6.863J Final Project

English-Japanese Transliteration

Author:
Edward Benson
Stephen Pueblo
Fuming Shih

Professor:
Dr. Robert Berwick

May 16, 2009

1 Introduction

Transliteration is the task of converting a word in one language into a sequence of characters from a different language while attempting to best approximate the native pronunciation. Transliteration is essentially a translation problem, from the set of sounds in one language to those from another language. This phonetic translation process is made more difficult by the fact that the model may need to additionally incorporate the translation between the written representations of each language and the phonetic representations.

Machine transliteration is an important topic area for both information retrieval systems and also machine translation systems. In both scenarios, it allows the machine a way to handle out of vocabulary (OOV) words when other methods fail. This is particularly important for names and technical terms, as these comprise two sets of words that are often transliterated in real text due to lack of any corresponding term in the foreign language. "Michael Jordan" in Chinese, for example, is a *transliteration* of the name Michael Jordan, rather than translation, because is it converted from one language to another solely based on sound, rather than meaning. Experiments have shown that information retrieval systems that incorporate a transliteration model have significant performance improvement when searching through foreign-language documents. [7].

Our project is an attempt at building a probabilistic transliteration model capable of transforming English words into the Japanese *katakana* writing system. We based our system after previous systems that have had success in this area. Our model is based on phonetics: it trains and operates on intermediate phonetic representations of the two languages, but we attempt a new type of phoneme alignment based on traditional linguistic study of Japanese transliteration [8] rather than probabilistic modeling.

Our modeling achieved an 86.4% average character-level accuracy across 20-fold cross-validation testing when comparing all characters transliterated. When looking at per-word accuracy, our model was able to transliterate 54.8% of all words to within 80% of their correct output, and 34.2% of words completely correctly. Our average C.A. score of 86.4% rivals that of published results.

The remainder of this paper is organized as follows: Section 2 describes a brief overview of *katakana* and Japanese phonology. Section 3 describes previous work in the ares of transliteration. Section 4 describes our approach to the problem and the system that we built. Section 5 contains a performance evaluation or our system, and Section 6 contains conclusions from this project.

2 Katakana and Japanese Phonology

The Japanese language is written using three character sets: ideographic characters imported from China, called *kanji*, and two phonetic syllabaries, called *hiragana* and *katakana*. In practice, *hiragana* is used for grammatical markings and words for which there is no ideogram, while *katakana* is used in modern times to write sound effects and transcribe foreign words. This second use of *katakana* is the reason it is the target output serialization for this project.

The Japanese language is one of the few in the world based on morae rather than syllables. A mora is a unit of sound that determine a syllable's weight. In the case of Japanese, a syllable's weight is a unit of time. One might think of Japanese as spoken to a beat: holding the *A* sound for one beat is meaningfully different than holding it for two beats. Likewise, *not* making a sound for one beat in a word is as meaningful as making one. *Katakana*, then, like *hiragana*, is more accurately described as a way to write the set of Japanese morae rather than the set of Japanese syllables, as each symbol represents not a syllable but rather a unit of sound that can fill one full "beat" of Japanese speech. All Japanese mora, and thus all *katakana* symbols, are one of four sounds:

1. a vowel
2. a consonant plus a vowel
3. the nasal *N*
4. a pause

Fundamental differences between languages such as the importance of the mora pose some of the most interesting aspects of the transliteration challenge. Many of the more well known features of Japanese phonology, such as the lack of distinction between the English *R* and *L*, for example, are challenges for transliteration, but at least they are challenges that concern features of language (consonant and vowel sounds) that have meaning and representation in both source and target language. A mora-based writing system in Japanese represents a dimension of the language – time – that has no corresponding representation in English, however. Similarly, the tones in Chinese represent a dimension – pitch – that is completely absent in English. This, in our opinions, poses one of the most fascinating challenges of transliteration that we have realized over the course of this project: that the source and destination feature spaces are not necessarily of the same dimensionality.

3 Previous Work

Approaches to transliteration range from the completely hand-tuned models to completely learned. We survey three points along this spectrum here to show the different possibilities.

Traditional linguistic approaches, such as those found in the Dictionary of Intermediate Japanese Grammar [8] provide guidelines for transliterating Japanese based on the International Phonetic Alphabet. While this source does not explain how these transliteration guidelines were created, our assumption is that they are based on years of experience translating the two languages rather than an analytic model applied to example data.

Knight and Graehl [5] propose a multi-step approach to English-Japanese transliteration and back-transliteration based on probabilistic finite state transducers (PFSTs). Their scheme uses PFSTs to map English words to English phonemes, English phonemes to Japanese phonemes, and then Japanese phonemes to Japanese words. Their original work also includes a final step to model the output of an OCR system which is omitted in the equations and figures here. Figure 1 shows this subset of their model.

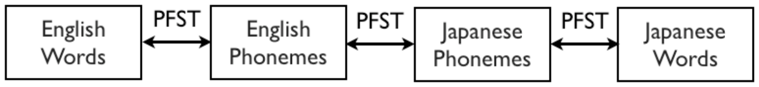


Figure 1: Knight and Graehl's Transliteration Model

With their model, they then describe the task of *back*-transliteration from Japanese to English as that of solving for

$$\arg\max_{w} \sum_{e} \sum_{j} P(w)P(e|w)P(j|e)P(k|j)$$

Where w is the English word, e is the corresponding sequence of English phonemes, j is the sequence of Japanese phonemes, and k is the *katakana* word.

