Transliteration Generation and Mining with Limited Training Resources

Sittichai Jiampojamarn, Kenneth Dwyer, Shane Bergsma, Aditya Bhargava, Qing Dou, Mi-Young Kim, Grzegorz Kondrak
Department of Computing Science
University of Alberta
Edmonton, AB, T6G 2E8, Canada
{sj,dwyer,bergsma,abhargava,qdou,miyoung2,kondrak}@cs.ualberta.ca

Abstract

We present DIRECTL+: an online discriminative sequence prediction model based on many-to-many alignments, which is further augmented by the incorporation of joint n-gram features. Experimental results show improvement over the results achieved by DIRECTL in 2009. We also explore a number of diverse resource-free and language-independent approaches to transliteration mining, which range from simple to sophisticated.

1 Introduction

Many out-of-vocabulary words in statistical machine translation and cross-language information retrieval are named entities. If the languages in question use different writing scripts, such names must be transliterated. Transliteration can be defined as the conversion of a word from one writing script to another, which is usually based on the phonetics of the original word.

DIRECTL+ is our current approach to name transliteration which is an extension of the DIRECTL system (Jiampojamarn et al., 2009). We augmented the feature set with joint n-gram features which allow the discriminative model to utilize long dependencies of joint information of source and target substrings (Jiampojamarn et al., 2010). Experimental results suggest an improvement over the results achieved by DIRECTL in 2009.

Transliteration mining aims at automatically obtaining bilingual lists of names written in different scripts. We explore a number of different approaches to transliteration mining in the context of the NEWS 2010 Shared Task.[1] The sole resource that is provided for each language pair is a "seed" dataset that contains 1K transliteration word pairs. The objective is then to mine transliteration pairs from a collection of Wikipedia titles/topics that are given in both languages.

We explore a number of diverse resource-free and language-independent approaches to transliteration mining. One approach is to bootstrap the seed data by generating pseudo-negative examples, which are combined with the positives to form a dataset that can be used to train a classifier. We are particularly interested in achieving good performance without utilizing language-specific resources, so that the same approach can be applied with minimal or no modifications to an array of diverse language pairs.

This paper is divided in two main parts that correspond to the two tasks of transliteration generation and transliteration mining.

2 Transliteration generation

The structure of this section is as follows. In Section 2.1, we describe the pre-processing steps that were applied to all datasets. Section 2.2 reviews two methods for aligning the source and target symbols in the training data. We provide details on the DIRECTL+ systems in Section 2.3. In Section 2.4, we discuss extensions of DIRECTL+ that incorporate language-specific information. Section 2.5 summarizes our results.

2.1 Pre-processing

For all generation tasks, we pre-process the provided data as follows. First, we convert all characters in the source word to lower case. Then, we remove non-alphabetic characters unless they appear in both the source and target words. We normalize whitespace that surrounds a comma, so that there are no spaces before the comma and exactly one space following the comma. Finally, we separate multi-word titles into single words, using whitespace as the separator. We assume a mono-

[1] http://translit.i2r.a-star.edu.sg/news2010

Proceedings of the 2010 Named Entities Workshop, ACL 2010, pages 39–47,
Uppsala, Sweden, 16 July 2010. ©2010 Association for Computational Linguistics

tonic matching and ignore the titles that have a different number of words on both sides.

We observed that in the ArAe task there are cases where an extra space is added to the target when transliterating from Arabic names to their English equivalents; e.g., "Al Riyad", "El Sayed", etc. In order to prevent the pre-processing from removing too many title pairs, we allow non-equal matching if the source title is a single word.

For the English-Chinese (EnCh) task, we convert the English letter "x" to "ks" to facilitate better matching with its Chinese targets.

During testing, we pre-process test data in the same manner, except that we do not remove non-alphabetic characters. After the pre-processing steps, our system proposes 10-best lists for single word titles in the test data. For multi-word titles, we construct 10-best lists by ranking the combination scores of single words that make up the test titles.

