A Generalized Framework for Machine Transliteration

Jeremy Ozymandias Fallick
University of Maryland
College Park, Maryland
jfallick@umd.edu

Hal Daumé III
University of Maryland
College Park, Maryland
hal@umiacs.umd.edu

ABSTRACT

This paper details the creation of a framework for machine transliteration between two scripts. It extends the work done by Knight and Graehl on transliteration between English and Japanese using weigheted finite-state transducers (WFSTs) to represent a generative model for the process. The framework outlined in this paper is a modular hub-and-spoke model composed of WFSTs that allows transliteration between any two languages for which WFSTs are available. The composition of the WFSTs and the results of testing with all languages for which data was available are given.

1. INTRODUCTION

Translation is the transformation of utterances from one language to another preserving meaning as best as possible. However, for some phrases, such as proper names, technical and cultural terms, and concepts without clear analogues in the target language, translation isn't appropriate, because there is no fitting image in that language. In such situations, if the languages do not share a script, the mapping is replaced with transliteration, under which utterances are transformed from one script to another preserving pronunciation.

Transliteration, especially as an automated process, is not as straightforward as it may initially seem. Not all sounds are represented in all languages, and so the pronunciation of a word in one language may not be representable in the script of another. As a result, sounds must often be approximated in the target script, making transliteration a lossy process: information is destroyed by approximation, and a given transliteration does not necessarily have a unique original pronunciation. In addition, scripts do not encode the same amount of information in each character, so there is no one-to-one mapping between the characters of two scripts; in fact, there is often not even a constant-to-constant mapping.

This paper details the creation of a framework for machine transliteration between any two languages. This is more specific than transliteration between two scripts, because different languages can use the same script differently (just look at English and Gaelic). This has the obvious application of script-to-script transliteration, but it also has the effect of making it possible to "transliterate" because two different languages' uses of the same script, such that pronunciation is preserved as best as possible for a speaker of only the target language reading the result of the transliteration. This means that the framework has another application in creating pronunciation guides for loanwords.

The goal of this project was to make a framework that can be used with any two given languages, and so is not designed with any particular language pair in mind. This gave rise to a modular, hub-and-spoke structure, into which languages can be added with no change to the rest of the framework. To further the goal of generalizability, we wanted to use corpora that were not specific to particular languages; the Wikimedia projects lend themselves nicely to this purpose, since they have a goal of accumulating content in as many languages as possible (see Appendix for statistics on language distribution in the corpora).

2. APPROACH

2.1 Previous Work

In their 1998 paper, Knight and Graehl discussed their creation of a machine transliteration model between English and Japanese, with the specific goal of achieving backtransliteration from Japanese to English, i.e., determining the original English phrase given its Japanese transliteration [1]. They created a generative model for the process comprising the following stages:

1. Writing of English script
2. Reading of English pronunciation
3. Translation to Japanese pronunciation
4. Transcription to Japanese script

The original model includes an additional stage for the act of writing in Japanese script as distinct from the selection of Japanese characters. This is because Knight and Graehl expected data in the form of imaged Japanese script, which would then be processed using optical character recognition (OCR). To appropriately model the failures of OCR, the model had to include a step introducing transcription error. Since the project with which this paper deals does not include an OCR step, the generative model is limited to the four steps given above.

Each step in the generative model above can be represented as a probability distribution, which in turn can be implemented as finite-state automata: the first as a weighted finite-state acceptor (WFSA) and the rest as weighted finite-state transducers (WFSTs). The complete model for English-Japanese transliteration is produced by the composition of these automata; fortunately, automata can be reversed, so this model can handle either direction of transliteration!

Because the mapping between English and Japanese sounds, represented by step 3 in the generative model, is neither one-to-one nor constant-to-constant, Knight and Graehl used an estimation-maximization (EM) algorithm to generate the WFST. This process in outlined in their paper.

