systems presented at WA05. There are three systems, one limited and two un-limited, which participated in the English/Hindi task. I outperformed the limited system and one un-limited system. The other un-limited system uses language dependent information like transliteration similarity and dictionary lookup. It's results are not directly comparable with my results.

[2]The English/Arabic results reported here are obtained by interpolating for every iteration of Model1 only because for this pair t-table probabilities go to infinity when interpolation is done for Model4 and HMM model.

λ	P	R	F
5	56.8	50.4	53.4
10	59.5	50.6	54.7
15	61.0	50.8	55.4
20	61.8	52.3	56.6
25	60.5	52.1	56.0
30	62.1	53.1	57.3
35	62.4	53.1	**57.4**
40	59.6	51.5	55.3
45	59.2	50.6	54.6
50	59.4	51.0	54.9
55	60.5	51.5	55.6
60	60.0	51.0	55.2
65	60.9	51.7	55.9
70	61.0	51.7	56.0
75	61.1	52.5	56.5
80	61.3	52.5	56.5
85	60.9	51.7	55.9
90	60.3	51.5	55.5
95	61.0	51.7	56.0

Table 6.2.: Lambda optimization on the gold standard development set of English/Hindi. The transliteration module is trained and tested on the word-aligned list

6.4.3. Additional Experiments

The interpolation parameter is optimized on the development set. A value between 20 to 100 works fine. Table 6.2 shows the effect of varying λ on the development set of English/Hindi. The transliteration module is trained and tested on the word-aligned list.

In the word alignment experiments above, I trained the transliteration system on the list of transliteration pairs extracted from the word-aligned list. I later call this word alignment system $Alignment_{WA}$. I build another version of

	P	R	F
Baseline	49.2	44.2	46.6
$Alignment_{WA}$	62.4	53.1	57.4
$Alignment_{CP}$	61.2	50.6	55.4

Table 6.3.: Word alignment results on the development data of English/Hindi. *Baseline* is the alignment result of GIZA++, $Alignment_{WA}$ is the word alignment system where the transliteration module is built on the transliteration pairs extracted from the word-aligned list and in $Alignment_{CP}$, they are extracted from the cross-product list

the word alignment system in which the transliteration module is built using the list of transliteration pairs extracted from the cross-product list. I call this system $Alignment_{CP}$ for later reference. Table 6.3 compares the results of these two word alignment systems. $Alignment_{WA}$ shows 0.1% better precision and 2.5% better recall than $Alignment_{CP}$. It achieves an overall F-measure gain of 1.5% over $Alignment_{CP}$. The mined list of transliteration pairs extracted from the word-aligned system is very clean and contains mostly transliterations and close transliterations. The close transliterations are not harmful for word alignment as the alignment of these words are correct. The transliteration pairs extracted from the mining system using the cross-product list are noisy. They contain transliterations, close transliterations and wrong alignment pairs. These are the word pairs where one word is a frequent word like a Hindi case marker and is attached with almost every other word in the cross-product list. These word pairs are also mined as transliteration pairs by the system. For word alignment, they are wrong alignments and are harmful for the system.

6.5. Summary

I presented an unsupervised word alignment model that uses transliteration information to improve the quality of word alignment. The alignment model

has two components, the translation model and the transliteration model. I used the GIZA++ word aligner which provides the translation model probabilities. For transliteration probabilities, I trained my unsupervised mining system on the training data and extracted a list of transliteration pairs. I built a transliteration system using Moses on the extracted transliteration pairs and calculated the transliteration probabilities. I interpolated the transliteration and non-transliteration models inside of GIZA++. The new word alignment system showed a large improvement in the quality of word alignment.

6.6. Research Contribution

I presented an unsupervised word aligner that uses both translation and transliteration information and showed that it improves the quality of word alignment.

7. Contributions and Future Work

7.1. Conclusion

I revise the contributions that I claimed in Section 1.3 and discuss the shortcomings in Section 7.3. Section 7.4 presents the suggested future work.

7.2. Contributions

7.2.1. Theoretical Contributions

The following are my theoretical contributions:

- **Transliteration in machine translation:** I have presented a novel way to integrate transliterations into machine translation. I have showed that for language pairs with significant vocabulary overlap, transliteration can be effective in machine translation for more than just translating OOV words. I have used transliteration as a tool for disambiguation of homonyms which can be translated or transliterated or transliterated differently based on different contexts.

