Using Machine Learning Techniques to Uncover What Makes Understanding Spoken Chinese Difficult for Non-native Speakers

John Kowalski
Advisor: Geoff Gordon

Abstract

The Pinyin Tutor has been used for the past few years in over thirty classrooms at universities around the world. A large amount of data have been collected from this program on the types of errors students make when trying to spell the pinyin of the Chinese phrase spoken to them. We plan to use this data to help answer the question of what is hard about understanding Chinese. Is it a particular set of consonants, vowels, or tones? Or perhaps do certain difficulties arise in the context in which these sounds are spoken? Since each pinyin phrase can be broken down into features (consonants, vowel sounds, and tones), we can apply machine learning techniques to uncover the most confounding aspects for beginning students of Chinese. We can extend the methods we developed here to create an ML engine that learns on the fly for each student what they find difficult. The items to be presented to the learner can be chosen from a pool based on the predicted probability of being correctly answered by the student. This will allow the Chinese learner to focus on what he or she is having most difficulty and hopefully more quickly understand spoken Chinese than without such focused "intelligent" instruction.

1. Introduction

One of the hurdles for students in introductory Chinese courses is understanding what Chinese word or phrase has been spoken to them. On top of learning new vocabulary, translation, and grammar skills, simply answering, "What did you hear?" is often difficult for beginning students. Improving this rudimentary skill for each student in the classroom can be tedious and in general a poor use of class time. Luckily, computer tutors are perfectly suited for such instruction. At the Pittsburgh Science of Learning Center (PSLC), the Pinyin Tutor was developed under the direction of Dr. Brian MacWhinney for this purpose *(figure 1)*. Pinyin is a system of writing Standard Mandarin using Roman characters and is commonly taught to first-semester students of Chinese in the first year of instruction. The Pinyin Tutor's instructional model is quite simple: a Chinese word or phrase is "spoken" through the students' personal computer speaker and the task is to enter the pinyin of the sound they heard. If correct, the tutor congratulates the student and presents the next item in the lesson. If incorrect, the tutor gives feedback on what part of the phrase is incorrect and gives the student an opportunity to try again. The items the student answered incorrectly on the first try are put back into the pool to be presented again; items answered correctly on the first try are eliminated from the pool. The student assignment is to continue until all items in the lesson have been answered correctly on the first try (without feedback from the tutor).

2. Pinyin Basics

Each syllable in Pinyin can be thought of as being comprised of three components: an "initial" consonant sound (b, n, zh, …), a "final" vowel sound (ai, ing, uang, …), and a tone marking (1,2,3,4,5) indicating the rising and/or falling pitch of the syllable. In total there are twenty-three initials, thirty-six finals, and five tones *(figure 2)*.

3. Identifying What Causes Students the Most Difficulty

The current version of the Pinyin Tutor keeps track of what Chinese words or phrases are answered incorrectly by students and re-drills the word or phrase until the student can identify it without help from the tutor. Beyond knowing what words or phrases students are having trouble with, the obvious next question to ask is what component(s) of these items are causing the most difficulty. Are there some initials, finals, or tones particularly problematic? And then beyond asking this question for individual components, we must consider difficulties that may arise in the context in which these sounds are spoken. For instance, is final “ao” easy to hear in “hao3”, but not in “bao4”?

To answer these questions, we can train a statistical model to measure student performance. If we can obtain a reasonably low generalization error with this model, we can use the parameters learned to discover what the easy and difficult components are.

3.1. Modeling a Pinyin Syllable

As a first try, we can model a syllable by constructing a linear prediction with 64 covariates (one for each 23 initials, 36 finals, 5 tones), and output whether this item was answered correctly. So we have:

$$\hat{y} = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \ldots + \beta_{64} * x_{64}$$

Where:
y=1 (correct), y=0 (incorrect).
x_n=1 if feature n is present, 0 otherwise.
β_n=coefficient for feature n to be learned.

So once this model is trained and we have learned a value for each β, we can use the model to help us predict whether a syllable will be answered correctly by setting three covariates (x_n’s) to 1 corresponding to the initial, final, and tone present in that syllable, and the rest of the 61 covariates to 0. The output $\hat{y}$ will then give us a clue as to whether this syllable will likely be answered correctly. For instance, for $\hat{y}$ in the range [0,1], we can consider values above .5 likely correct, and values below .5 likely incorrect.

We can supplement this single-syllable model with interaction terms, adding a term for each interaction of initial-final, initial-tone, and final-tone. This will enable us to answer for instance, whether some features are typically easy by themselves, but when combined are difficult. We can also add terms to indicate the number of times a student was exposed to these features and contexts, enabling us to potentially discover some feature to be initially difficult, but fast to learn once presented.

Another way to define a model of a syllable is by using the logistic "squashing" function and performing logistic regression analysis to train it. When using this technique, instead of fitting directly to a linear equation, we fit our data to a logistic curve:

$$\hat{y} = f(\beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \ldots + \beta_{64} * x_{64})$$

Where:

$y=1$ (correct), $y=0$ (incorrect).