Knight and Graehl describe their decision to use intermediate phonetic representations of the English and Japanese alphabets as critical to the success of their model because of the misalignment between English letters and Japanese morae. English, for example, separates consonants, such as *K*, from the vowel sounds that may follow it, such as *A*. Japanese instead represents only wholly formed morae, such as the combined カ (KA) sound.

This particular example, as well as others result in generalization difficulties when trying to model the transliteration between the two languages. For example, the possible Japanese morae, and thus the possible *katakana*, that begin with the *K* are カ (KA), キ (KI), ク (KU), ケ (KE), and コ (KO). Because these are all completely separate characters, a phonetic model based on *katakana* as a serialization format has trouble representing that the K sound in カ (KA) is the same K sound in コ (KO).

Knight and Graehl's intermediate phonetic Japanese alphabet solves this problem by separating out consonant+vowel combinations as above (カ → K A) as well as collapsing other characters together (ッカ → KK A). This this new representation, they are able to collapse similar Japanese consonant and vowel sounds into the same representational states.

Li, Zhang, and Su [6] provide an alternative approach that attempts to find a direct orthographical mapping between the two languages. This approach attempts to map directly from one set of characters to another set of characters, skipping an intermediate phonetic representation. Their approach uses a joint source-channel model to map from English to Chinese. Instead of solving for the *native→foreign* transliteration as $\arg\max_F P(N,F) = P(N|F) * P(F)$, which models how the native representation can be mapped onto the foreign representation, they propose a model that represents how both native and foreign representations are generated simultaneously:

$$\begin{aligned} P(N,F) &= P(n_1, n_2 \dots n_K, f_1, f_2 \dots f_K) \\ &= P(<n,f>_1, <n,f>_2 \cdots <n,f>_K) \\ &= \prod_{k=1}^{K} P(<n,f>k| <n,f>_1^{k-1}) \end{aligned}$$

A particular challenge in this model is the alignment of the units $<n,f>_i$ seen above. Given a native word $N = x_1, x_2 \dots x_a$ and foreign transliteration $F = y_1, y_2 \dots y_b$, just how should these be split up evenly into $<n,f>_i$ pairs? Li *et al* propose introducing a parameter γ which represents an alignment between the two words that assigns each n_i and f_i to some subsequence of $x_1 \dots x_a$ and $y_1 \dots y_b$, respectively. For example:

$$<n,f>_1 = <x_1, x_2, x_3, y_1, y_2>$$

$$<n,f>_2 = <x_4, y_3>$$

With the character chunking between characters parameterized, the transliteration from native to foreign language may thus be modeled as

$$\bar{F} = \arg\max_{F,\gamma} P(N,F,\gamma)$$

Most transliteration literature we surveyed across many languages (Hindi, Chinese, Japanese, Korean, etc) tended to take an approach similar to one of the two probabilistic techniques described above. The particulars of their mathematical models, of course, varied from paper to paper. In general, three general themes arose across all of the papers on transliteration:

- Approaches can be divided into either phonetic or alphabetic. Phonetic approaches first either solve, or assume a solution to, the problem of converting source and destination word into phonetic representation. The transliteration task is then done at the level of phonemes. Alphabetic approaches attempt to avoid the need to have intermediate phonetic representations by converting directly from the spelling in the native language into the spelling in the foreign language.

- For models that require training examples, a point of significant difficulty is the alignment between source and target phonemes. This is handled in a range of different ways across the literature, from expanding the feature space to include parameterization of all possible alignments to manual intervention.

- In all cases, notions of "correctness" of a transliteration result is complicated by the fact that there often exist multiple possible ways to transliterate a word, all of which may be considered of varying degrees of correctness. The Japanese language, for example, has seen the recent creation of new combinations of *katakana* characters to stand for foreign sounds impossible to pronounce in traditional Japanese. This means that there are now many English words for which multiple correct variations of transliterated Japanese exist. This flexibility has spawned literature aimed not to find a single transliterated word in Japanese but rather to examine the entire set of possible Japanese *katakana* sequences for a given foreign word.

4 Approach

We performed English to Japanese transliteration using an approach similar to that of Knight and Graehl in that it was phonetic rather than alphabetic. Our approach, shown in Figure 2 used similar high level steps but with different techniques for implementation. In particular, where Knight and Graehl used probabilistic finite state transduction to convert from written to phonetic representations of source and target language, we used our own method (equivalent to non-probabilistic finite state transduction) for converting between Japanese written and phonetic forms and outsourced the English-phoneme conversion to the DECTalk software package.