- **The first unsupervised algorithm for transliteration mining:** I have presented the first unsupervised algorithm for transliteration mining. It is an iterative algorithm which trains on unlabeled data. In every iteration, it filters out a few word pairs which are less likely to be transliterations. The algorithm stops based on a stopping criterion and returns

a list of transliteration pairs. The unsupervised mining system is evaluated on parallel corpora. I showed that it has competitive results with the state-of-the-art semi-supervised and supervised systems.

- **A novel model for unsupervised transliteration mining:** I have proposed a novel model for unsupervised transliteration mining defined as an interpolation of a transliteration sub-model and a non-transliteration sub-model. The unsupervised mining model is very efficient and language pair independent. It models the unlabeled data and mines transliteration pairs from it. The results showed that it performs better than most of the semi-supervised and supervised systems.

  A framework for transliteration mining: I have presented the first general framework for transliteration mining that uses a single model for all – unsupervised, semi-supervised and supervised learning. I evaluated my mining system using different learning models and showed that the semi-supervised system is better than the unsupervised and supervised systems. This shows that both labeled and unlabeled data are important for learning. The labeled data helps to achieve high precision while the unlabeled data helps to avoid the problem of data sparseness. A combination of them balances out these two effects.

- **An application to word alignment:** I have incorporated the unsupervised transliteration mining module into the unsupervised word alignment toolkit GIZA++. The new alignment system is also unsupervised. I showed that applying transliteration mining and transliteration modeling improves word alignment in terms of both precision and recall when compared against gold standard word alignments.

7.2.2. Resource Contributions

- **Gold standard for transliteration mining:** I have built a gold standard for the English/Hindi and English/Arabic transliteration mining experiments. I annotated a subset of the word pairs extracted from the

word-aligned corpora as either transliteration or non-transliteration. The gold standard has been made freely available.

- **A transliteration mining tool:** I have implemented the model-based transliteration mining system. The system has four major modules – character aligner, unsupervised mining, semi-supervised mining and supervised mining. The tool provides the options to manually set the parameters instead of using the automatically calculated values of the parameters. The system is freely available.

7.3. Shortcomings

The unsupervised mining system has the limitation of learning only unigram multigram units. Higher order multigram units should help to learn a wide variety of transliteration phenomena which the unigram multigram units are unable to learn. For example, consider the English character sequence *ph* and its equivalent transliteration character in Urdu *f*. The unigram training aligns *p* to *f* and aligns *h* to ∅ while the higher order training produces a single alignment *ph*→*f*. The alignment *p*→*f* and *h*→ ∅ learned by the unigram model are wrong. If these alignments are frequent in the training data, the unsupervised mining system would learn to align *p* to *f* and to delete *h* with a high probability.

The unsupervised mining system on the English/Hindi cross-product list observed the problem of high deletion probability and wrongly classified a few non-transliteration pairs. These are the pairs where Hindi word is a small string consisted of 2-3 characters and English word is a large string. However, the English word contains the transliteration of the Hindi word as its substring, like the word pair Keeper/पर (par).[1] Due to the noisy nature of the training data, the unsupervised transliteration mining system learns to delete at high probability.

[1]The Hindi word contains only two characters. Its romanization is represented by three characters.

For the above word pair, it aligns English p to Hindi p and English r to Hindi r and delete other English characters k and e at high probability. The word pair gets a high transliteration probability and is incorrectly mined as a transliteration pair.

I may modify the system to learn m-to-n alignments. However, the training data is very noisy. The unsupervised system memorizes larger noisy units from the unlabeled data and performs poorly (see Section 5.7.5 for results on the higher order models). This is a standard problem of the approaches based on maximum likelihood training. They tend to overfit the training data. A solution to this problem is to use the Bayesian approach as an alternative to maximum likelihood training. It avoids the overfitting of the data by preferring smaller character alignments. Section 7.4.6 talks about using the Bayesian approach in detail.

Non-transliteration data consists of translations, misalignments, close transliterations, numbers[2] and word pairs for which both source and target language words are identical (identical words like English to English). The current model considers them under one class and models them as randomly seeing source and target language characters together. But, aside from translations and misalignments, most of these word pairs do have some character relationship between them. I call them *regular non-transliteration categories*. The non-transliteration model is not capturing this information and is not a true representative of all types of non-transliterations. The regular non-transliteration categories can cause problems for the mining system when they occur frequently in the training data. For example, suppose the percentage of identical words is larger than the percentage of transliterations in the training data. This would mislead the transliteration model and it will learn identical-words as transliterations.[3] In Section 7.4.3, I talk about dividing the non-transliteration model into more than one smaller models. This would separately model differ-

[2]Numbers are classified as non-transliterations according to the NEWS10 guidelines.

