Pinyin to Hanzi Conversion
with Hidden Markov Models

Lucas Freitas and Cynthia Meng
May 21, 2013

1. Introduction to Pinyin

The Chinese language is composed entirely of characters, with each character representing generally at least one word. When different characters are combined, different words might also be formed. Before modern technology, characters were written solely with brush strokes. During the 1950s, however, a system called pinyin was developed to aid the transcription of Chinese characters into the Latin alphabet.

Pinyin proved incredibly useful during the latter half of the century, as it was used to help teach the language to non-native Mandarin speakers, and eventually would be used as the main input method for entering Chinese characters into computers. It is still used today in Chinese language programs around the world, as it is one of the most efficient methods of teaching Chinese to native speakers of Latin-alphabet-based languages, such as English, Spanish, and others.

As a brief overview of Pinyin (of which a small amount of knowledge will help with the understanding of this project), there are five different tones in Mandarin Chinese, which can be represented by using the tone marks above vowels in the Pinyin of a character (Table 1).

<table>
<thead>
<tr>
<th>Tone</th>
<th>Pinyin</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat or level (first)</td>
<td>m¯a</td>
<td>mother</td>
</tr>
<tr>
<td>Rising (second)</td>
<td>m´a</td>
<td>hemp</td>
</tr>
<tr>
<td>Falling-Rising (third)</td>
<td>m`a</td>
<td>horse</td>
</tr>
<tr>
<td>Falling (fourth)</td>
<td>m˘a</td>
<td>scold</td>
</tr>
<tr>
<td>Neutral</td>
<td>ma</td>
<td>question</td>
</tr>
</tbody>
</table>

Table 1: The Mandarin Chinese tones

Both of us have a strong interest in the Chinese language and the relationship between characters and Pinyin, and were interested in implementing a project that would somehow draw these two integral components of the Chinese language together.

Inspired by the word segmentation problem set from earlier on in the course, we decided to attempt an implementation of Pinyin to Hanzi conversion involves a statistical model.

<table>
<thead>
<tr>
<th>Pinyin (no tone)</th>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hen</td>
<td>很</td>
<td>to hate</td>
</tr>
<tr>
<td>hen</td>
<td>很</td>
<td>very</td>
</tr>
<tr>
<td>shi</td>
<td>十</td>
<td>ten</td>
</tr>
<tr>
<td>shi</td>
<td>是</td>
<td>to be</td>
</tr>
<tr>
<td>he</td>
<td>和</td>
<td>and</td>
</tr>
<tr>
<td>he</td>
<td>喝</td>
<td>to drink</td>
</tr>
</tbody>
</table>

Table 2: Examples of heterophones in Chinese
3. Heterographs
In Chinese, a given syllable could have up to five different tones (including the neutral tone), each of which could stand for a different character. When a piece of text is presented without tones marks, it becomes exceedingly difficult to discern which character is the correct one to use. In this case, using context clues is particularly helpful, which our program would need to accommodate. Examples of this phenomenon are in Table 2.

4. Homographs (多音字)
In addition to the problem of heterophones, there exists also the problem that certain characters can have different pronunciations and meanings depending on their contexts (homographs). This phenomenon is called 多音字 (duōyīnzì) and can be seen in Table 3 below.

<table>
<thead>
<tr>
<th>Pinyin (no tone)</th>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>wo jian ta le</td>
<td>我见他了</td>
<td>I saw him</td>
</tr>
<tr>
<td>wo shou bu liao</td>
<td>我受不了</td>
<td>I can’t stand it</td>
</tr>
<tr>
<td>yin hang</td>
<td>银行</td>
<td>bank</td>
</tr>
<tr>
<td>zi xing che</td>
<td>自行车</td>
<td>bicycle</td>
</tr>
</tbody>
</table>

Table 3: Examples of homographs in Chinese

5. Goals
Given these difficulties with Pinyin, we decided that our goal for this project would be to implement a system that, given a text composed of unsegmented pinyin, would return the proper Chinese character conversion of that Pinyin. Examples of input and output are shown in Table 4.

<table>
<thead>
<tr>
<th>Pinyin (no tone)</th>
<th>Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>wo hen ai ta de shu</td>
<td>我很爱她的书</td>
</tr>
<tr>
<td>ta zai kan yi ben shu</td>
<td>她在看一本书</td>
</tr>
</tbody>
</table>

Table 4: Desired outputs for given inputs

As a design decision, we chose to work with Simplified Chinese, although we could easily adapt our program to account for Traditional by using training data written in Traditional Chinese.

5. Possible Methods
Before beginning with our implementation, we brainstormed several different methods that could be used to implement this program. Among the factors we took into account were efficiency (both for the coders and the users), speed of performance, and compatibility with the problem we were trying to solve.