2.2 Framework

Our project's goal is a generalized framework that can operate between any two languages. This lends itself to a modular, hub-and-spoke structure. Our generative model is very similar to that of Knight and Graehl:

1. Writing of script for language A
2. Reading of pronunciation from language A
3. Translation to pronunciation in language B
4. Transcription to script for language B

Each of these, again, is modeled as an automaton. In the center of the structure is the universal International Phonetic Alphabet (IPA) hub. This can be implemented as a trivial WFSA, which accepts every string that can be composed of IPA characters; in practice, this is equivalent to leaving it out of the composition.

Each spoke represents a given language. The inner WFST transforms the pronunciation of a word under the universal IPA, i.e., in an inventory in which all sounds are available, into the pronunciation of the word in that language. In reverse, this is just an identity mapping (since no sounds need be approximated), so the WFST can be left out of the composition. The outer WFST transforms a word from its pronunciation in the language to its written representation in the appropriate script.

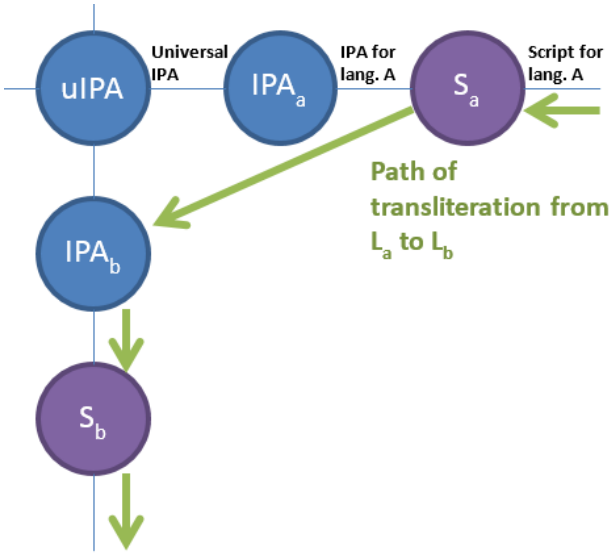

Figure 1. The transliteration framework, with an example path

With this framework, the procedure to transliterate a word from language L_a to language L_b is to transcribe it to IPA for L_a, which is equivalent to the universal IPA transcription for that word, using the outer WFST on the L_a spoke, then to transform that IPA to IPA for L_b using the inner WFST on the L_b spoke, and finally to transcribe the resulting IPA to L_b's script using the outer WFST on the L_b spoke.

3. MODELS

3.1 Implementation Note

Carmel, a simple and powerful finite-state transducer by Jonathan Graehl, was used to train and run all automata in this project [1].

3.2 Script to IPA

The corpus for these models is the complete dump of Wiktionary, the dictionary of the WikiMedia project[1]. From this, words with IPA pronunciation keys were extracted. To generate the WFST for each language, the EM algorithm used by Knight and Graehl was employed, using word-pronunciation pairs in place of loanword pairs.

3.3 IPA to IPA

For the inner WFST for a language, the IPA inventory of the language was determined by observing which IPA segments were present in the Wiktionary dump. The WFST was built as a single final state with transitions for each pairing in which a segment from the language's inventory is consumed and a segment from the universal IPA is produced, as well as transitions in which no segment is consumed and transitions in which no segment is produced.

To train these WFSTs, it was necessary to assemble pairs of loanword pronunciations. For this, we used the titles of all Wikipedia pages in the Cities category, gathering their spellings in other languages by following the interlanguage links that Wikipedia uses to connect equivalent articles on different-language editions. Cities were chosen because city names are generally adapted into other languages instead of being translated, so they are likely to be loanwords, and because they are common on Wikipedia and frequently appear in several language editions.

Once pairs of article titles were obtained, they were run through the script-to-IPA WFSTs described above for the appropriate languages in order to obtain IPA transcriptions. These paired pronunciations were then used to train the IPA-to-IPA WFSTs.

4. RESULTS

The training data in this project was randomly separated into three sets: 80% training, 10% development, and 10% test. Results of transliteration were scored by their Levenshtein distance[2] from the expected result divided by the number of characters in the expected results. The top score among the four most likely transliterations (as predicted by the model) was used to score each pair. Each language was given two scores: the average score of translating words from the language to another and the average score of translating words to the language. The results can be seen in Table A2.