[3]Currently, I am automatically removing identical-words in a pre-processing step but this problem holds also for other categories like numbers.

ent kinds of non-transliterations and would better handle the noise.

The unsupervised mining model has the problem of wrongly classifying close-transliterations as transliterations. The decision on whether a pair is transliteration or non-transliteration is based on the posterior probability of non-transliteration. For the English/Russian dataset where the problem of close transliteration is severe, there is a big difference between the value of precision and recall. The system achieves high recall (97.1%) and low precision (67.2%). For some applications, it would be better to choose a different decision boundary that balances out precision and recall. Section 7.4.4 talks about a solution to this problem.

7.4. Future Work

7.4.1. Training using Noisier Data

I have experimented using the NEWS10 dataset and parallel corpora. The NEWS10 dataset contains title of Wikipedia pages written in different languages. The percentage of transliterations in the dataset are about 14-18% depending on the language pair. The word-aligned list extracted from the parallel corpora contains approximately 4-8% transliteration pairs. The unsupervised transliteration mining of the NEWS10 dataset is easier than the word-aligned list of parallel corpora as it contains higher percentage of transliteration pairs. However, my unsupervised system performed well on both datasets. An interesting experiment for the unsupervised system would be to see the variation in its performance with the decrease in the percentage of transliteration pairs in the training data. The cross-product list extracted from the English/Arabic parallel corpus contains about 2% transliteration pairs and is the noisiest available data. I could not train my system on it as it was computationally expensive. I can artificially generate noise in the data and test my mining system on it but it might not reflect the true behavior of the system. The natural noise in the data contains also close transliterations which are not transliterations but provide useful information to the unsupervised system. I would like to

improve the computational complexity of the software and would run it on the cross-product list of the English/Arabic parallel corpus.

7.4.2. Comparable Corpora

Another interesting experiment would be to apply unsupervised transliteration mining to the word pairs extracted from comparable corpora. This data would be noisier than the cross-product list (described in Section 7.4.1) and might contain very few transliteration pairs.

7.4.3. Various Non-transliteration Models

The non-transliteration category contains various kinds of noise as described in Section 7.3. The current non-transliteration model does not handle them separately. A modeling solution to this problem is to divide the non-transliteration model into several sub-models, one for each kind of noise. In this way, every type can be modeled separately. For identical words, this would introduce two models – one to handle source language identical-words and other to handle target language identical-words. The training data for the source language identical-words could be all source language words paired with themselves.

Close transliterations are hard to model separately without having labeled data for their training. In weakly supervised setting, one may include (very limited) interactive user feedback that helps in identifying a few close transliterations from the unlabeled data.

7.4.4. Back Transliteration

In order to solve the low precision problem of my unsupervised system, I propose a different decision boundary that balances out precision and recall in case of English/Russian and increases precision from recall for other language pairs. The idea is to use a decoder to transliterate source words and target words of test data and then compare them with the original words under

some constraints. For a unigram model, it is very less likely that the decoder transliterates a word to a close transliteration rather than a transliteration.

The procedure is explained as follows:

For a word pair (e, f) of the test set, use a decoder to produce transliteration of the word e, say f'. Repeat the similar step for word f and generate its transliteration, say e'. Now, there are three pairs (e, f), (e, f') and (e', f). One way to compare the generated transliteration to the original word is to calculate their edit distance. If they differ by more than a certain threshold, classify the original word pair (e, f) as non-transliteration. Another way is to compare the probabilities of the word pairs. The decoder provides the probability of the most likely sequence it generates. For the original word pair (e, f), one can use constraint decoding to get its probability according to the model. If these probabilities are close to each other and have a difference less than a certain threshold, classify the word pair as a transliteration pair.

7.4.5. Transliteration Mining involving Non-alphabetic Languages

I tested my transliteration mining system only on alphabetic languages. A further step is to adapt the model to language pairs where one language is alphabetic and other language is non-alphabetic such as English/Japanese. Non-alphabetic languages have logographic writing systems. Each single unit (character) may represent multiple characters or words of alphabetic languages. Solving the transliteration mining problem requires learning one-to-many and/or many-to-one character alignments.