5.1 Noisy Channel Model
Our first idea was to use a noisy channel model for our implementation, which would mainly involve adapting the transducer system implemented for text disabbreviation. For such method, we could treat Pinyin as the abbreviated text and The Pinyin and characters combined as the disabbreviated corpus. Although that sounded like a good strategy, we then read about Hidden Markov Models, which literature mentioned to be more precise for the problem.

5.2 Hidden Markov Models
After reading the article “A Segment-based Hidden Markov Model for Real-Setting Pinyin-to-Chinese Conversion”, we got convinced that a Hidden Markov Model (HMM) would fit the goal of our project perfectly, and could achieve considerably high character accuracy (82.9% according to the paper).

An HMM is composed of hidden states which emit a sequence of observations over time according to a transition model, as can be seen in Figure 1, which represents a very simplistic representation of a Hidden Markov Model with hidden states (in red), observations (in green), emission probabilities, and a transition model.

That is, in an HMM for the problem we would be able to see a sequence of observations (Pinyin), but we would not know the hidden states (characters) from which the obser-
vations were generated. We would know, however, the probabilities with which the possible hidden state generates an observation (emission model) and the probabilities that a given character comes after another (transition model).

Figure 1: Example of an HMM for the problem

This aligns very well with our goals regarding the project. Given that a certain Pinyin could come from multiple different characters (generally from four to hundreds of different possibilities), we just need to decipher which character is most likely to be the best fit based on the context of the sentence.

In Figure 1, for instance, given the Pinyin “ni”, we would first try and figure out which character is most likely to be the correct one (there are actually far more than two possibilities, but we have only shown two here for simplicity’s sake: 你, which means “you”, and 泥, which means “dirt”). We would do the same for “hao”, with the two characters here being 好, which means “good”, and 号, which means “number”. That method would take in account both the probability of the character having that Pinyin, and the probability of having a character (hidden state) given the last guessed states.

In the end, we would of course want the pairing你好, which is the common way to say “hello” in Chinese. We actually ended up choosing this method to implement, and will explain the details of it in section 6.

5.3. Segmental HMM

Another option we considered, but ultimately rejected, was the Segmented Hidden Markov Model (SHMM). The SHMM is very similar to the HMM, but does make another distinction that the HMM does not: it differentiates between bigrams formed by two words, and Chinese words themselves. Thus, when we search the lexicon for a given segment, if the segment is in the lexicon, then we would treat it as the product of bigrams (two-character segments); otherwise, we would give it as a whole some probability according to a different method.

This type of method would be extremely helpful in the case of idioms, in which we would not have the problem of overestimating bigrams – that is, if a bigram is part of a longer phrase (i.e., an idiom) the count for that bigram as a part of the word is removed, and it is easier to discern the larger word phrase composed of smaller word phrases rather than the two separate word phrases. This means that in general SHMM is better at handling word segments that are composed of more than two characters.

Ultimately, however, we decided that the complexity of this type of model, as well as our scope for the project did not fit the SHMM, and we decided to use the HMM instead, at the expense of some accuracy percentages, although literature reports very similar performance rates for both methods. Literature also reports that SHMMs have much slower execution time, so for this project’s sake, we decided to eliminate such possibility.

6. Implementation

Our implementation is a Hidden Markov Model that uses Viterbi’s Algorithm to return the likely sequence of characters given a sequence of Pinyin. Our project was coded entirely in Python. Viterbi’s algorithm is used to find the most likely sequence of hidden states given a sequence of observations and emission and tran-
sition models. That is, we want to find

\[ x_1, \ldots, x_T = \arg \max_{x_1, \ldots, x_T} P^* \]

Such that \( P^* \) is:

\[ p(X_1 = x_1, \ldots, X_T = x_T | O_1 = o_1, \ldots, O_T = o_T) \]

The algorithm requires several components in order to return the sequence of characters:

i) \( O \): the list of all possible Pinyin
ii) \( S \): the state space, or list of all possible characters (Hanzi)
iii) \( p_i \): the list of initial probabilities, i.e. the probability that a word will be the initial character in a line
iv) \( Y \): the Pinyin text
v) \( A \): the transition matrix (i.e., the bigrams count)
vi) \( B \): the emission matrix (i.e., the probabilities of observing a certain output state from an input state)

Viterbi’s Algorithm is a dynamic programming algorithm, and essentially works by creating two tables (i.e., two-dimensional arrays), \( T_1 \) and \( T_2 \), that record the most likely sequences so far. \( T_1[i,j] \) keeps track of the probability of the most likely path so far that includes the hidden state \( i \) and covers the \( j \) first pinyin in \( Y \). \( T_2[i,j] \) keeps track of the actual states that compose that path.

We can see here that an HMM is a very efficient way of converting Pinyin to Hanzi, considering that the Hanzi are the hidden states in this model, and the Pinyin are our “observations”, or observed states. We wrote functions to calculate all the needed pieces of data, very similarly to the way we implemented the calculation of bigrams for the TANGO algorithm.

