Named Entity Transcription with Pair *n*-Gram Models

Martin Jansche
Google Inc.
mjansche@google.com

Richard Sproat
Google Inc. and OHSU
rws@google.com

Abstract

We submitted results for each of the eight shared tasks. Except for Japanese name kanji restoration, which uses a noisy channel model, our Standard Run submissions were produced by generative long-range pair *n*-gram models, which we mostly augmented with publicly available data (either from LDC datasets or mined from Wikipedia) for the Non-Standard Runs.

1 Introduction

This paper describes the work that we did at Google, Inc. for the NEWS 2009 Machine Transliteration Shared Task (Li et al., 2009b; Li et al., 2009a). Except for the Japanese kanji task (which we describe below), all models were pair *n*-gram language models. Briefly, we took the training data, and ran an iterative alignment algorithm using a single-state weighted finite-state transducer (WFST). We then trained a language model on the input-output pairs of the alignment, which was then converted into a WFST encoding a joint model. For the Non-Standard runs, we use additional data from Wikipedia or from the LDC, except where noted below. In the few instances where we used data not available from Wikipedia or LDC, we will be happy to share them with other participants of this competition.

2 Korean

For Korean, we created a mapping between each Hangul glyph and its phonetic transcription in WorldBet (Hieronymus, 1993) based on the tables from Unitran (Yoon et al., 2007). Vowel-initial syllables were augmented with a "0" at the beginning of the syllable, to avoid spurious resyllabifications: *Abbott* should be 애버트, never 앱얻으. We also filtered the set of possible Hangul syllable combinations, since certain syllables are never used in transliterations, e.g. any with two consonants in the coda. The mapping between Hangul syllables and phonetic transcription was handled with a simple FST.

The main transliteration model for the Standard Run was a 10-gram pair language model trained on an alignment of English letters to Korean phonemes. All transliteration pairs observed in the training/ development data were cached, and made available if those names should recur in the test data. We also submitted a Non-Standard Run with English/ Korean pairs mined from Wikipedia. These were derived from the titles of corresponding interlinked English and Korean articles. Obviously not all such pairs are transliterations, so we filtered the raw list by predicting, for each English word, and using the trained transliteration model, what the ten most likely transliterations were in Korean; and then accepting any pair in Wikipedia where the string in Korean also occurred in the set of predicted transliterations. This resulted in 11,169 transliteration pairs. In addition a dictionary of 9,047 English and Korean transliteration pairs that we had obtained from another source was added. These pairs were added to the cache, and were also used to retrain the transliteration model, along with the provided data.

3 Indian Languages

For the Indian languages Hindi, Tamil and Kannada, the same basic approach as for Korean was used. We created a reversible map between Devanagari, Tamil or Kannada symbols and their phonemic values, using a modified version of Unitran. However, since Brahmi-derived scripts distinguish between diacritic and full vowel forms, in order to map back from phonemic transcription into the script form, it is necessary to know whether a vowel comes after a consonant or not, in order to select the correct form. These and other constraints were implemented with a simple hand-constructed WFST for each script.

The main transliteration model for the Standard Run was a 6-gram pair language model trained on an alignment of English letters to Hindi, Kannada

Proceedings of the 2009 Named Entities Workshop, ACL-IJCNLP 2009, pages 32–35,
Suntec, Singapore, 7 August 2009. ©2009 ACL and AFNLP

or Tamil phonemes in the training and development sets. At test time, this WFST was composed with the phoneme to letter WFST just described to produce a WFST that maps directly between English letters and Indian script forms. As with Korean, all observed transliteration pairs from the training/development data were cached, and made available if those names should recur in the test data. For each Indian language we also submitted a Non-Standard Run which included English/Devanagari, English/Tamil and English/Kannada pairs mined from Wikipedia, and filtered as described above for Korean. This resulted in 11,674 pairs for English/Hindi, 10,957 pairs for English/Tamil and 2,436 pairs for English/Kannada. These pairs were then added to the cache, and were also used to retrain the transliteration model, along with the provided data.