I have built higher ngram order unsupervised mining systems in Section 5.7.5. Their results showed that the unsupervised system starts incorrectly memorizing larger units from the training data when moved from unigram to higher order causing poor performance. In Section 7.4.6, I describe a Bayesian approach to transliteration mining. It should be able to learn larger multigrams without memorizing the noise from the data.

7.4.6. Bayesian Approach

My transliteration mining model is based on maximum likelihood training (ML). The problem with ML training is that it overfits the training data and gives larger units too much weight, which hurts generalization. Finch and Sumita (2010) present a Bayesian approach to generate m-to-n character alignments for transliteration and compare their results with a ML based m-to-n aligner. The Bayesian approach prefers frequent character alignments and assigns a lower probability to infrequent character alignments. In this way, the model penalizes alignments of larger character units and relies more on unigram character alignments.

We could extend our model to induce m-to-n alignments and modify the training to use the Bayesian approach instead of ML training. In this way, the model is able to learn reliable larger multigrams without memorizing the noise from the training data. This model is different from Finch and Sumita (2010) as it still models every word pair using both transliteration and non-transliteration sub-models. This approach should also work well when used with non-alphabetic languages.

7.4.7. Incorporate Unsupervised Mining Model to NLP Applications

Word Alignment

I used transliteration information in word alignment and showed that it improves the performance of the systems. I mined transliteration pairs in a preprocessing step and calculated the transliteration probabilities. During word alignment, I interpolated the translation probabilities of t-table with the transliteration probabilities. Here, the t-table is updated after every iteration of EM. However, the transliteration probabilities are fixed and calculated from the transliteration pairs extracted from the cross-product list.

A better way to add transliteration information to alignment is to incorporate the transliteration mining module inside of EM. In this way, the translit-

eration probabilities can also improve for every iteration of EM like the t-table probabilities.

The procedure is as follows:

After every iteration of EM, generate a list of word pairs from the current alignment of the aligner, mine transliteration pairs from it and calculate transliteration probabilities. Then interpolate them to t-table probabilities and resume the word aligner to the next iteration. Here, the transliteration mining system runs after every iteration of EM on an updated list of word pairs and the transliteration probabilities are also calculated for every iteration. The t-table probabilities and the the transliteration probabilities are improving for every iteration of EM. This would lead to a maximum improvement in the word alignment quality.

Machine Translation

Transliteration can be used in machine translation at two levels – at the time of word alignment and in decoding to generate transliteration of OOV words. I have separately tested the effect of transliteration in the word alignment task (see Chapter 6) and in the machine translation task (see Chapter 3). Using it at both levels in one system might be very helpful in improving the overall quality of translation.

Bibliography

Steven Abney. *Semisupervised Learning for Computational Linguistics*. Chapman & Hall/CRC, 1st edition, 2007.

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

Waleed Ammar, Chris Dyer, and Noah Smith. Transliteration by sequence labeling with lattice encodings and reranking. In *Proceedings of the 4th Named Entity Workshop*, Jeju, Korea, 2012.

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

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2), 1993.

Kareem Darwish. Transliteration mining with phonetic conflation and iterative training. In *Proceedings of the 2010 Named Entities Workshop*, Uppsala, Sweden, 2010.

Sabine Deligne and Frédéric Bimbot. Language modeling by variable length sequences : Theoretical formulation and evaluation of multigrams. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, Los Alamitos, CA, USA, 1995.

Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

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

Andreas Eisele and Yu Chen. MultiUN: A multilingual corpus from United Nation documents. In *Proceedings of the Seventh conference on International Language Resources and Evaluation*, Valletta, Malta, 2010.

Asif Ekbal, Sudip Kumar Naskar, and Sivaji Bandyopadhyay. A modified joint source-channel model for transliteration. In *Proceedings of the International Conference on Computational Linguistics and Association for Computational Linguistics*, Sydney, Australia, 2006.

Andrew Finch and Eiichiro Sumita. Phrase-based machine transliteration. In *Proceedings of the Workshop on Technologies and Corpora for Asia-Pacific Speech Translation*, Hyderabad, India, 2008.

Andrew Finch and Eiichiro Sumita. A bayesian model of bilingual segmentation for transliteration. In *Proceedings of the seventh International Workshop on Spoken Language Translation*, Paris, France, 2010.

Alexander Fraser and Daniel Marcu. Measuring word alignment quality for statistical machine translation. *Computational Linguistics*, 33(3), 2007.

William A. Gale and Kenneth W. Church. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1), 1993.