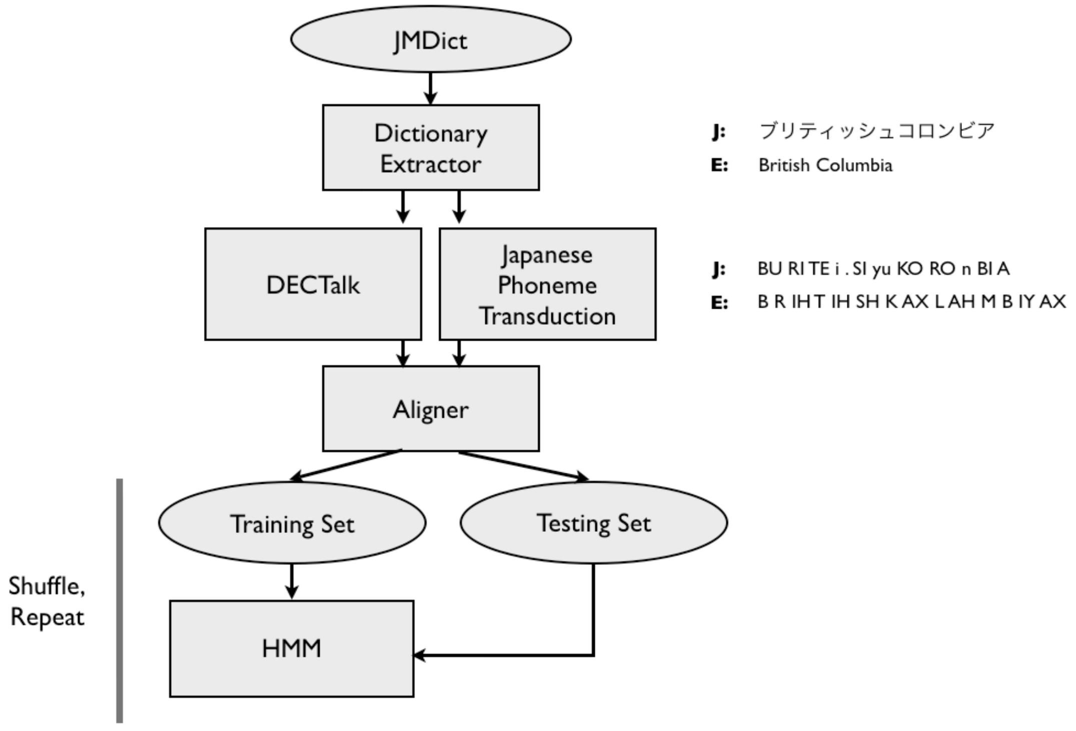

JMDict
Dictionary Extractor
DECTalk
Japanese Phoneme Transduction
Aligner
Training Set
Testing Set
HMM
Shuffle, Repeat
J: ブリティッシュコロンビア
E: British Columbia
J: BU RI TE i . SI yu KO RO n BI A
E: B R IHT IH SH K AX L AH M B IY AX

Figure 2: System Diagram

For the phonetic conversion from Japanese to English, we used a Hiddem Markov Model trained on fully observed Japanese and English phonetic sequence examples. These examples were build with an alignment algorithm that we devised targeted to the particular interactions between the Japanese and English phonologies. This algorithm allowed us to automatically align phoneme sequences between the two languages, despite different phonetic representations, and we believe it played a large role in boosting our result accuracy. We tested our results using bigram, trigram, and 4-gram models and 20-Fold cross validation.

4.1 Building a Training Corpus

Our approach was built assuming the existence of a English-to-*katakana* training set that would provide training examples of English words and their official equivalent when transliterated into *katakana*. We were unable to find such a corpus available for download, so instead built an extractor to approximate such a data set that we ran against the open source `JMDict` dictionary file [2].

Our extractor examined each Japanese-English entry in `JMDict` and extracted only those that met all of the following heuristic properties, designed using knowledge of the Japanese language:

1. The English phrase consisted of only alphabetical characters and spaces (no punctuation, for

example)

2. The Japanese phrase consisted of only *katakana* characters

3. The English phrase contained at most three words

4. The English phrase contained at least one lowercase character

5. The Japanese phrase did not appear to be an onomontopea

The first four rules were designed to try to narrow the set of extracted phrases to short, non-acronym words and phrases that appear as direct transliterations into *katakana*. The fifth rule was designed to weed out the other primary use of *katakana* in Japanese: the expression of onomontopea. Onomontopeas in Japanese generally occur as doubled sound sequences, such as *pakupaku* (the act of eating in large mouthfuls) and the resulting *gorogoro* (the rumblings of an upset stomach). Our extractor exploited this regularity by simply discarding any *katakana* entry whose first half was equal to its second half.

The resulting dictionary was 17798 entries long. This data set is not without noise: many *katakana* are either English abbreviations, such as *pasokan* for *personal computer*, or words from non-English languages, such as *arubaito* from the German word for part-time job. These terms are not transliterations, and thus provide us with bad training examples. Nevertheless, our base assumption was that enough of the words extracted would in fact be good transliteration examples that they would overcome the bad examples when training the parameters of our model.

4.2 Phonetic Transduction

The next step in constructing our training corpus was the phonetic transduction of both the Japanese and English word pairs that we extracted. We devised our own representation for Japanese phonetic elements and wrote a program that transduced these elements from Japanese *katakana*, and we used the DECTalk package for transduction of English into English phonemes.

4.2.1 Japanese Phonetic Transduction

We created our own, Roman character-based, phonetic representation of Japanese and wrote a program in Objective-C that would transform back and forth between *katakana* and our own custom phonetic representation of Japanese. This transformation is 1-to-1: our roman character representation output can be converted back perfectly into the original *katakana* that produced it. Knight and Graehl pointed out that such a scheme is necessary in order to generalize certain sounds that occur in Japanese but lack a *katakana* representation on their own. Our scheme improves upon the one provided by Knight and Graehl by allowing for further generalization.

We initially considered using the standard *romanji* romanization of Japanese as this phonetic alphabet, but we found it lacks two key features necessary for our use. The first two rows of Table 1 demonstrated that *romanji* is a nondeterministic mapping back into *katakaa.* Notice how the sequence *ana* could mean either アンア or アナ in *katakana.* We fixed this problem in our representation by differentiating between the consonant n, denoted *N*, and the nasal n, denoted *n.* The latter two rows in Table 1 demonstrate how *romanji* does not allow for generalization of the **find-term-full-stop** full stop found in Japanese because it represents it by doubling the following consonant. Knight and Graehl's representation also suffers from this fault. We instead represent this element with a period, allowing it to have an identity across all consonants which follow it.