4 Russian

For Russian, we computed a direct letter/letter correspondences between the Latin representation of English and the Cyrillic representation of Russian words. This seemed to be a reasonable choice since Russian orthography is fairly phonemic, at least at an abstract level, and it was doubtful that any gain would be had from trying to model the pronunciation better. We note that many of the examples were, in fact, not English to begin with, but a variety of languages, including Polish and others, that happen to be written in the Latin script.

We used a 6-gram pair language model for the Standard Run. For the Non-Standard Runs we included: (for NSR1) a list of 3,687 English/Russian pairs mined from the Web; and (for NSR2), those, plus a set of 1,826 mined from Wikipedia and filtered as described above. In each case, the found pairs were put in the cache, and were used to retrain the language model.

5 Chinese

For Chinese, we built a direct stochastic model between strings of Latin characters representing the English names and strings of hanzi representing their Chinese transcription. It is well known (Zhang et al., 2004) that the direct approach produces significantly better transcription quality than indirect approaches based on intermediate pinyin or phoneme representations. This observation is consistent with our own experience during system development.

In our version of the direct approach, we first aligned the English letter strings with their corresponding Chinese hanzi strings using the same memoryless monotonic alignment model as before. We then built standard *n*-gram models over the alignments, which were then turned, for use at runtime, into weighted FSTs computing a mapping from English to Chinese.

The transcription model we chose for the Standard Run is a 6-gram language model over alignments, built with Kneser-Ney smoothing and a minimal amount of Seymore-Rosenfeld shrinking.

We submitted two Non-Standard Runs with additional names taken from the LDC Chinese/English Name Entity Lists v 1.0 (LDC2005T34). The only list from this collection we used was Propernames People EC, which contains 572,213 "English" names (in fact, names from many languages, all represented in the Latin alphabet) with one or more Chinese transcriptions for each name. Data of similar quality can be easily extracted from the Web as well. For the sake of reproducible results, we deliberately chose to work with a standard corpus. The LDC name lists have all of the problems that are usually associated with data extracted from the Web, including improbable entries, genuine mistakes, character substitutions, a variety of unspecified source languages, etc.

We removed names with symbols other than letters 'a' through 'z' from the list and divided it into a held-out portion, consisting of names that occur in the development or test data of the Shared Task, and a training portion, consisting of everything else, for a total of 622,187 unique English/Chinese name pairs. We then used the model from the Standard Run to predict multiple pronunciations for each of the names in the training portion of the LDC list and retained up to 5 pronunciations for each English name where the prediction from the Standard model agreed with a pronunciation found in the LDC list.

For our first Non-Standard Run, we trained a 7-gram language model based on the Shared Task training data (31,961 name pairs) plus an additional 95,576 name pairs from the intersection of the LDC list and the Standard model predictions. Since the selection of additional training data was, by design, very conservative, we got a small improvement over the Standard Run.

The reason for this cautious approach was that the additional LDC data did not match the provided training and development data very well, partly due to noise, partly due to different transcription conventions. For example, the Pinyin syllable *bó* is predominantly written as 博 in the LDC data, but 博 does not occur at all in the Shared Task training data:

Character	Occurrences	
	Train	LDC
博	0	13,110
伯	1,547	3,709

We normalized the LDC data (towards the transcription conventions implicit in the Shared Task data) by replacing hanzi for frequent Pinyin syllables with the predominant homophonous hanzi from the Shared Task data. This resembles a related approach to pronunciation extraction from the web (Ghoshal et al., 2009), where extraction validation and pronunciation normalization steps were found to be tremendously helpful, even necessary, when using web-derived pronunciations. One of the conclusions there was that extracted pronunciations should be used directly when available.

This is what we did in our second Non-Standard Run. We used the filtered and normalized LDC data as a static dictionary in which to look up the transcription of names in the test data. This is how the shared task problem would be solved in practice and it resulted in a huge gain in quality. Notice, however, that doing so is non-trivial, because of the data quality and data mismatch problems described above.