Qin Gao and Stephan Vogel. Parallel implementations of word alignment tool. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, Columbus, Ohio, 2008.

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

Xiaodong He. Using word dependent transition models in hmm based word alignment for statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, Prague, Czech Republic, 2007.

Ulf Hermjakob, Kevin Knight, and Hal Daumé III. Name translation in statistical machine translation - learning when to transliterate. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Columbus, Ohio, 2008.

Fei Huang. *Multilingual named entity extraction and translation from text and speech*. PhD thesis, Language Technology Institute, Carnegie Mellon University, 2005.

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

Sittichai Jiampojamarn, Grzegorz Kondrak, and Tarek Sherif. Applying many-to-many alignments and hidden markov models to letter-to-phoneme conversion. In *Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference*, Rochester, New York, 2007.

Sittichai Jiampojamarn, Aditya Bhargava, Qing Dou, Kenneth Dwyer, and Grzegorz Kondrak. DirecTL: a language independent approach to transliteration. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, Suntec, Singapore, 2009.

Sittichai Jiampojamarn, Kenneth Dwyer, Shane Bergsma, Aditya Bhargava, Qing Dou, Mi-Young Kim, and Grzegorz Kondrak. Transliteration genera-

tion and mining with limited training resources. In *Proceedings of the 2010 Named Entities Workshop*, Uppsala, Sweden, 2010.

Ali El Kahki, Kareem Darwish, Ahmed Saad El Din, Mohamed Abd El-Wahab, Ahmed Hefny, and Waleed Ammar. Improved transliteration mining using graph reinforcement. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Edinburgh, UK, 2011.

Mehdi M. Kashani, Eric Joanis, Roland Kuhn, George Foster, and Fred Popowich. Integration of an Arabic transliteration module into a statistical machine translation system. In *Proceedings of the Second Workshop on Statistical Machine Translation*, Prague, Czech Republic, 2007a.

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

Alexandre Klementiev and Dan Roth. Weakly supervised named entity transliteration and discovery from multilingual comparable corpora. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, Morristown, NJ, USA, 2006.

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

Philipp Koehn. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *Conference of the Association for Machine Translation in the Americas*, Washington DC, 2004a.

Philipp Koehn. Statistical significance tests for machine translation evaluation. In Dekang Lin and Dekai Wu, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, 2004b.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, The Edinburgh Building, Shaftsbury Road, Cambridge, 2010.

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

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Demonstration Program*, Prague, Czech Republic, 2007.

A Kumaran, Mitesh M. Khapra, and Haizhou Li. Whitepaper of NEWS 2010 shared task on transliteration mining. In *Proceedings of the 2010 Named Entities Workshop*, Uppsala, Sweden, 2010.

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

Wesley Mackay and Grzegorz Kondrak. Computing word similarity and identifying cognates with pair hidden markov models. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, Stroudsburg, PA, USA, 2005.

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

Joel Martin, Rada Mihalcea, and Ted Pedersen. Word alignment for languages with scarce resources. In *ParaText '05: Proceedings of the Association for*

Computational Linguistics Workshop on Building and Using Parallel Texts, Morristown, NJ, USA, 2005.

Robert C. Moore. Fast and accurate sentence alignment of bilingual corpora. In *Conference of the Association for Machine Translation in the Americas*, Tiburon, California, 2002.

Peter Nabende. Transliteration system using pair HMM with weighted fsts. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, NEWS '09, Suntec, Singapore, 2009.

Peter Nabende. Mining transliterations from Wikipedia using Pair HMMs. In *Proceedings of the 2010 Named Entities Workshop*, Uppsala, Sweden, 2010.

Sara Noeman. Language independent transliteration system using phrase based SMT approach on substrings. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, NEWS '09, Suntec, Singapore, 2009.

Sara Noeman and Amgad Madkour. Language independent transliteration mining system using finite state automata framework. In *Proceedings of the 2010 Named Entities Workshop*, Uppsala, Sweden, 2010.

Franz J. Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan, 2003.

Franz J. Och and Hermann Ney. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, 2000.

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

Franz J. Och, Christoph Tillmann, and Hermann Ney. Improved alignment models for statistical machine translation. In *Joint SIGDAT Conference on*

Empirical Methods in Natural Language Processing and Very Large Corpora, University of Maryland, College Park, MD, 1999.

Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 1990.

Taraka Rama and Karthik Gali. Modeling machine transliteration as a phrase based statistical machine translation problem. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, Morristown, NJ, USA, 2009.