The data we primarily worked with was a Simplified Chinese translation of the first chapter of Lewis Caroll’s Alice in Wonderland, 爱丽丝梦游仙境. We first worked to remove all punctuation from the text (for our own ease), and then calculated the rest of the variables. After this, we implemented Viterbi’s algorithm and successfully got Chinese characters from Pinyin input. As a note, we used 1-Laplace smoothing for the case of unseen bigrams.

7. Problems Encountered

Needless to say, we encountered many hardships along the way when implementing the Hidden Markov Model, most of them with regards to the fact that Chinese characters are not ASCII characters and are thus more difficult to deal with when coding.

One of the main problems encountered was dealing with reading in Chinese characters to begin with. We had to do a little bit of research in reading in Unicode characters, which ended up being more difficult than initially imagined. In addition, we found that we needed an easy way to read in characters and their Pinyin without needing to have both a Chinese character text and its corresponding Pinyin text.

We used the cjk and unidecode libraries to translate Chinese characters to Pinyin (the reverse function of our project), and generate training and test data from large pieces of Chinese text. We noticed along the project, however, that both libraries incorrectly translate a variety of characters, which made our code’s precision drop. For instance, the character 么 is translated by both libraries to “yao” instead of “me”.

8. Results and Discussion

When we started testing our program, we were happy to see its good accuracy, specially when we tested sentences with context related to Alice in Wonderland. We also noticed, however, that sentences not related to the training data didn’t do as well, which makes sense considering that we’re using a model based on bigrams-frequencies.
8.1 Validation Set Performance

We left part of Alice in Wonderland off the training data in order to use it as a validation set. The precision we got for that data was 71.3%, which we considered to be particularly good, and very close to the 82.9% mentioned in literature. The text below shows the performance of the algorithm in the first sentence of the book:

爱丽丝靠着姐姐坐在河岸边很久了游于没游什么情可坐她开什感到厌倦她易次游次第瞧瞧姐姐正在读的那本书可书丽没游兔画也没游对画爱丽丝想么书丽本书丽没游兔画河对画那还游什么易丝呢

8.2 Training Set Performance

Since performance is highly biased on context, we decided to test sentences using different training sets and comparing results. Examples of such tests can be seen below, alice_tokenized representing the training data for Alice in Wonderland and catcher_tokenized for The Catcher in the Rye. Examples marked as “A” were tested with alice_tokenized and “C” with catcher_tokenized.

1. wo ai ni
   1.1 我爱你 for C, with 0.67 precision
   1.2 我爱呢 for A, with 0.66 precision

2. wo men ke yi chu qu chi fan
   2.1 我们可以出去离烦 for C with 0.75 precision
   2.2 我们可易锥趣迟烦 for A with 0.375 precision

3. ni gen shui qu wan
   3.1 你根谁去完 for C with 0.6 precision
   3.2 你跟睡趣味碗 for A with 0.4 precision

4. wo diu diao le wo de shou biao
   4.1 我丢掉乐我的首婊 for C with 0.625 precision
   4.2 我丢掉乐我的首表 for A with 0.625 precision

8.3 Analysis

We were very happy with the results we got for the project. We also noticed that the length of our training data was probably too short also, which explains why The Catcher in the Rye (in which we used the whole book) has better precision than Alice in Wonderland for many of the tests. Notice also the bias of the training data in the tests: “ai” is correctly translated to 爱 in Alice in Wonderland, while it is incorrectly translated to 埃 in The Catcher in the Rye. That makes sense, since the character 爱 appears everywhere in Lewis Carroll’s book, since it’s the first character of Alice’s Chinese name, but it probably appears very rarely in The Catcher in the Rye, which makes us think that the latter does not mention love (爱) a lot.

Conclusion

Ultimately, we discerned that the problem at hand was certainly more complicated than we had anticipated. The data we acquired showed a strong bias towards words from the training data, as we had expected, shown by the fact that certain words were better predicted when using Salinger’s novel as training data as opposed to Carroll’s novel. While we were not entirely pleased with our rather low accuracy rate, we recognized that there are an incredible amount of nuances associated with Pinyin and Hanzi, and that it would be fairly impossible to reach a perfect translation, given the diversity of Hanzi characters associated with each separate Pinyin syllable, and the complexity of the Chinese language itself.

We have considered extending our implementation or optimizing it for future use, as this problem is very applicable to many different areas of linguistics. Among these are the possible ways to extend our project:

1. Accounting for tone marks with input (probably for better precision as well)
2. Accounting for unsegmented Pinyin (thus probably using a separate algorithm to segment
the Pinyin, and then trying to convert it to characters)
3. Implementing Hanzi-to-Pinyin translation (not too difficult)
4. Accounting for idioms and longer phrases (likely with the use of an SHMM)

While we are interested in extending this project further, we are also pleased with the amount we have learned about computational linguistics through the scope of this project, especially in dealing with foreign characters and Unicode. We hope that our project has shed light on the complexities of Pinyin-to-Hanzi translation, and that it inspires some questions on further research in the field.