These are just some of several changes to *romanji* we made to better generalize the Japanese sound system. Our dictionary parser project can be examined for the full reversible mapping of *katakana* to our custom representation.

Table 1: Disadvantages of Romanji as a Phonetic Serialization

Katakana	Romanji	Our Phonetic Transcription
アンア	ana	AnA
アナ	ana	ANA
キッテ	kitte	KI.TE
キップ	kippu	KI.PU

4.2.2 English Phonetic Transduction

The DECtalk API was used to transform English words into their phonetic structure, which is composed of phonemes (e.g., eyeline=AY L AY N). Phonemes are the smallest linguistically distinctive unit of sound. DECtalk was created largely by Dennis Klatt in the 1980s, and it initially ran on machines specifically built for DECtalk [9]. As time progressed and computational power increased, instead of taking up the majority of a CPU's load, DECtalk could run on any average computer using a minimal amount of resources. This makes DECtalk a prime choice in doing the phonemic transcription of the thousands of English words in our corpus.

Using DECtalk's API, it was possible to feed our phonetic transformation code an English word, and DECtalk would output numerical values to use in its ARPABET lookup table to find the corresponding phonemes for the word. For personal preference, we decided to instead use the CMU Sphinx Phone Set. Therefore, whenever we received a numerical phonemic value to use in DECtalk's ARPABET lookup table, we intercepted and pointed that value to our own CMU Sphinx Phone Set lookup table that was created. Once the phoneme was obtain from our lookup table we printed it to a text file, which would be used later in our alignment function. Our alignment process is discussed in detail in Section 4.3

For clarification, a phone set is a set of phonemes that represents the phonetic sounds that particular phone set is trying to capture. Different phone sets use different phoneme labels to capture the phonemic sounds of a word or utterance.

4.3 Phoneme Alignment

We built an algorithm that attempts to align our Japanese and English phonemes into a fully-observed set of training data for use in training the parameters of a Hidden Markov Model. To start with, the set of Japanese phonemes J and English phonemes E are of different lengths; these must be aligned for us to use them as training data. As noted before in the paper, past work on this problem have taken a generalized approach of either deriving these alignments probabilistically or incorporating all possible alignments into the parameters of the model to be learned.

We present an algorithm that takes a different approach, grounded in traditional linguistic work specific to English-Japanese translation [8]. This approach uses phonetic English-Japanese mappings published in a Japanese reference material to provide alignment guides that we call *anchors* to help an alignment algorithm produce well-formed training data. Because our transliteration model treats Japanese sounds as the hidden states and English phonemes as the observations, our alignment algorithm fits the Japanese sounds onto the English instead of the other way around. We believe that our work in this area is the main area of contribution to be found in this project.

Our algorithm is described by the following problem statement:

Problem Definition. Given a sequence $\vec{J} = j_1 \dots j_n$ of Japanese phonetic symbols (defined above) and a sequence $\vec{E} = e_1 \dots e_m$ of English phonemes, find a new sequence $J' = j'_1 \dots j'_m$ where each $j' \in J'$ contains one more more contiguous elements of $j \in J$ and `concat(J) == concat(J')`.

Our algorithm achieves this using a four-step process:

1. **Anchor.** Locate anchors in the English phoneme sequence and find their corresponding anchor in the Japanese sequence (if it exists).

2. **Chunk.** Subdivide the English and Japanese phoneme sequences into $n + 1$ sub-problems, where n is the number of anchors that were mapped from English to Japanese.

3. **Align.** Align each sub-problem.

4. **Recombine.** Recombine the aligned sub-problems back into a full word-length sequence.

We will now visit each of these steps to explain it in further detail and provide an example. The running example for this explanation will be the English word *British Columbia.* Table 2 shows the output of the dictionary extractor and phonetic transduction for this word.

Table 2: British Columbia Training Example

Word	`British Columbia`
DECtalk Output	`B R IH T IH SH K AK L AH M B IY AX`
Katakana	ブリティシュコロンビア
Phonetic Japanese	`BU RI TE i .  SI yu KO RO n BI A`

4.3.1 Achoring

Our anchoring approach is designed to divide the problem of phoneme alignment into multiple sub-problems that are more likely to be trivial to solve. We do this based on the fact that certain sounds between English and Japanese are transliterated with more regularity than others. Namely, we exploit the fact that Japanese mora construction rules dictate that any consonant (except the nasal *n*) must be followed by a vowel sound. This results in more predictable transliteration of consonant sounds into Japanese than vowel sounds. We used a mapping of the International Phonetic Alphabet onto *katakana* [8] to build Table 3 of affinities between DECTalk English consonant sounds and Japanese consonant sounds.

Table 3: English-Japanese Consonant Affinities

DECTalk Consonant	Japanese Consonant
AXR, R, L	R
B,V	B
D,JH,	D
H,F	H
G,JH	G
K,Q	K
M	M
N	N,n
P	P
R	R
S,J	S
T,CH	T
W	W
Z,JH	Z

Some of the mappings in Table 3 might appear strange (for instance `J->S`), but that is simply due to the inevitable edge-cases that occur when a non-Western sound system is represented in Roman characters.