Eric Sven Ristad and Peter N. Yianilos. Learning string-edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5), 1998.

Hassan Sajjad, Nadir Durrani, Helmut Schmid, and Alexander Fraser. Comparing two techniques for learning transliteration models using a parallel corpus. In *Proceedings of the 5th International Joint Conference on Natural Language Processing*, Chiang Mai, Thailand, 2011a.

Hassan Sajjad, Alexander Fraser, and Helmut Schmid. 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, 2011b.

Hassan Sajjad, Alexander Fraser, and Helmut Schmid. A statistical model for unsupervised and semi-supervised transliteration mining. In *Proceedings of the 50th Annual Conference of the Association for Computational Linguistics*, Jeju, Korea, 2012.

K Saravanan, Raghavendra Udupa, and A Kumaran. Improving cross-language information retrieval by transliteration generation and mining. *LNCS volume on Forum for Information Retrieval Evaluation proceedings*, 2010.

Tarek Sherif and Grzegorz Kondrak. Bootstrapping a stochastic transducer for Arabic-English transliteration extraction. In *Proceedings of the 45th Annual*

Meeting of the Association for Computational Linguistics, Prague, Czech Republic, 2007a.

Tarek Sherif and Grzegorz Kondrak. Substring-based transliteration. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Prague, Czech Republic, June 2007b.

Praneeth Shishtla, V. Surya Ganesh, Sethuramalingam Subramaniam, and Vasudeva Varma. A language-independent transliteration schema using character aligned models at NEWS 2009. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, Morristown, NJ, USA, 2009.

R. Mahesh K. Sinha. Developing English-Urdu machine translation via Hindi. In *Third Workshop on Computational Approaches to Arabic Script-based Languages, MT Summit XII*, Ottawa, Canada, 2009.

Richard Sproat, Tao Tao, and ChengXiang Zhai. Named entity transliteration with comparable corpora. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia, 2006.

Bonnie G. Stalls and Kevin Knight. Translating names and technical terms in Arabic text. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - The Workshop on Computational Approaches to Semitic Languages*, Montreal, Quebec, Canada, 1998.

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

Tao Tao, Su-Yoon Yoon, Andrew Fister, Richard Sproat, and ChengXiang Zhai. Unsupervised named entity transliteration using temporal and phonetic correlation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, 2006.

Joerg Tiedemann and Peter Nabende. Translating transliterations. *international journal of computing and ICT research*, 3(1), 2009.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *16th International Conference on Computational Linguistics*, Copenhagen, Denmark, 1996.

Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4), 1991.

Bing Zhao, Nguyen Bach, Ian Lane, and Stephan Vogel. A log-linear block transliteration model based on bi-stream HMMs. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, Rochester, New York, 2007.

A. Transliteration Mining Software

The transliteration mining software consists of four modules – character aligner, unsupervised transliteration miner, semi-supervised transliteration miner and supervised transliteration miner. The software uses two kinds of data – unlabelled data and labelled data. The unlabelled data consists of a list of word pairs which are either transliterations or non-transliterations. The labelled data is a list of transliteration pairs.

The software is released under a Creative Commons license. I request a citation of the following paper if the software is used in a publication.

Sajjad, Hassan; Fraser, Alexander; Schmid, Helmut (2012). A statistical model for unsupervised and semi-supervised transliteration mining. Proceedings of the 50th Annual Conference of the Association for Computational Linguistics. Jeju, Korea.

A.1. Modules

A.1.1. Character Aligner

The character aligner takes a list of word pairs for training. In the test mode, the trained model is applied to test data. The output of the aligner is the test data aligned at character level. The generated multigrams (character alignments) are restricted to 0–1,1–1,1–0 i.e. zero or one character on the source and target side. Following is the command to run the aligner.

java -Xmx5g -Dfile.encoding=UTF-8 -jar multigrams.jar -aligner -train trainFile -test testFile

If no test is specified, trainFile will be taken for test.

A.1.2. Unsupervised Transliteration Miner

The unsupervised transliteration miner trains on the unlabelled data. The training supports an ngram order up to trigram. The trained model is then used to mine transliteration pairs from a test set. A sample executable command is as follows:

java -Xmx5g -Dfile.encoding=UTF-8 -jar multigrams.jar -unsupervised -train trainFile -test testFile

If no test is specified, trainFile will be taken for test.