2.2 Alignment

In the transliteration tasks, training data consist of pairs of names written in source and target scripts without explicit character-level alignment. In our experiments, we applied two different algorithms to automatically generate alignments in the training data. The generated alignments provide hypotheses of substring mappings in the training data. Given aligned training data, a transliteration model is trained to generate names in the target language given names in the source language.

The M2M-aligner (Jiampojamarn et al., 2007) is based on the expectation maximization (EM) algorithm. It allows us to create alignments between substrings of various lengths. We optimized the maximum substring sizes for the source and target based on the performance of the end task on the development sets. We allowed empty strings (*nulls*) only on the target side. We used the M2M-aligner for all alignment tasks, except for English-Pinyin alignment. The source code of the M2M-aligner is publicly available.[2]

An alternative alignment algorithm is based on the phonetic similarity of graphemes. The key idea of this approach is to represent each grapheme by a phoneme or a sequence of phonemes that is likely to be represented by the grapheme. The sequences of phonemes on the source side and the target side can then be aligned on the basis of phonetic similarity between phonemes. The main advantage of the phonetic alignment is that it requires no training data. We use the ALINE phonetic aligner (Kondrak, 2000), which aligns two strings of phonemes. The example in Figure 1 shows the alignment of the word *Barclay* to its Katakana transliteration *ba-ku-ri*. The one-to-one alignment can then be converted to a many-to-many alignment by grouping the Japanese phonemes that correspond to individual Katakana symbols.

```
b a r c - l a y
| | | | | | | |
b a - k u r - i
```

Figure 1: An alignment example.

2.3 DIRECTL+

We refer to our present approach to transliteration as DIRECTL+. It is an extension of our DIRECTL system (Jiampojamarn et al., 2009). It includes additional "joint n-gram" features that allow the discriminative model to correlate longer source and target substrings. The additional features allow our discriminative model to train on information that is present in generative joint n-gram models, and additionally train on rich source-side context, transition, and linear-chain features that have been demonstrated to be important in the transliteration task (Jiampojamarn et al., 2010).

Our model is based on an online discriminative framework. At each training iteration, the model generates an m-best list for each given source name based on the current feature weights. The feature weights are updated according to the gold-standard answers and the generated m-best answer lists using the Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003). This training process iterates over the training examples until the model converges. For m-best and n-gram parameters, we set $m = 10$ and $n = 6$ for all language pairs. These parameters as well as others were optimized on the development sets.

We trained our models directly on the data that were provided by the organizers, with three exceptions. In order to improve performance, we gave special treatment to English-Korean (EnKo), English-Chinese (EnCh), and English-Hindi (EnHi). These special cases are described in the next section.

[2] `http://code.google.com/p/m2m-aligner/`

2.4 Beyond DIRECTL+

2.4.1 Korean Jaso

A Korean syllable can be decomposed into two or three components called *Jaso*: an initial consonant, a middle vowel, and optionally a final consonant. The Korean generation for EnKo involves the following three steps: (1) English-to-Jaso generation, (2) correction of illegal Jaso sequences, and (3) Jaso-to-Korean conversion.

In order to correct illegal Jaso sequences that cannot be combined into Korean syllables in step 2, we consider both vowel and consonant rules. A Korean vowel can be either a simple vowel or a complex vowel that combines two simple vowels. We can use this information in order to replace double vowels with one complex vowel. We also use the silent consonant *o* (i-eung) when we need to insert a consonant between double vowels. A Korean vowel - (eu) is most commonly inserted between two English consonants in transliteration. In order to resolve three consecutive consonants, it can be placed into the most probable position according to the probability distribution of the training data.

2.4.2 Japanese Katakana

In the Japanese Katakana generation task, we replace each Katakana symbol with one or two letters using standard romanization tables. This has the effect of expressing the target side in Latin letters, which facilitates the alignment. DIRECTL+ is trained on the converted data to generate the target from the source. A post-processing program then attempts to convert the generated letters back into Katakana symbols. Sequences of letters that cannot be converted into Katakana are removed from the output m-best lists and replaced by lower scoring sequences that pass the back-conversion filter. Otherwise, there is usually a single valid mapping because most Katakana symbols are represented by single vowels or a consonant-vowel pair. The only apparent ambiguity involves the letter n, which can either stand by itself or cluster with the following vowel letter. We resolve the ambiguity by always assuming the latter case unless the letter n occurs at the end of the word.