Given this table, our algorithm performs the anchoring step of the alignment process. To summarize this process (some nuances are glossed over), our code steps through the English phonemes until it finds an anchor. When the algorithm finds an English anchor, it will search through the Japanese until it

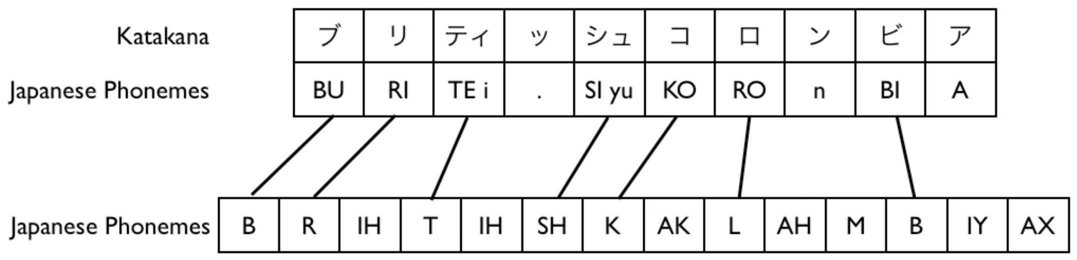


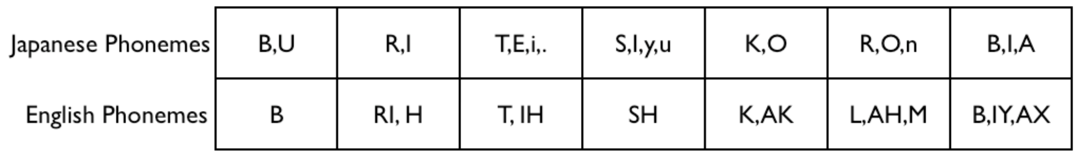
Figure 3: Anchoring British Columbia

finds an acceptable matching anchor in Japanese. If it does not find a matching anchor, the algorithm skips to the next English anchor and continues. If it does find an anchor, it marks the locations of these anchors. On the example of "British Columbia", this anchoring step has the result shown in Figure 3.

Note that this anchoring step does not detect that the n and M are equivalent between the two representations. This will be reconciled during the alignment process.

4.3.2 Chunking

The next step in our algorithm is to "chunk" the problem into several sub-problems based on the anchors found. Our code iterates through the anchors found and, upon arriving at each one, slices off all preceding Japanese and English phonemes into one chunk. The anchoring found for "British Columbia" would thus result in the chunks shown in Figure 4



Japanese Phonemes	B,U	R,I	T,E,i,.	S,I,y,u	K,O	R,O,n	B,I,A
English Phonemes	B	RI, H	T, IH	SH	K,AK	L,AH,M	B,IY,AX

Figure 4: Chunking British Columbia

Each of these chunks will now be aligned separately, with the expectation that less alignment error will be introduced into these smaller sub-problems because they have already been split upon high-probability points of alignment.

4.3.3 Alignment

The next step is to align each corresponding chunk of English and Japanese. The alignment function works in the following manner: it first tries to see if the number of phonemes in the English word chunk matches the number of syllables in the Japanese word chunk. If so, the alignment function is done. If not and the number of English phonemes is greater than the number of Japanese syllables, the alignment function will try to split one of the Japanese syllables into two smaller sub-syllables. If the number of English phonemes is less than the number of Japanese syllables, the alignment function will try to merge two of the Japanese syllables together.

This process is repeated over each chunk until the number of Japanese character segments and the number of English phonemes match. If the process reaches a state where it determines that alignment will be impossible, it returns an error and the process simply skips this dictionary entry as a training example. Errors such as this will happen when there are certain severe mismatches in the number of Japanese and English states that can not be reconciled by splitting or recombining subsequences of the Japanese phonetic transcription.

This process can be described by the following pseudocode:

```
# Returns a set of japanese states that is the same
# in length as english
align_chunk(J, E):
   J’ = J.copy
   while (|J’| > |E|):
       collapse(J’)
   while (|J’| < |E|):
       split(J’)
   return J’

# Picks a Japanese state to merge and merge it
collapse(J’):
   # Pick two j’ in J’ to merge together.
   lowest_cost = INF
   lowest_cost_index = 0

   # Count backwards so that we prefer the earlier match
   # among equal cost ones
   for (i=|J’|-2...0):
      cost = cost(J’[i]) + cost(J’[i+1])
      if (cost < lowest_cost):
         lowest_cost = cost
         lowest_cost_index = i

   merge(J’, lowest_cost_index, lowest_cost_index+1)
   return J’

# Pick a Japanese state to split and split it
split(J’):
   highest_cost = -INF
   highest_cost_index = 0

   for i in range(J’):
      if cost(J’[i] > highest_cost):
          highest_cost_index = i

   if len(J’[highest_cost_index]) > 1:
      split_char_sequence(J’, highest_cost_index)
   else:
      Error # This can not be split

   return J’

cost(a):
   return |a|
```

An alternative version of this algorithm, which works using only the `merge` operation, provides equivalent performance using a modified version of the Japanese input in which each character of the phonetic sequence is initially separated.

4.3.4 Recombination

The final step is to simply recombine all the chunks into one ordered list. Figure 5 shows the two aligned lists our algorithm outputs for “British Columbia”:

These two aligned lists are then added as a training example for our Hidden Markov Model, with the sequence of Japanese as the hidden state sequence and the sequence of English phonemes as the English

Japanese States	BU	R	I	TE	i.	Slyu	K	O	R	O	n	B	I	A
English Phonemes	B	R	IH	T	IH	SH	K	AK	L	AH	M	B	IY	AX

Figure 5: Aligning British Columbia

observations.