A.1.3. Semi-supervised Transliteration Miner

The semi-supervised transliteration miner takes the unlabelled data and the labelled data for training. The training supports an ngram order up to trigram. In the test mode, the trained model is applied to testFile to mine transliteration pairs.

java -Xmx5g -Dfile.encoding=UTF-8 -jar multigrams.jar -semisupervised -train trainFile -test testFile -seed seedFile

If no test is specified, trainFile will be taken for test.

A.1.4. Supervised Transliteration Miner

The supervised transliteration miner trains on the labelled data and mines transliteration pairs from the test data. The training supports an ngram order up to trigram. Following is the command to run the supervised miner.

java -Xmx5g -Dfile.encoding=UTF-8 -jar multigrams.jar -supervised -train trainFile -test testFile

trainFile is the labelled data. testFile is mandatory for the supervised miner.

A.2. Instructions to Run

The system does not require any installation. **multigram.jar** is the main executable file. Following is the basic command to run the miner. Table A.1 describes the options provided with the executable.

java -Xmx5g -Dfile.encoding=UTF-8 -jar multigrams.jar [-aligner | -unsupervised | -semisupervised | -supervised] -train trainFile -test testFile [-seed seedFile]

The aligner, unsupervised and supervised modules require a training file which is specified with the -train option. If there is no test file specified, the train file will be taken for testing. The semi-supervised module additionally requires a seed file for training.

Option	Description
-Xmx5g	It is the memory required by the software to load and process input data. The memory requirement varies with the size of the train and test data
-Dfile.encoding=UTF-8	This specifies the encoding of the input data in UTF-8
-aligner	Invokes the alignment module. It requires a training file specified with -train option and character aligns the test data based on the trained model
-unsupervised	Runs unsupervised transliteration miner. It requires a training file specified with -train option

-semisupervised	Runs semi-supervised transliteration miner. It requires a training file and a seed file specified with -train and -seed options respectively
-supervised	Runs supervised transliteration miner. It requires a training file specified with -train option and a test file to mine transliteration pairs
-train trainFile	format is a word pair on each line where words are separated by a tab
-test testFile	format is a word pair on each line where words are separated by a tab
-seed seedFile	format is a word pair on each line where words are separated by a tab
-unsupervisedIteration N	Specifies the number of iterations N for the aligner, unsupervised miner, unsupervised part of semi-supervised miner and supervised miner. Default is calculated automatically based on the likelihood of the training data
-semisupervisedIteration M	Specifies the number of iterations M for the semi-supervised part of the semi-supervised system. Default is calculated automatically based on the likelihood of the training data
-transliterationorder K	Specifies the order of the transliteration sub-model. Default is unigram order
-nontransliterationorder L	Specifies the order of the non-transliteration sub-model. Default is unigram order
-t T	Threshold on the posterior probability of non-transliterations. Default is no threshold and it outputs the complete test data with their posterior probability of non-transliteration. Generally, a threshold of 0.5 works fine for most of the language pairs

-s S	Value of the smoothing parameter in the semi-supervised miner. Default value is the number of unigrams in the Viterbi alignments of the seed data
-tirules	Outputs multigrams learned from the training data with their log probabilities in a file
-a	Outputs the test data aligned at character level with their Viterbi probabilities

Table A.1.: Transliteration mining software user manual

B. Gold Standard for English/Hindi

I manually created a gold standard for the English/Hindi language pair. The word pairs in the gold standard are extracted from a word aligned parallel corpus of English/Hindi made available for the shared task on word alignment, organized as part of the ACL 2005 Workshop on Building and Using Parallel Texts (Martin et al., 2005). The English/Hindi parallel corpus is available at www.cse.unt.edu/~rada/wpt05/. The gold standard is available at www.ims.uni-stuttgart.de/~sajjad/resources.html.

The data is released under the Creative Commons license. I request a citation of the following paper if the data is used in a publication.

Sajjad, Hassan; Fraser, Alexander; Schmid, Helmut (2011). An Algorithm for Unsupervised Transliteration Mining with an Application to Word Alignment. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-11).

B.1. Data Format

The English/Hindi gold standard is in a single line format, where each line contains a word pair and its tag (indicating whether a word is a transliteration). The words and tags are separated by a tab like the following:

AUGUST	अगस्त	ti
AUTHORITIES	अथारिटीज़	ti
APPEAL	अपील	ti
OFFICERS	अफिसरज़	ti

Table B.1.: English/Hindi transliteration examples

english hindi tag

There are three tags – transliteration, close transliteration and non-transliteration. A close transliteration is also a non-transliteration but I annotate it separately to better analyse the behavior of mining systems on different kinds of non-transliterations. In this thesis, I have not analyzed the results using different non-transliteration categories. I have merged them under the non-transliteration category. However I hope that they are useful for the analysis of future transliteration mining methods. The gold standard contains 412 transliteration pairs, 72 close transliteration pairs and 12339 non-transliteration pairs.