5. DISCUSSION

5.1 Additional functionality from Knight and Graehl

Knight and Graehl included some functionality in their work that this project could be expanded to include. One is the use of OCR. This project uses digital text as input, but expanding it to be able to take input from images would increase its applicability. The other is a word likelihood step, which allows the program to make a more intelligent choice between possible mappings. However, this is only applicable to back-transliteration. It could be included, as long as the program can be told when it is performing back-transliteration, so it is only used for appropriate operations.

5.2 Future Work

5.2.1 Less naïve IPA-to-IPA step

The model for the inner IPA-to-IPA step used in this paper assumes a one-to-one correspondence in segments between languages, but a more accurate model would allow for multiple segments to be replaced with a single segment and vice versa. A possible approach is to use the EM algorithm from the script-to-IPA step again.

5.2.2 Metric

The metric we used to evaluate attempted transliterations does a good job of accounting for different kinds of errors, but considers all substitutions to be equally bad. A more intelligent metric would penalize less for the use of phonologically similar characters.

[1] Statistics on the corpora used in this project can be found in Table A1.

[2] The Levenshtein difference between words A and B is the minimum number of single-character insertions, deletions, and substitutions that must be done to change A into B.

5.2.3 Corpora

The Wiktionary dump served as a fantastic corpus for the script-to-IPA step, but the city corpus was far more limited for the IPA-to-IPA set, since city names are often not reproduced faithfully in other languages. Personal names, which are generally approximated as closely as possible in other languages, would be better for language pairs that use different scripts, but are not as readily available, and would not be informative for language pairs that use the same script, since the spelling of names is usually preserved within a script. A better corpus should be sought.

5.2.4 Influence from related languages

A tweak that would improve performance would be to allow languages with little data to be trained with influence from related languages which more data. This relies on the assumption that related languages are phonologically similar, and would make it possible to obtain useful, less-noisy behavior for low-data languages.

6. REFERENCES

[1] K. Knight and J. Graehl, "Machine Transliteration," *Computational Linguistics*, vol. 24, no. 4, pp. 599-612, 1998.

[2] J. Graehl, "Carmel finite-state toolkit," University of Southern California Information Sciences Institute, [Online]. Available: http://www.isi.edu/licensed-sw/carmel/.

APPENDIX

Table A1. Corpora statistics

Language	# IPA entries	# city pairs	Language	# IPA entries	# city pairs	Language	# IPA entries	# city pairs
Abaza	7	0	Manx	80	19182	Portuguese	273	612843
Afrikaans	21	85631	Hebrew	355	200560	Romanian	650	305295
Old English	1678	20283	Hindi	121	48844	Russian	231	928784
Arabic	42	22177	Fiji Hindi	78	25241	Sicilian	492	68058
Aramaic	1175	0	Hiligaynon	16	0	Scots	280	98157
Egyptian Arabic	12	8135	Croatian	25	94198	Seri	9	0
Libyan Arabic	42	0	Hungarian	10721	219918	Old Irish	11	0
Bulgarian	4224	264516	Interlingua	8	5056	Shan	1	0
Bengali	29	3010	Icelandic	2064	82989	Slovene	227	112538
Breton	192	110352	Italian	714	829918	Albanian	397	77094
Catalan	38	370511	Japanese	242	0	Serbian	74	102653
Mandarin Chinese	32	0	Lojban	1347	6139	Saterland Frisian	11	3046
Corsican	14	2805	Georgian	40	111678	Swedish	306	367662
Czech	264	317105	Guugu Yimithirr	15	0	Swahili	36	87452
Kashubian	13	3837	Gamilaraay	39	0	Syriac	21	0
Welsh	106	84192	Khmer	41	0	Tamil	11	3135
Danish	2097	257061	Korean	334	0	Tajik	94	121382
German	3722	641700	Latin	3037	159114	Thai	617	93600
Ewe	73	8586	Lombard	23	56219	Tagalog	117	140126
Modern Greek	137	140998	Lao	435	1563	Tswana	82	1694
English	19605	1069887	Lithuanian	23	167422	Tok Pisin	9	1021
Esperanto	5330	448554	Luo	13	0	Turkish	44	252631
Spanish	819	775527	Macedonian	489	114841	Taos	163	0
Estonian	10	19773	Mandinka	17	0	Uyghur	55	21571
Basque	20	251171	Malay	7	1144	Ukrainian	10	45917
Persian	574	376639	Maltese	246	14481	Urdu	50	34192
Finnish	2126	371826	Nahuatl	92	9927	Vietnamese	41	280984
Filipino	113	0	Min Nan	13	0	Martuthunira	26	0
Faroese	27	19449	Classical Nahuatl	219	0	Classical Armenian	107	0
French	11837	831045	Dutch	1288	629478	Yiddish	37	47141
Old French	29	0	Norwegian	143	307836	Yucatec Maya	34	0
West Frisian	125	55386	Occitan	24	75552	Yue Chinese	177	0
Irish Gaelic	111	46095	Polish	1390	578964			
Scottish Gaelic	109	40086	Old Prussian	36	0			
Swiss German	20	0	Pashto	241	8473			