4.3.5 Alignment Performance

Of the 15233 candidate training example produced by our dictionary extraction script, our aligner was able to align 12668 of them, a success rate of 83%. This list of aligned words was then used in HMM training and cross-validation.

4.4 Phoneme-Phoneme Transliteration HMM

For mapping English phonetic representation to Japanese phonetic presentation, we used a first order Hidden Markov Model trained from the previous aligned phoneme sequences output by our alignment process. In a hidden Markov model, the state is not directly visible, but output dependent on the state is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. In this project, we model the Japanese phonetic sequence as the hidden states in HMM that will generate the observed English phonetic sequence.

After the training, the HMM will have two probability matrixes, the transitional probability matrix T and the emission probability matrix E. The transitional probability $t_{i,i+1}$is the probability of Japanese phoneme state transiting from position i to position $i+1$. The emission probability e_i models the probability of the state emitting the observed output at position i.

We manipulated our aligned Japanese-English training sequences such that we could test the performance of bigram, trigram, and 4-gram models of transliteration with the HMM.

5 Evaluation

5.1 Evaluation Strategy

Unlike machine translation, there is no standard for measuring the result of transliteration. In [4], the evaluation has two levels of accuracy, i.e. character-level accuracy($C.A.$) and the word level accuracy ($W.A.$)

$$C.A. = \frac{L - (i + d + s)}{L} \tag{1}$$

$$W.A = \frac{\#\ of\ correct\ nouns\ generated}{\#\ of\ tested\ nouns} \tag{2}$$

In (1), L is the length of the standard transliteration of a given English noun from our own phonetic transduction, and i, d, and s are number of insertion, deletion, and substitution respectively. The "Levenshtein distance" [1] was calculated using bottom-up dynamic programming, which calculates the best score of current position based on the previous ones. Following is pseudocode for calculating this score:

```
int LevenshteinDistance(char s[1..m], char t[1..n])
// d is a table with m+1 rows and n+1 columns
declare int d[0..m, 0..n]

for i from 0 to m
```

```
  d[i, 0] := i
 for j from 0 to n
  d[0, j] := j

 for i from 1 to n
  for j from 1 to m {
   if s[i] = t[j] then
     cost := 0
   else
     cost := 1
   d[i, j] := minimum(
                d[i-1, j] + 1,     // insertion
                d[i, j-1] + 1,     // deletion
                d[i-1, j-1] + cost   // substitution
             )
  }
  return d[m, n]
```

The equation in (2) gives the absolute correctness evaluation of the transliteration result. However, a name itself often has many acceptable alternatives in transliteration(e.g. the word "computer" has more 7 variants in transliteration[7]). Thus, we will also measure the percentage of those transliterations that are "acceptably correct", of which the score *C.A.* is between 1.0 and 0.8. This measurement was referred to as *C.A.* Distribution (*C.A.D*) in [3]:

$$C.A.D. = \frac{\#\ of\ nouns\ with\ C.A. \in [r_1, r_2)}{\#\ of\ tested\ nouns} \tag{3}$$

5.2 Cross-Validation Performance

We used k-fold cross-validation($k = 20$). The corpus, consisting of 12,668 aligned phonetic sequences, was divided into k subsets with random distribution, and one set was used for testing while the rest was used for training the HMM. This process was repeated ten times, and the results were averaged over all trials.

In each trial of our experiments, we sumed up the *C.A.* and C.A.D. scores of the results of all English words, and computed the average. In keeping with our expectations from existing general knowledge about n-gram modeling, we found that trigrams provided the best performance. This is due to the fact that 4-gram modeling enlarged the state space enough that it was both too sparse and also too overfit to the training data set.

Figure 6 shows our performance when using a bigram model to train and test the system. This histogram arranges our model's *katakana* guesses according to how correct guess each was: with buckets ranging from 0-10% correct to 90-100% correct as measured by our `C.A.` score. We see that the majority of the mass (56.54% of it) is in the 80% to 100% region.

Figure 7 shows our performance when using a trigram model. Here, only 54.77% of the words tested were within 80% to 100% correctness, which is less than the bigram model, but as we will see in Table 4, the trigram model has better W.A. and C.A.D. scores.

Table 4 shows that both the C.A. and W.A. scores are best with the trigram model. Recall that the C.A. measures the character-per-character performance of the model's output, and the W.A. models the ratio of output words that were completely correct. The final columns show the percentage of output words that were within 80% and 70% correctness for the two models.

The C.A.D precision shows that both our bigram and our trigram models are capable of generating transliterated Japanese with roughly 75% accuracy on a character-per-character basis.