B.1.1. Transliteration

All transliteration pairs have the tag *ti*. Table B.1 shows a few examples of transliteration pairs.

B.1.2. Close Transliteration

There are a few word pairs that differ by one or two characters from a transliteration pair. I tag them as *tm*. Table B.2 shows a few examples of them.

OFFICERS	अफ़्सर	tm
ARBITRATOR'S	आरबिट्रेटर	tm
AGENT	एजंट्स	tm
CUSTOMER	कस्टमर्ज़	tm

Table B.2.: English/Hindi close transliteration examples

IT	अपेक्षित	ma
SIDE	अप्रिय	ma
LACK	अभाव	ma
GETS	असर	ma

Table B.3.: English/Hindi non-transliteration examples

B.1.3. Non-Transliteration

All non-transliterations other than *tm* are tagged *ma*. Table B.3 shows a few examples of them.

C. Gold Standard for English/Arabic

I manually created a gold standard for English/Arabic. The word pairs in the gold standard are extracted from a freely available English/Arabic parallel corpus of United Nations (UN) (Eisele and Chen, 2010). The parallel corpus is available at http://www.euromatrixplus.eu/multi-un/. I randomly take 200,000 parallel sentences from the UN corpus of the year 2000, word align it and extract the list of word pairs from it. For gold standard, I randomly pick a few word pairs and annotate them. The gold standard is available at www.ims.uni-stuttgart.de/~sajjad/resources.html.

The data is released under the Creative Commons license. I request a citation of the following paper if the data is used in a publication.

Sajjad, Hassan; Fraser, Alexander; Schmid, Helmut (2011). An Algorithm for Unsupervised Transliteration Mining with an Application to Word Alignment. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-11).

C.1. Data Format

The English/Arabic gold standard is in a single line format, where each line contains a word pair and its tag (indicating whether a word is a transliteration). The words and tags are separated by a tab like the following:

BAKU	باكو	ti
BAKER	بيكر	ti
BILLION	بليــون	ti
BURIMA	بوريما	ti

Table C.1.: English/Arabic transliteration examples

english arabic tag

There are four kinds of tags – transliteration, close transliteration, non-transliteration and word pairs where English and Arabic words just differ by an additional affix on the Arabic side from a transliteration (affix pair). Close transliteration and affix pair are a type of non-transliteration. For the analysis in this thesis, I have not analyzed the results using different non-transliteration categories. I have merged them under non-transliteration pairs. However I hope that they are useful for the analysis of future transliteration mining methods. The gold standard contains 288 transliteration pairs, 43 close transliteration pairs, 75 affix pairs and 6521 non-transliteration pairs.

C.1.1. Transliteration

All transliteration pairs have tag *ti*. Table C.1 shows a few examples of transliteration pairs.

C.1.2. Close Transliteration

Word pairs which differ by one or two characters to qualify as transliteration pairs are tagged *d*. Table C.2 shows a few examples of them.

BERNARD	برنار	d
BAGILISHEMA	راغيليشيما	d
BANGLADESH	لبنغلاديش	d
BIOLOGY	بيولوجيا	d

Table C.2.: English/Arabic close transliteration examples

BUTANE	البيوتين	tm
BAHAMA	البهاما	tm
BAKOOL	وباكول	tm
BOLIVIA	وبوليفيا	tm

Table C.3.: English/Arabic affix pairs

C.1.3. Affix Pair

In Arabic, article "al" and conjunction "wao" are attached to the word. There are cases in the list of word pairs where if an Arabic word is considered without "al" and "wao", it is a perfect transliteration of its corresponding English word. I tag these word pairs as *tm*. Note that the tag *tm* is really a special type of near transliteration which I otherwise mark with *d*. Table C.3 shows a few examples of affix pairs.

C.1.4. Non-Transliteration

The non-transliterations other than close transliterations and affix pairs are tagged *ma*. A few examples are shown in Table C.4.

BURDENED	المعاناة	ma
BACK-UP	الاحتياطية	ma
BE	وإ قفال	ma
BEAUTIFUL	جميلة	ma

Table C.4.: English/Arabic non-transliteration examples