2.4.3 Chinese Pinyin

Following (Jiampojamarn et al., 2009), we experimented with converting the original Chinese characters to Pinyin as an intermediate representation. Pinyin is the most commonly known romanization system for Standard Mandarin and many free tools are available for converting Chinese characters to Pinyin. Its alphabet contains the same 26 letters as English. Each Chinese character can be transcribed phonetically into Pinyin. A small percentage of Chinese characters have multiple pronunciations, and are thus represented by different Pinyin sequences. For those characters, we manually selected the pronunciations that are normally used for names. This pre-processing step significantly reduces the size of the target symbols: from 370 distinct Chinese characters to 26 Pinyin symbols. This allows our system to produce better alignments.

We developed three models: (1) trained on the original Chinese characters, (2) trained on Pinyin, and (3) the model that incorporates the phonetic alignment described in Section 2.2. The combination of the predictions of the different systems was performed using the following simple algorithm (Jiampojamarn et al., 2009). First, we rank the individual systems according to their top-1 accuracy on the development set. To obtain the top-1 prediction for each input word, we use simple voting, with ties broken according to the ranking of the systems. We generalize this approach to handle n-best lists by first ordering the candidate transliterations according to the rank assigned by each individual system, and then similarly breaking ties by voting and using the ranking of the systems.

2.4.4 Language identification for Hindi

Bhargava and Kondrak (2010) apply support vector machines (SVMs) to the task of identifying the language of names. The intuition here is that language information can inform transliteration. Bhargava and Kondrak (2010) test this hypothesis on the NEWS 2009 English-Hindi transliteration data by training language identification on data manually tagged as being of either Indian or non-Indian origin. It was found that splitting the data disjointly into two sets and training separate transliteration models yields no performance increase due to the decreased size of the data for the models.

We adopt this approach for the NEWS 2010 task, but here we do not use disjoint splits. Instead, we use the SVMs to generate probabilities, and then we apply a threshold to these probabilities to generate two datasets. For example, if we set the threshold to be 0.05, then we determine the

probabilities of a given name being of Indian origin (p_{hi}) and of being of non-Indian origin (p_{en}). If $p_{hi} < 0.05$ then the name is excluded from the Indian set, and if $p_{en} < 0.05$ then the name is excluded from the non-Indian set. Using the two obtained non-disjoint sets, we then train a transliteration model for each set using DIRECTL+.

Since the two sets are not disjoint, we must decide how to combine the two results. Given that a name occurs in both sets, and both models provide a ranked list of possible targets for that name, we obtain a combined ranking using a linear combination over the mean reciprocal ranks (MRRs) of the two lists. The weights used are p_{hi} and p_{en} so that the more likely a name is considered to be of Indian origin, the more strongly the result from the Indian set is considered relative to the result from the non-Indian set.

2.5 Evaluation

In the context of the NEWS 2010 Machine Transliteration Shared Task we tested our system on all twelve datasets: from English to Chinese (EnCh), Thai (EnTh), Hindi (EnHi), Tamil (EnTa), Bangla (EnBa), Kannada (EnKa), Korean Hangul (EnKo), Japanese Katakana (EnJa), Japanese Kanji (JnJk); and, in the opposite direction, to English from Arabic (ArAe), Chinese (ChEn), and Thai (ThEn). For all datasets, we trained transliteration models on the provided training and development sets without additional resources.