6 Japanese Katakana

The "English" to Japanese katakana task suffered from the usual problem that the Latin alphabet side covered many languages besides English. It thus became an exercise in guessing which one of many valid ways of pronouncing the Latin letter string would be chosen as the basis for the Japanese transcription. We toyed with the idea of building mixture models before deciding that this issue is more appropriate for a pronunciation modeling shared task. In the end, we built the same kinds of straightforward pair *n*-gram models as in the tasks described earlier.

For Japanese katakana we performed a similar kind of preprocessing as for the Indian languages: since it is possible (under minimal assumptions) to construct an isomorphism between katakana and Japanese phonemes, we chose to use phonemes as the main level of representation in our model. This is because Latin letters encode phonemes as opposed to syllables or morae (to a first approximation) and one pays a penalty (a loss of about 4% in accuracy on the development data) for constructing models that go from Latin letters directly to katakana.

For the Standard Run, we built a 5-gram model that maps from Latin letter strings to Japanese phoneme strings. The model used the same kind of Kneser-Ney smoothing and Seymore-Rosenfeld shrinking as before. In addition, we restrict the model to only produce well-formed Japanese phoneme strings, by composing it with an unweighted Japanese phonotactic model that enforces the basic syllable structure.

7 Japanese Name Kanji

It is important to note that the Japanese name kanji task is conceptually completely different from all of the other tasks. We argue that this conceptual difference must translate into a different modeling and system building approach.

The conceptual difference is this: In all other tasks, we're given well-formed "English" names. For the sake of argument, let's say that they are indeed just English names. These names have an English pronunciation which is then mapped to a corresponding Hindi or Korean pronunciation, and the resulting Hindi or Korean "words" (which do not look like ordinary Hindi or Korean words at all, except for superficially following the phonology of the target language) can be written down in Devanagari or Hangul. Information is lost when distinct English sounds get mapped to the same phonemes in the target language and when semantic information (such as the gender of the bearer of a name) is simply not transmitted across the phonetic channel that produces the approximation in the target language (transcription into Chinese is an exception in this regard). We call this *forward transcription* because we're projecting the original representation of a name onto an impoverished approximation.

In name kanji restoration, we're moving in the opposite direction. The most natural, information-rich form of a Japanese name is its kanji representation (ja-Hani). When this gets transcribed into rōmaji (ja-Latn), only the sound of the name is preserved. In this task, we're asked to recover the richer kanji form from the impoverished rōmaji form. This is the opposite of the forward transcription tasks and just begs to be described by a noisy channel model, which is exactly what we did.

The noisy channel model is a factored generative model that can be thought of as operating by drawing an item (kanji string) from a source model over the universe of Japanese names, and then, conditional on the kanji, generating the observation (rōmaji string) in a noisy, nondeterministic fashion, by drawing it at random from a channel model (in this case, basically a model of kanji readings).

To simplify things, we make the natural assumption that there is a latent segmentation of the rōmaji string into segments of one or more syllables and that each individual kanji in a name generates exactly one segment. For illustration, consider the example *abukawa* 虻川, which has three possible segmentations: *a+bukawa*, *abu+kawa*, and *abuka+wa*. Note that boundaries can fall into the middle of ambisyllabic long consonants, as in *matto* 松任.

Complicating this simple picture are several kinds of noise in the training data: First, Chinese pinyin mixed in with Japanese rōmaji, which we removed mostly automatically from the training and development data and for which we deliberately chose not to produce guesses in the submitted runs on the test data. Second, the seemingly arbitrary coalescence of certain vowel sequences. For example, *ōnuma* 大沼 and *onuma* 小沼 appear as *onuma*, and *kouda* 国府田 and *kōda* 幸田 appear as *koda* in the training data. Severe space limitations prevent us from going into further details here: we will however discuss the issues during our presentation at the workshop.