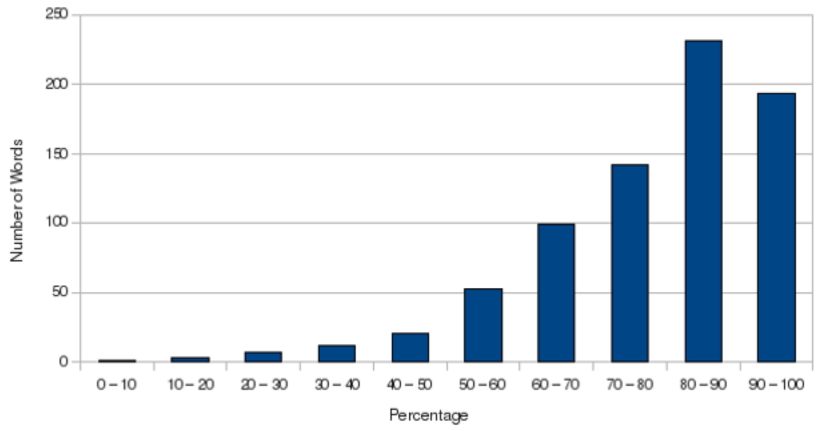


Figure 6: Table of 2-Gram Word Correctness

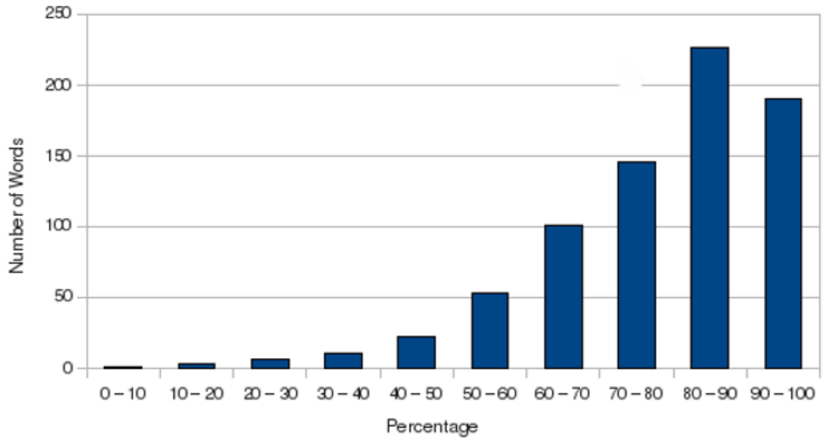


Figure 7: Table of 3-Gram Word Correctness

Table 4: Table of N-Gram Correctness

	C.A.	**W.A.**	**C.A.D [0.8,1.0)**	**C.A.D[0.7,1.0)**
2-Gram	77.59%	12.97%	56.54%	75.49%
3-Gram	83.38%	34.21%	54.77%	73.85%
4-Gram	80.49%	32.15%	N/A	N/A

5.3 Errors Analysis

We used a compiled list of transliterations that were incorrect at the word-level, but acceptable (over 80%) at the character level. We examined the transliteration errors in this list to identify whether or not there was commonality between the errors across words. If so, this would provide a clear idea of what further tuning may need to be done to our model to achieve better performance. Table 5 shows a listing of the common categories of error we found, along with examples of some of the errors that fell into these categories.

NOTE: Latex is incorrectly rendering the Katakana dash character as a space, meaning the katakana in PDF form below is ***not*** *rendered correctly for any word whose phonetic Japanese representation contains a dash (-) character.*

Table 5: Errors common to our model

English phonemes	**Correct Japanese**	**Guessed Japanese**
Duplicating or missing a doubled vowel sound		
`D IY M AX N`	`DE-MOn` デ□モン	`DEMOn` デモン
`S EH SH IX N L IY D ER`	`SESIyonRI-DA` セションリ□ダ	`SESIyonRIDA-` セションリダ□
`OW V ER L AE P IX`	`O-BARA.PU` オ□バラップ	`O-BA-RA.PU` オ□バ□ラップ
Change “EI” to “E-” or “Ei” to “I”		
`D EY T R EY D ER`	`DEITORE-DA-` デイトレ□ダ□	`DE-TORE-DA` デ□トレ□ダ
`K AX L EH K T IX V IX Z AX M`	`KOREKUTEiBIZUMU` コレクティビズム	`KOREKUTIBIZUMU` コレクチビズム
Change “A” to “O”		
`K AA L AX M CH AA R T AX`	`KARAMUTIya-TO` カラムチャ□ト	`KORAMUTIya-TO` コラムチャ□ト
`SH IH K AO G OW`	`SIKAGO` シカゴ	`SIKOGO` シコゴ
Missing pause “.”		
`F IH L IX P IX`	`HUiRI.PU` フィリップ	`HUiRIPU` フィリピ
`S T AE F`	`SUTA.HU` スタッフ	`SUTAHU` スタフ

From the testing results and the analysis on the errors, we found that about 50% of the words incorrect were those with only minor errors on sounding variations(e.g. vowels that sound alike, length of the sounding, and rhythm of the sounding). So, the difference here could be seen as variation of transliteration rather than errors. In fact, keeping all similar transliteration variant[7] increases the performance of task in cross-lingual retrieval.

Two of the most prevalent errors concerned two dimensions of Japanese for which English has no equivalent: the one mora-length pause between syllables that can occur within a word (denoted as a period in our phonetic serialization), and the one mora-length extension of a vowel sound that can occur within a word (denoted as a dash in our phonetic serialization). While we expected that these units of sound should be no different than any other, since they too are represented in the character serialization, it appears that more complex phonetic interactions are at play that our phonetic representation did not sufficiently capture.

Another common error we found related to the combination of E and I vowel sounds. Section 3 mentioned that certain modern changes to the rules of *katakana* have been made to accommodate foreign vowel sounds not found in native Japanese speech. These changes complicate the ability to have one single correct transliteration of a foreign phoneme sequence: one single phoneme sequence in English may map to a variety of different Japanese serializations in the dictionary, depending on the time period in which the word was first transliterated into Japanese.