Table 1 shows our best results obtained on the datasets in terms of top-1 accuracy and mean F-score. We also include the rank in standard runs ordered by top-1 word accuracy. The EnCh result presented in the table refers to the output of the three-system combination, using the combination algorithm described in Section 2.4.3. The respective results for the three component EnCh systems were: 0.357, 0.360, and 0.363. The EnJa result in the table refers the system described in Section 2.4.2 that applied specific treatment to Japanese Katakana. Based on our development results, this specific treatment improves as much as 2% top-1 accuracy over the language-independent model. The EnHi system that incorporates language identification obtained exactly the same top-1 accuracy as the language-independent model. However, the EnKo system with Jaso correction produced the top-1 accuracy of 0.554, which is a significant improvement over 0.387 achieved by the language-independent model.

Task	top-1	F-score	Rank
EnCh	0.363	0.707	2
ChEn	0.137	0.740	1
EnTh	0.378	0.866	2
ThEn	0.352	0.861	2
EnHi	0.456	0.884	1
EnTa	0.390	0.891	2
EnKa	0.341	0.867	2
EnJa	0.398	0.791	1
EnKo	0.554	0.770	1
JnJk	0.126	0.426	1
ArAe	0.464	0.924	1
EnBa	0.395	0.877	2

Table 1: Transliteration generation results

3 Transliteration mining

This section is structured as follows. In Section 3.1, we describe the method of extracting transliteration candidates that serves as the input to the subsequently presented mining approaches. Two techniques for generating negative examples are discussed in Section 3.2. Our language-independent approaches to transliteration mining are described in Section 3.3, and a technique for mining English-Chinese pairs is proposed in Section 3.4. In Section 3.5, we address the issue of overlapping predictions. Finally, Section 3.6 and Section 3.7 summarize our results.

3.1 Extracting transliteration candidates

We cast the transliteration mining task as a binary classification problem. That is, given a word in the source language and a word in the target language, a classifier predicts whether or not the pair constitutes a valid transliteration. As a preprocessing step, we extract candidate transliterations from the pairs of Wikipedia titles. Word segmentation is performed based on sequences of one or more spaces and/or punctuation symbols, which include hyphens, underscores, brackets, and several other non-alphanumeric characters. Apostrophes and single quotes are not used for segmentation (and therefore remain in a given word); however, all single quote-like characters are converted into a generic apostrophe. Once an English title and its target language counterpart have been

segmented into words, we form the candidate set for this title as the cross product of the two sets of words after discarding any words that contain fewer than two characters.

After the candidates have been extracted, individual words are flagged for certain attributes that may be used by our supervised learner as additional features. Alternatively, the flags may serve as criteria for filtering the list of candidate pairs prior to classification. We identify words that are capitalized, consist of all lowercase (or all capital) letters, and/or contain one or more digits. We also attempt to encode each word in the target language as an ASCII string, and flag that word if the operation succeeds. This can be used to filter out words that are written in English on both the source and target side, which are not transliterations by definition.

3.2 Generating negative training examples

The main issue with applying a supervised learning approach to the NEWS 2010 Shared Task is that annotated task-specific data is not available to train the system. However, the seed pairs do provide example transliterations, and these can be used as positive training examples. The remaining issue is how to select the negative examples.

We adopt two approaches for selecting negatives. First, we generate all possible source-target pairs in the seed data, and take as negatives those pairs which are not transliterations but have a longest common subsequence ratio (LCSR) above 0.58; this mirrors the approach used by Bergsma and Kondrak (2007). The method assumes that the source and target words are written in the same script (e.g., the foreign word has been romanized).

A second possibility is to generate all seed pairings as above, but then randomly select negative examples, thus mirroring the approach in Klementiev and Roth (2006). In this case, the source and target scripts do not need to be the same. Compared with the LCSR technique, random sampling in this manner has the potential to produce negative examples that are very "easy" (i.e., clearly not transliterations), and which may be of limited utility when training a classifier. On the other hand, at test time, the set of candidates extracted from the Wikipedia data will include pairs that have very low LCSR scores; hence, it can be argued that dissimilar pairs should also appear as negative examples in the training set.

3.3 Language-independent approaches

In this section, we describe methods for transliteration mining that can, in principle, be applied to a wide variety of language pairs without additional modification. For the purposes of the Shared Task, however, we convert all source (English) words to ASCII by removing diacritics and making appropriate substitutions for foreign letters. This is done to mitigate sparsity in the relatively small seed sets when training our classifiers.

