CS 221 Project Progress Report
Comparative linguistic reconstruction with machine learning

Andrew Yang ycm, Ken Hong kenhong, Nash Luxsuwong nashlux

Summary

Our project aims to apply machine learning to comparative linguistic reconstruction. Many East Asian languages today have standardized pronunciations for Chinese characters. The character 中 ('middle') is pronounced /ʈʂɔŋ/ in Mandarin, /t͡sʊŋ/ in Cantonese, /ʨuŋ/ in Korean, etc. Using the known pronunciation of Chinese characters in modern Sinitic and Sino-Xenic languages, historical linguists have created reconstructions ancestors of modern Chinese languages thought to exist a long time ago.

However thorough a reconstruction scheme may be, it is inevitable that a historical linguist omits certain obscure characters from their system—after all, there are tens of thousands of Chinese characters, many of which were created after the ancestral language in question became extinct. By applying machine learning to linguistic reconstruction, we hope to teach a model the relationships between modern and archaic pronunciations so that we can extend the work that human linguists have done to Chinese characters outside the common scope of historical linguistic research.

For our project, we focus on reconstructions of Middle Chinese, an archaic prestige variety of Chinese spoken over a thousand years ago. We parsed Wiktionary to gather the pronunciations of over 15,000 Chinese characters in East Asian languages as diverse as Mandarin, Japanese, and Korean, and aligned them to reconstructed pronunciations devised by influential historical linguists such as Bernhard Karlgren, Zhengzhang Shangfang, and others (also from Wiktionary). Because individual reconstruction schemes are not directly comparable, we are working only with Karlgren's system at the moment.

Preprocessing

After parsing Wiktionary, we ended up with examples like this:

Entry	**Mandarin**	...	**Japanese**	**Korean**	**Reconst. tone**	**Reconst.**
中	zhōng		ちゅう	중	Level	ʈi̯ˬuŋ

We began by separating these unstructured pronunciation data into a format separated into syllabic units, as is standard in the field of phonology. Each Chinese character is one syllable, so we separated pronunciations into *onset* + *nucleus* + *coda* (initial consonant, vowel, final consonant) for each language we are working with. For example, we would divide the unstructured pronunciation /ʈʂɔŋ/ into /ʈʂ/, /ɔ/, and /ŋ/. (Where applicable, we also include the *tone* of a character.) Aside from phonological tradition, this was motivated by the fact that our dataset is quite small; we believe there is simply not enough data for an end-to-end model. Instead, our current architectures predict onset, nucleus, and coda independently. After this step, our data looked like:

Entry	**Mandarin**	**Mand. Nucleus**	...	**Kor. Onset**	**Kor. Nucleus**	**…**	**Reconst. Coda.**
中	⟨zh⟩	⟨o⟩		⟨ㅈ⟩	⟨ㅜ⟩		/ŋ/

Because these data is still categorical, we performed one-hot encoding of each entry to get a sparse binary matrix with dimensions 15250×472.

Code
Baseline model (part 1): https://github.com/ycm/cs221-proj/blob/master/baseline-model.ipynb
Baseline model (part 2): https://github.com/ycm/cs221-proj/blob/master/model-logreg.ipynb
Current model: https://github.com/ycm/cs221-proj/blob/master/model-one-vs-rest-lr.ipynb

Baselines
We believed that a baseline model for our task should ignore relevant phonological attributes of the data. However, since our end product would hopefully be entire syllables, it makes sense that our baseline model tries to reconstruct syllables as well. This way, we can compare the syllables generated by the baseline to the syllables generated by whatever final model we come up with. So, we constructed two types of baseline models: one that generates entire syllables, and one that wasn't.

First approach (generates actual syllables):
Because we analyze every character as *onset* + *nucleus* + *coda* + *tone*, we would need to generate each one of each element. To do this, we made a simple model that predicts randomly (call it **Model A**) and another model that always predicted the most frequent label (call it **Model B**). Model A consisted of four random predictors—the first one guessed a random onset, the second guessed a random nucleus, etc. Model B also consisted of four predictors, where the first one predicts the most common onset, etc.