Table A2. Transliteration scores[3]

Language	To	From	Language	To	From
Afrikaans	0.364874	0.701885	Georgian		0.849916
Old English	0.414463	0.659817	Latin	0.26677	0.820853
Arabic		0.65424	Lombard	0.25641	0.722312
Egyptian Arabic		0.547718	Lao	0.018182	0.60396
Bulgarian	0.163947	0.681601	Lithuanian	0.240931	0.862137
Bengali		0.4	Macedonian	0.038418	0.727504
Breton	0.467857	0.617725	Malay		0.413793
Catalan	0.421215	0.609492	Maltese	0.367213	0.639773
Corsican	0.833333	0.388601	Nahuatl	0.311301	0.570064
Czech	0.458534	0.745898	Dutch	0.475014	0.629512
Kashubian		0.527523	Norwegian	0.459506	0.634823
Welsh	0.497754	0.720327	Occitan	0.346026	0.646238
Danish	0.426706	0.659233	Polish	0.480445	0.714986
German	0.458967	0.537164	Pashto		0.616327
Ewe	0.390342	0.648352	Portuguese	0.503216	0.603653
Modern Greek	0.092752	0.763835	Romanian	0.454886	0.684343
English	0.186509	0.579267	Russian	0.056528	0.767493
Esperanto	0.494643	0.752227	Sicilian	0.446411	0.696462
Spanish	0.488086	0.715101	Scots	0.457447	0.711025
Estonian	0.142857	0.6489	Slovene	0.374927	0.757197
Basque	0.196721	0.653504	Albanian	0.472436	0.740868
Persian	0.221469	0.671058	Serbian	0.028369	0.713299
Finnish	0.463465	0.674484	Saterland Frisian	0.580645	0.369565
Faroese	0.278302	0.64898	Swedish	0.426888	0.638648
French	0.393805	0.505779	Swahili	0.180579	0.722561
West Frisian	0.408188	0.631654	Tamil	0.020833	1.175676
Irish Gaelic	0.41742	0.670078	Tajik	0.027273	0.678282
Scottish Gaelic	0.349518	0.560106	Thai	0.022669	0.745732
Manx	0.322796	0.652901	Tagalog	0.374446	0.726637
Hebrew	0.103395	0.624463	Tswana		0.45045
Hindi	0.089188	0.671261	Tok Pisin		0.285714
Fiji Hindi	0.427873	0.713891	Turkish	0.340317	0.714013
Croatian	0.333333	0.756377	Uyghur	0.012346	0.6916
Hungarian	0.49795	0.557865	Ukrainian		0.662248
Interlingua		0.698745	Urdu	0.146727	0.597581
Icelandic	0.496498	0.749766	Vietnamese	0.372308	0.646612
Italian	0.440775	0.642445	Yiddish	0.076321	0.738375

[3] Gaps indicate a lack of available data for testing.