3.3.1 Alignment-derived romanization

We developed a simple method of performing romanization of foreign scripts. Initially, the seed set of transliterations is aligned using the one-to-one option of the M2M-aligner approach (Jiampojamarn et al., 2007). We allow nulls on both the source and target sides. The resulting alignment model contains pairs of Latin letters and foreign script symbols (graphemes) sorted by their conditional probability. Then, for each grapheme, we select a letter (or a null symbol) that has the highest conditional probability. The process produces an approximate romanization table that can be obtained without any knowledge of the target script. This method of romanization was used by all methods described in the remainder of Section 3.3.

3.3.2 Normalized edit distance

Normalized edit distance (NED) is a measure of the similarity of two strings. We define a uniform edit cost for each of the three operations: substitution, insertion, and deletion. NED is computed by dividing the minimum edit distance by the length of the longer string, and subtracting the resulting fraction from 1. Thus, the extreme values of NED are 1 for identical strings, and 0 for strings that have no characters in common.

Our baseline method, NED+ is simply the NED measure augmented with filtering of the candidate pairs described in Section 3.1. In order to address the issue of morphological variants, we also filter out the pairs in which the English word ends in a consonant and the foreign word ends with a vowel. With no development set provided, we set the similarity thresholds for individual languages on the basis of the average word length in the seed sets. The values were 0.38, 0.48, 0.52, and 0.58 for Hindi, Arabic, Tamil, and Russian, respectively, with the last number taken from Bergsma and Kondrak (2007).

3.3.3 Alignment-based string similarity

NED selects transliteration candidates when the romanized foreign strings have high character overlap with their English counterparts. The measure is independent of the language pair. This is suboptimal for several reasons. First of all, phonetically unrelated words can share many incidental character matches. For example, the French word 'recettes' and the English word 'proceeds' share the letters *r,c,e,e,s* as a common subsequence, but the words are phonetically unrelated. Secondly, many reliable, recurrent, language-specific substring matches are prevalent in true transliterations. These pairings may or may not involve matching characters. NED can not learn or adapt to these language-specific patterns.

In light of these drawbacks, researchers have proposed string similarity measures that can learn from provided example pairs and adapt the similarity function to a specific task (Ristad and Yianilos, 1998; Bilenko and Mooney, 2003; McCallum et al., 2005; Klementiev and Roth, 2006).

One particularly successful approach is by Bergsma and Kondrak (2007), who use discriminative learning with an improved feature representation. The features are substring pairs that are consistent with a character-level alignment of the two strings. This approach strongly improved performance on cognate identification, while variations of it have also proven successful in transliteration discovery (Goldwasser and Roth, 2008). We therefore adopted this approach for the transliteration mining task.

We produce negative training examples using the LCSR threshold approach described in Section 3.2. For features, we extract from the aligned word pairs all substring pairs up to a maximum length of three. We also append characters marking the beginning and end of words, as described in Bergsma and Kondrak (2007). For our classifier, we use a Support Vector Machine (SVM) training with the very efficient LIBLINEAR package (Fan et al., 2008). We optimize the SVM's regularization parameter using 10-fold cross validation on the generated training data. At test time, we apply our classifier to all the transliteration candidates extracted from the Wikipedia titles, generating transliteration pairs whenever there is a positive classification.

3.3.4 String kernel classifier

The alignment-based classifier described in the preceding section is limited to using substring features that are up to (roughly) three or four letters in length, due to the combinatorial explosion in the number of unique features as the substring length increases. It is natural to ask whether longer substrings can be utilized to learn a more accurate predictor.

This question inspired the development of a second SVM-based learner that uses a string kernel, and therefore does not have to explicitly represent feature vectors. Our kernel is a standard n-gram (or spectrum) kernel that implicitly embeds a string in a feature space that has one co-ordinate for each unique n-gram (see, e.g., (Shawe-Taylor and Cristianini, 2004)). Let us denote the alphabet over input strings as A. Given two input strings x and x', this kernel function computes:

$$k(x, x') = \sum_{s \in A^n} \#(s, x)\#(s, x')$$

where s is an n-gram and $\#(a, b)$ counts the number of times a appears as a substring of b.