For the Standard Run, we built a trigram character language model on the kanji names (16,182 from the training data plus 3,539 from the development data, discarding pinyin names). We assume a zero-order channel model, where each kanji generates its portion of the rōmaji observation independent of its kanji or rōmaji context. We applied an EM algorithm to the parallel rōmaji/kanji data (19,684 items) in order to segment the rōmaji under the stated assumptions and train the channel model. We pruned the model by replacing the last EM step with a Viterbi step, resulting in faster runtime with no loss in quality. NSR 1 uses more than 100k additional names (kanji only, no additional parallel data) extracted from biographical articles in Wikipedia, as well as a list, found on the Web, of the 10,000 most common Japanese surnames. A total of 117,782 names were used to train a trigram source model. Everything else is identical to the Standard Run. NSR 2 is like NSR 1 but adds dictionary lookup. If we find the rōmaji name in a dictionary of 27,358 names extracted from Wikipedia and if a corresponding kanji name from the dictionary is among the top 10 hypotheses produced by the model, that hypothesis is promoted to the top (again, this performs better than using the extracted names blindly). NSR 3 is like NSR 1 but the channel model is trained on a total of 108,172 rōmaji/kanji pairs consisting of the training and development data plus data extracted from biographies in Wikipedia. Finally NSR 4 is like NSR 3 but adds the same kind of dictionary lookup as in NSR 2. Note that the biggest gains are due first to the richer source model in NSR 1 and second to the richer channel model in NSR 3. The improvements due to dictionary lookups in NSR 2 and 4 are small by comparison.

8 Results

Results for the runs are summarized below. "Rank" is rank in SR/NSR as appropriate:

	Run	ACC	F	Rank
en/ta	SR	0.436	0.894	2
	NSR1	0.437	0.894	5
ja-Latn/	SR	0.606	0.749	2
ja-Hani	NSR1	0.681	0.790	4
	NSR2	0.703	0.805	3
	NSR3	0.698	0.805	2
	NSR4	**0.717**	**0.818**	**1**
en/ru	SR	0.597	0.925	3
	NSR1	0.609	0.928	2
	NSR2	**0.955**	**0.989**	**1**
en/zh	SR	0.646	0.867	6
	NSR1	0.658	0.865	10
	NSR2	**0.909**	**0.960**	**1**
en/hi	SR	0.415	0.858	9
	NSR1	0.424	0.862	8
en/ko	**SR**	**0.476**	**0.742**	**1**
	NSR1	**0.794**	**0.894**	**1**
en/kn	SR	0.370	0.867	2
	NSR1	0.374	0.868	4
en/ja-Kana	SR	0.503	0.843	3
	NSR1	0.564	0.862	n/a

Acknowledgments

The authors acknowledge the use of the English-Chinese (EnCh) (Li et al., 2004), English-Japanese Katakana (EnJa), English-Korean Hangul (EnKo), Japanese Name (in English)-Japanese Kanji (JnJk) (`http://www.cjk.org`), and English-Hindi (EnHi), English-Tamil (EnTa), English-Kannada (EnKa), English-Russian (EnRu) (Kumaran and Kellner, 2007) corpora.

References

Arnab Ghoshal, Martin Jansche, Sanjeev Khudanpur, Michael Riley, and Morgan E. Ulinksi. 2009. Web-derived pronunciations. In *ICASSP*.

James L. Hieronymus. 1993. ASCII phonetic symbols for the world's languages: Worldbet. AT&T Bell Laboratories, technical memorandum.

A. Kumaran and Tobias Kellner. 2007. A generic framework for machine transliteration. In *SIGIR--30*.

Haizhou Li, Min Zhang, and Jian Su. 2004. A joint source channel model for machine transliteration. In *ACL-42*.

Haizhou Li, A. Kumaran, Vladimir Pervouchine, and Min Zhang. 2009a. Report on NEWS 2009 machine transliteration shared task. In *ACL-IJCNLP 2009 Named Entities Workshop*, Singapore.

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

Su-Youn Yoon, Kyoung-Young Kim, and Richard Sproat. 2007. Multilingual transliteration using feature based phonetic method. In *ACL*.

Min Zhang, Haizhou Li, and Jian Su. 2004. Direct orthographical mapping for machine transliteration. In *COLING*.