The possible E+I combinations in *katakana* are exactly one of these modern developments, making it a difficult set of vowels to learn to apply correctly. The phonetic sequence ティ (`TEi`), for example (seen in the Table 5 above) is a modern notation for the sound “tea” in English, which is impossible to pronounce with the traditional Japanese sound set.

5.4 Out of Dictionary Testing

We performed a few final fun experiments by searching for out-of-dictionary words on Google using the output of our model. We chose six famous proper nouns that we suspected might be transliterated into

Japanese, ran them through our model, and then entered our model's output in Google. Table 6 shows the results.

Table 6: Google Searches for Out-Of-Dictionary Words

E. Word	**E. Phonemes**	**Japanese Guess**	**Google Results**
Twitter	`T W IH T ER`	`TOUITA` トウイタ	8
Microsoft	`M AY K R OW S AO F T AX`	`MAIKUROSOHUTO` マイクロソフト	52,300,000
Katrina	`K AX T R IY N AX`	`KATORINA` カトリナ	13,200
California	`K AE L AX F OW R N Y AX`	`KARAHUo-NIA` カラフォ□ニア	10
Miami Beach	`M AY AE M IY B IY CH`	`MIAMI-BI-TI` ミアミ□ビ□チ	0
Hillary Clinton	`HH IH L EH R IY K L IH N T AX N`	`HIRERI-KURInTOn` ヒレリ□クリントン	9,750

Of the six words chosen, it appears Microsoft, Katrina (the hurricane), and Hillary Clinton were successful transliteration results. The other three were either transliterated incorrectly by our model, or simply are not used in transliterated form in Japanese.

NOTE: Again, our Latex build is mis-rendering the katakana dash (-) character as a space, so the above katakana words for California, Miami Beach, and Hillary Clinton can not be copied and pasted into Google from this PDF to achieve the same result. The top three rows render fine, however. Our apologies!

6 Conclusions

This project explored a modified approach to the challenge of English-Japanese transliteration. We performed a literature review to examine existing techniques in machine-transliteration between multiple language pairings, extracted a training set of data from freely available sources, devised a phonetic serialization for Japanese that improved upon existing serializations available, and devised an algorithm to automatically align Japanese and English phoneme sequences. We then performed k-fold cross validation testing on a Hidden Markov Model used to recover the Japanese phonetic sequence given an English one.

We were surprised to achieve a transliteration accuracy so high – 83.38% accurate on a per-character basis using a trigram model. These results are on par with what other literature achieves through purely probabilistic methods, without the phonetic anchor-based alignment algorithm we introduced. We believe that with more tweaking of the training set, phonetic representation, and alignment algorithm, our strategy has the potential to perform even better.

In class the general debate between modeling the rules of the world versus treating everything as a learnable parameter came up several times. Both certainly have clear advantages and disadvantages. We feel our approach was a successful attempt to use *both* techniques to leverage their strength: we drew upon hand-tuned language modeling for our anchor-based phoneme alignment algorithm, but then we transformed the output of that into N-Grams and fed them to a Hidden Markov Model.

If we were to perform a second iteration of this project, we would take a closer look at our phoneme alignment algorithm and try to find out whether we could tweak it for better alignment within a chunk. We feel that our anchoring strategy worked well for us, but there is room for improvement in our chunk alignment algorithm. This is evidenced by the types of errors that our model made: they were almost exclusively vowel sound or vowel duration errors rather than consonant errors.

7 Appendix: Running Our Code

To run our code, run the following checkout command with username/password of redsox/redsox.

```
svn co http://svn.edwardbenson.com/katakana katakana
```

The dictionary file is already prebuild (`data/dictionary.txt`) containing 4-tuples of English, Katakana, Japanese Phonemes, and English Phonemes. The main file to run is `aligner/hmm.py`. This **must** be run with Python 2.5 and NLTK 0.9.7 or 0.9.8. There is a bug in the most recent version of NLTK (0.9.9) that will prevent it from working.

Inspect the `main()` method of `hmm.py` to see the different ways you can run the script. Running it as it is checked in will result in a cross-validation test using 2-Grams. (Note that the `ngramorder` variable you see in `hmm.py` is one *less* than the N of the N-Grams you are running with..)

References

[1] Levenshtein distance - wikipedia, the free encyclopedia.

[2] Jim Breen. Edict.

[3] Wei Gao, Kam-Fai Wong, and Wai Lam. *Improving Transliteration with Precise Alignment of Phoneme Chunks and Using Contextual Features*, pages 106–117. 2005.

[4] In-Ho Kang and GilChang Kim. English-to-Korean transliteration using multiple unbounded overlapping phoneme chunks. In *Proceedings of the 18th conference on Computational linguistics - Volume 1*, pages 418–424, Saarbrcken, Germany, 2000. Association for Computational Linguistics.

[5] Kevin Knight and Jonathan Graehl. Machine transliteration. 1998.

[6] Haizhou Li, Min Zhang, and Jian Su. A joint source-channel model for machine transliteration. 2004.

[7] Yan Qu, Gregory Grefenstette, and David A. Evans. Automatic transliteration for Japanese-to-English text retrieval. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 353–360, Toronto, Canada, 2003. ACM.

[8] Seiichi Makino and Michio Tsutsui. A dictionary of intermediate japanese grammar, 1995.

[9] W. I. Hallahan. DECtalk Software: Text-to-Speech Technology and Implementation. *Digital Technical Journal*, 7(4):5–19, April 1995.