An extension of the n-gram kernel that we employ here is to consider all n-grams of length $1 \leq n \leq k$, and weight each n-gram as a function of its length. In particular, we specify a value λ and weight each n-gram by a factor of λ^n. We implemented this kernel in the LIBSVM software package (Chang and Lin, 2001). Optimal values for k, λ, and the SVM's regularization parameter were estimated for each dataset using 5-fold cross-validation. The values of (k, λ) that we ultimately used were: EnAr $(3, 0.8)$, EnHi $(8, 0.8)$, EnRu $(5, 1.2)$, and EnTa $(5, 1.0)$.

Our input string representation for a candidate pair is formed by first aligning the source and target words using M2M-aligner (Jiampojamarn et al., 2007). Specifically, an alignment model is trained on the seed examples, which are subsequently aligned and used as positive training examples. We then generate 20K negative examples by random sampling (cf. Section 3.2) and apply the alignment model to this set. Not all of these 20K word pairs will necessarily be aligned; we randomly select 10K of the successfully aligned pairs to use as negative examples in the training set.

Each aligned pair is converted into an "alignment string" by placing the letters that appear in

Word pair	zubtsov	зубцов
Aligned pair	z|u|b|t|s|o|v|	з|у|б|ц|_|о|в|
Align't string	zз|uу|bб|tц|s_|oо|vв	

Table 2: An example showing how an alignment string (the input representation for the string kernel) is created from a word pair.

the same position in the source and target next to one another, while retaining the separator characters (see Table 2). We also appended beginning and end of word markers. Note that no romanization of the target words is necessary for this procedure.

At test time, we apply the alignment model to the candidate word pairs that have been extracted from the *train* data, and retain *all* the successfully aligned pairs. Here, M2M-aligner also acts as a filter, since we cannot form alignment strings from unaligned pairs — these yield negative predictions by default. We also filter out pairs that met any of the following conditions: 1) the English word consists of all all capital or lowercase letters, 2) the target word can be converted to ASCII (cf. Section 3.1), or 3) either word contains a digit.

3.3.5 Generation-based approach

In the mining tasks, we are interested in whether a candidate pair (x, y) is a transliteration pair. One approach is to determine if the generated transliterations of a source word $\hat{y} = \alpha(x)$ and a target word $\hat{x} = \beta(y)$ are similar to the given candidate pair. We applied DIRECTL+ to the mining tasks by training transliteration generation models on the provided seed data in forward and backward transliteration directions, creating $\alpha(x)$ and $\beta(y)$ models. We now define a transliteration score function in Eq. 1. $N(\hat{x}, x)$ is the normalized edit distance between string $\hat{x}$ and x, and w_1 and w_2 are combination weights to favor forward and backward transliteration models.

$$S(x, y) = \frac{w_1 \cdot N(\hat{y}, y) + w_2 \cdot N(\hat{x}, x)}{w_1 + w_2} \quad (1)$$

A candidate pair is considered a transliteration pair if its $S(x, y) > \tau$. Ideally, we would like to optimize these parameters, τ, w_1, w_2 based on a development set for each language pair. Unfortunately, no development sets were provided for the Shared Task. Therefore, following Bergsma and Kondrak (2007), we adopt the threshold of $\tau = 0.58$. We experimented with three sets of values for w_1 and w_2: $(1, 0)$, $(0.5, 0.5)$, and $(0, 1)$. Our final predictions were made using $w_0 = 0$ and $w_1 = 1$, which appeared to produce the best results. Thus, only the backward transliteration model was ultimately employed.

3.4 English-Chinese string matching

Due to the fact that names transliterated into Chinese consist of multiple Chinese characters and that the Chinese text provided in this shared task is not segmented, we have to adopt a different approach to the English-Chinese mining task (Unlike many other languages, there are no clear boundaries between Chinese words). We first train a generation model using the seed data and then apply a greedy string matching algorithm to extract transliteration pairs.