Second approach (not guaranteed to generate actual syllables):
Recall that we one-hot encoded each syllabic element, so now we have a lot of columns like `tone_rising`, `tone_level`, `Karlgren_onset_b`, `Karlgren_coda_ŋ`, etc. with values 0 or 1. Clearly, if an example has 1 for `tone_rising` then it would have 0 for `tone_level`, and so on. In this approach (call it **Model C**), we train a logistic regression classifier to predict every label column, which turns out to be over 100 logistic regression classifiers. Each logistic regression is trained on 70% of the data and tested on the remaining 30%, with features being the one-hot encoded modern syllabic elements and labels being the one-hot encoded Karlgren reconstruction syllabic elements. Under this scheme, we are no longer restricting our prediction to exactly one onset, and so on.

Baseline results (Accuracy = raw accuracy):
Model A: Random guessing performed exactly as we had expected:

Accuracy on Tone:	*24.45%*
Accuracy on Onset:	*2.71%*
Accuracy on Nucleus:	*1.60%*
Accuracy on Coda:	*12.73%*

There are four tones in Middle Chinese, and Model A guessed correctly a quarter of the time which makes sense.

Model B: Always guessing the most frequent label is better than guessing randomly, because the distribution of labels in our data is not uniform. For example, there is a plurality of characters with level tone in our data.

Accuracy on Tone:	*45.19%*
Accuracy on Onset:	*8.37%*
Accuracy on Nucleus:	*8.18%*
Accuracy on Coda:	*40.14%*

Model C: By separating the label matrix into individual label vectors, we end up with sparse vectors. Logistic regression classifiers assigned to predict the sparsest labels can simply learn to predict 0 every time and achieve very high accuracy.

Raw accuracy over 106 classifiers:	*0.988*
Mean recall:	*0.470*
Median recall:	*0.506*

Here, raw accuracy doesn't provide any useful insights, because most of that "accuracy" is likely due to classifiers assigned to sparse labels predicting 0 all the time. Notice that the recall is quite abysmal as well, as we're misclassifying half of the positive examples on average.

Current Algorithm:
After creating the baseline, we knew that we eventually needed to use routines that were more involved than simply predicting the most frequent label.

Our current model consists of four classifiers each assigned to one of our four syllabic categories. If category c has n_c labels, then the classifier assigned to c is a One-versus-Rest classifier[1] that fits a binary logistic regression classifier for each of the n_c labels.

As a concrete example, let's take a look at the *tone* category. Middle Chinese has four tones: *level*, *rising*, *departing*, and *checked*. We begin by creating a One-versus-Rest classifier for the *tone* category. This classifier now spawns four binary logistic regression classifiers, one for *level*, one for *rising*, one for *departing*, and one for *checked*. For the classifier assigned to *level*, it would treat all examples with *level* tone as 1, and all examples with another tone as 0. Given an example x, the parent classifier would simply go with whichever one of its daughter classifiers returned the highest probability for x.

Formally, let C_k be the parent One-versus-Rest classifier assigned to category k. Let n_k be the number of labels in category k. (e.g. if $k =$ "tone" then $n_k = 4$. We can also represent the n_k labels in category k as the ordered list $l_{k1}, ..., l_{kn_k}$. Now let C_k spawn n_k daughter classifiers, and assign a daughter classifier to each of the labels $l_{k1}, ..., l_{kn_k}$. Write the daughter classifier assigned to label l_{kj} as $c(l_{kj})$. For each daughter classifier $c(l_{kj})$, we fit to our training data by the following method:

For the mth training example, let $x^{(m)}$ be the features and $y^{(m)}$ be the labels. Because there are n_k such labels, $y^{(m)} \in R^{n_k}$ with each component $y_i{}^{(m)} \in \{0, 1\}$. Define a modified label $y_{*,kj}{}^{(m)} \in R$ such that $y_{*,kj}{}^{(m)} = \delta_{ij} = 1(i = j)$. (Incidentally, this is the Kronecker delta!) By doing this, we reduce the multiclass, one-hot encoded label vector $y^{(m)}$ into a binary scalar $y_{*,kj}{}^{(m)} \in \{0, 1\}$. By definition, this is the One-versus-Rest scheme. Now, we just need to fit $c(l_{kj})$ to all $(x^{(m)}, y_{*,kj}{}^{(m)})$. As defined in the scikit-Learn documentation[2], each logistic regression classifier simply minimizes the following regularized cost function:

$$\max_w \tfrac{1}{2} w^T w + \sum_{m=1}^{M} \log(\exp(-y_{*,kj}{}^{(m)})(x^{(m)T} w + b) + 1),$$

where the summation is over each training example.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html
[2] https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Given a test example with features x', the classifier $c(l_{kj})$ would make the prediction $y_{*,kj} = \hat{h}(c(l_{kj}), x')$. The parent classifier would then pick the $c(l_{kj})$ with the strongest prediction, so the predict label is $\hat{L} = \arg\max_{l_{kj}} h(c(l_{kj}),\ x')$.

The evaluation metric we will use is the raw accuracy over each syllabic category; because our model is guaranteed to make one prediction for each syllabic category, and each syllabic category has multiple labels, we can simply gauge our performance by dividing the number of corrected predicted examples by the total number of examples.

Current results:

Here are the results for each of the four One-versus-Rest classifiers:

Accuracy for Onset: 59.98%
Accuracy for Nucleus: 51.32%
Accuracy for Coda: 91.63%
Accuracy for Tone: 75.58%

From a historical linguistics perspective, it makes sense that codas and tones are more easily predicted than onsets and nuclei. This is because languages like Cantonese have preserved codas and tones very faithfully. Similarly, codas in Korean are often reliable proxies for codas in archaic varieties of Chinese. Impressively, our model predicts the correct nucleus more than half of the time. Considering that there are **sixty-two** possible nuclei in our dataset, this is quite a feat already. Even though vowels tend to be volatile, it seems like our model as learned important correspondences between modern varieties of Chinese and the reconstructed forms.

Next Steps:

We have defined a baseline model as well as explored possibilities for our desired algorithm, including the current implementation of multiple One-vs-Rest classifiers. However, we are not yet satisfied with the accuracy of the current reconstructions; as such, we plan to take the following plans to improve our results.

1. *Exploration of more sophisticated models.* There are many complex interactions between the different elements that are well-studied, and it is unlikely that a linear model will be able to fully capture these interactions. Thus, we will try using more sophisticated modeling, such as with neural nets, and see whether this will lead to more positive results.
2. *Incorporation of additional data*. Data used for our current model will be expanded upon with the addition of other languages that we deem to be useful for the reconstruction, with candidates being languages such as Vietnamese and Hokkien for their faithfulness to Middle Chinese phonology.
3. *Noise reduction*. We have not determined whether there exist any features in our model that are extraneous or are contributing negatively towards our results. We will take steps to determine whether this is the case.
4. *Hyperparameter tuning and cross-validation*: Whatever models we end up including, we will spend time adjusting the relevant parameters to ensure optimal performance on our data.
5. *Feature engineering:* We are looking into parsing constituent parts of characters as a feature: because Chinese characters are phono-semantic compounds, featurizing the phonetic components might prove very useful in determining the correct *nuclei* which are notoriously difficult to pinpoint. One idea we have at the moment is to parse Wiktionary or other sources for character compositions, and using those constituent characters as a proxy for *phonetic series*.

6. *K-Fold cross validation*: So far we have simply split our data into training and testing. Because we have only been experiment with different models up to now, we didn't feel like K-fold cross validation was necessary to get a rough idea of the performance of various simple models. As we build more complex models, we will incorporate K-fold cross validation to get a more accurate sense of how our models will perform on unseen test data.

Given the implementation of these steps, we are confident that much better results will appear.