The generation model is built using the discriminative training framework described in (Jiampojamarn et al., 2008). Two models are learned: one is trained using English and Chinese characters, while the other is trained on English and Pinyin (a standard phonetic representation of Chinese characters). In order to mine transliteration pairs from Wikipedia titles, we first use the generation model to produce transliterations for each English token on the source side as both Chinese characters and Pinyin. The generated Chinese characters are ultimately converted to Pinyin during string matching. We also convert all the Chinese characters on the target side to their Pinyin representations when performing string matching.

The transliteration pairs are then mined by combining two different strategies. First of all, we observe that most of the titles that contain a separation symbol " · " on the target side are transliterations. In this case, the number of tokens on both sides is often equal. Therefore, the mining task can be formulated as a matching problem. We use a competitive linking approach (Melamed, 2000) to find the best match. First, we select links between all possible pairs if similarity of strings on both sides is above a threshold ($0.6 * length(Pinyin)$). We then greedily extract the pairs with highest similarity until the number of unextracted segments on either side becomes zero.

The problem becomes harder when there is no indication of word segmentation for Chinese. Instead of trying to segment the Chinese characters first, we use an incremental string matching strat-

egy. For each token on the source side, the algorithm calculates its similarity with all possible n-grams ($2 \leq n \leq L$) on the target side, where L is the length of the Chinese title (i.e., the number of characters). If the similarity score of n-gram with the highest similarity surpasses a threshold ($0.5 \times length(Pinyin)$), the n-gram sequence is proposed as a possible transliteration for the current source token.

3.5 Resolving overlapping predictions

Given a set of candidate word pairs that have been extracted from a given Wikipedia title according to the procedure described in Section 3.1, our classifiers predict a class label for each pair independently of the others. Pairs that receive negative predictions are discarded immediately and are never reported as mined pairs. However, it is sometimes necessary to arbitrate between positive predictions, since it is possible for a classifier to mark as transliterations two or more pairs that involve the same English word or the same target word in the title. Clearly, mining multiple overlapping pairs will lower the system's precision, since there is (presumably) at most one correct transliteration in the target language version of the title for each English word.[3]

Our solution is to apply a greedy algorithm that sorts the word pair predictions for a given title in descending order according to the scores that were assigned by the classifier. We make one pass through the sorted list and report a pair of words as a mined pair unless the English word or the target language word has already been reported (for this particular title).[4]

3.6 Results

In the context of the NEWS 2010 Shared Task on Transliteration Generation we tested our system on all five data sets: from English to Russian (EnRu), Hindi (EnHi), Tamil (EnTa), Arabic (EnAr), and Chinese (EnCh). The EnCh set differs from the remaining sets in the lack of transparent word segmentation on the Chinese side. There were no development sets provided for any of the language pairs.

[3]On the other hand, mining all such pairs *might* improve recall.

[4]A bug was later discovered in our implementation of this algorithm, which had failed to add the words in a title's first mined pair to the "already reported" list. This sometimes caused up to two additional mined pairs per title to be reported in the prediction files that were submitted.

Task	System	F	P	R
EnRu	NED+	**.875**	.880	.869
	BK-2007	.778	.684	.902
	StringKernel*	.811	.746	.889
	DIRECTL+	.786	.778	.795
EnHi	NED+	.907	.875	.941
	BK-2007	.882	.883	.880
	StringKernel	**.924**	.954	.895
	DIRECTL+	.904	.945	.866
EnTa	NED+	.791	.916	.696
	BK-2007	.829	.808	.852
	StringKernel	**.914**	.923	.906
	DIRECTL+	.801	.919	.710
EnAr	NED+	.800	.818	.783
	BK-2007	.816	.834	.798
	StringKernel*	**.827**	.917	.753
	DIRECTL+	.742	.861	.652
EnCh	GreedyMatch	**.530**	.698	.427
	DIRECTL+	.009	.045	.005

Table 3: Transliteration mining results. An asterisk (*) indicates an unofficial result.

Table 3 shows the results obtained by our various systems on the final test sets, measured in terms of F-score (F), precision (P), and recall (R). The systems referred to as NED+, BK-2007, StringKernel, DIRECTL+, and GreedyMatch are described in Section 3.3.2, Section 3.3.3, Section 3.3.4, Section 3.3.5, and Section 3.4 respectively. The runs marked with an asterisk (*) were produced after the Shared Task deadline, and therefore are not included in the official results.

3.7 Discussion

No fixed ranking of the four approaches emerges across the four alphabetic language pairs (all except EnCh). However, StringKernel appears to be the most robust, achieving the highest F-score on three language pairs. This suggests that longer substring features are indeed useful for classifying candidate transliteration pairs. The simple NED+ method is a clear winner on EnRu, and obtains decent scores on the remaining alphabetic language pairs. The generation-based DIRECTL+ approach ranks no higher than third on any language pair, and it fails spectacularly on EnCh because of the word segmentation ambiguity.

Finally, we observe that there are a number of cases where the results for our discriminatively trained classifiers, BK-2007 and StringKernel, are

not significantly better than those of the simple NED+ approach. We conjecture that automatically generating training examples is suboptimal for this task. A more effective strategy may be to filter all possible word pairs in the seed data to only those with NED above a fixed threshold. We would then apply the same threshold to the Wikipedia candidates, only passing to the classifier those pairs that surpass the threshold. This would enable a better match between the training and test operation of the system.

4 Conclusion

The results obtained in the context of the NEWS 2010 Machine Transliteration Shared Task confirm the effectiveness of our discriminative approaches to transliteration generation and mining.

Acknowledgments

This research was supported by the Alberta Ingenuity Fund, Informatics Circle of Research Excellence (iCORE), and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Shane Bergsma and Grzegorz Kondrak. 2007. Alignment-based discriminative string similarity. In *Proc. ACL*.

Aditya Bhargava and Grzegorz Kondrak. 2010. Language identification of names with SVMs. In *Proc. NAACL-HLT*.

Mikhail Bilenko and Raymond J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proc. KDD*.

Chih-Chung Chang and Chih-Jen Lin. 2001. LIBSVM: a library for support vector machines. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874.

Dan Goldwasser and Dan Roth. 2008. Transliteration as constrained optimization. In *Proc. EMNLP*.

Sittichai Jiampojamarn, Grzegorz Kondrak, and Tarek Sherif. 2007. Applying many-to-many alignments and Hidden Markov Models to letter-to-phoneme conversion. In *Proc. HLT-NAACL*.

Sittichai Jiampojamarn, Colin Cherry, and Grzegorz Kondrak. 2008. Joint processing and discriminative training for letter-to-phoneme conversion. In *Proc. ACL*.

Sittichai Jiampojamarn, Aditya Bhargava, Qing Dou, Kenneth Dwyer, and Grzegorz Kondrak. 2009. DirecTL: a language-independent approach to transliteration. In *NEWS '09: Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, pages 28–31.

Sittichai Jiampojamarn, Colin Cherry, and Grzegorz Kondrak. 2010. Integrating joint n-gram features into a discriminative training framework. In *Proc. NAACL-HLT*.

Alexandre Klementiev and Dan Roth. 2006. Named entity transliteration and discovery from multilingual comparable corpora. In *Proc. HLT-NAACL*.

Grzegorz Kondrak. 2000. A new algorithm for the alignment of phonetic sequences. In *Proc. NAACL*, pages 288–295.

Andrew McCallum, Kedar Bellare, and Fernando Pereira. 2005. A conditional random field for discriminatively-trained finite-state string edit distance. In *Proc. UAI*.

I. Dan Melamed. 2000. Models of translational equivalence among words. *Computational Linguistics*, 26(2):221–249.

Eric Sven Ristad and Peter N. Yianilos. 1998. Learning string-edit distance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20(5).

John Shawe-Taylor and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.