$x_n=1$ if feature n is present, 0 otherwise.

β_n=coefficient for feature n to be learned.

f() = logistic function

The logistic function is defined:

$f(z) = e^z / (e^z + 1)$, where:

$z = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \ldots + \beta_{64} * x_{64}$

The logistic function f() will take any value from negative infinity to positive infinity and convert it to values between values of 0 and 1. Logistic analysis is a natural choice for fitting a model to our categorical data where correct=1 and incorrect=0.

3.2. Training the Models

Having constructed representations of student ability at identifying Chinese syllables, we now want to train these models and arrive at the coefficients to best predict student correctness. The method we will first use is the Least-Angle Regression (LARS), modified to simulate the lasso (**l**east **a**bsolute **s**hrinkage and **s**election **o**perator). Lars and the lasso are regression algorithms especially suited for high-dimensional data (Efron, et al, 2004).

Using data we collected from the Pinyin Tutor in Fall 2008, we have approximately 250,000 training examples (tuples of Chinese syllable and binary indicator of whether student correctly identified it). The lasso takes this input and fits our linear model described above using the criterion:

$$\text{Minimize}(\Sigma(y - \hat{y})^2), \text{ under constraint that:}$$
$$\Sigma_j (|\beta_j|) <= s$$

Where:

- y is the actual indicator of correctness from the training data (0=incorrect, 1=correct)
- $\hat{y}$ is the running predicted output of our model computed by the lasso. We assume if $\hat{y} > .5$, then predict correct, $\hat{y} <= .5$, then predict incorrect

- s is a bound that can be used as a tuning parameter.

What makes the lasso special is the "s" parameter. If it is sufficiently large, the lasso is basically the ordinary multiple linear least squares regression of y on the x_n covariates. But for smaller positive values, lasso computes a "shrunken" version of the usual least squares estimate. This shrunken version will often lead to some of the coefficients (β's) being zero, so choosing a value for s is like choosing the number of predictors in the model (Tibshirani, 1996). For a graph of LARS-Lasso coefficient learning progress for our 64-covariate pinyin model, see *(figure 3)*. To see how well the model predicts as we increase the value of the s parameter, see the graph of the K-fold cross-validated mean squared prediction error as the model is learned by LARS-Lasso *(figure 4)*.

Just as we have used LARS-Lasso to efficiently compute "shrunken" least squares estimates, we use L1-regularized logistic regression for its feature selection properties (Koh, 2007). This will give us the flexibility similar to the "s" parameter in lasso. We use the method developed by Koh, et al. for its ability to scale well to large sparse problems (like for our 2374 covariate model for all interaction terms of the basic 64 skills).

Formally, L1-regularized logistic regression learns a weight (coefficient) vector w^T and intercept v under the constraint that we:

$$\text{Minimize } (1/m) \sum_{i=1}^{m} f(w^T a_i + v b_i) + \lambda \|w\|_1$$

Where:

- m is the number of training examples
- b_i is the binary output of training example i
- w^T is the weight (coefficient) vector
- a_i is the vector of covariates for training example i
- v is the intercept
- $\|w\|_1$ denotes the L1-norm (sum of absolute values of all coefficients)
- $\lambda > 0$ is the regularization parameter (conceptually similar to the "s" parameter in Lars-Lasso)
- f() = logistic loss function, defined as $f(z) = \log(1+e^{-z})$

The results we've obtained running L1-logistic regression on the basic syllable model are similar to what we found with Lars-Lasso. We will keep this analysis technique and the efficient method to solve it in mind as we grow to more expressive models that are more computationally intensive to train.

3.3. Interpreting the Trained Models

Besides allowing us to predict whether a student will correctly answer a syllable, perhaps we can glean from these models some insight as to what features are causing the most difficulty. Since larger coefficients of each skill covariate will push the predicted output $\hat{y}$ closer to 1 (correct), and smaller coefficients will push $\hat{y}$ closer to 0 (incorrect), we can interpret these numbers as ranking skills with larger coefficients as easier and those with lower coefficients as more difficult *(figure 5).*

While this interpretation may be true within the mathematical scope of this model, ranking skills as such isn't as accurate a representation of reality as we can get. For one, tones fundamentally influence the way final vowel sounds are pronounced. The pinyin final "ia" is not pronounceable unless we know the tone marking. So to say some final ranks as more difficult than a tone doesn't make much sense. However, as we construct larger models and include interaction terms for each final-tone combination (and other terms as guided by language experts), we will move ever closer to a model representing the true nature of this task.

4. Making the Pinyin Tutor More Intelligent

While the practice the Pinyin Tutor gives students is valuable (Zhang, 2008), it may not be the most efficient way for students to learn since remediation is based on the whole item, not on the parts of the item they have shown to have difficulty. For example, if the tutor presents the two-syllable item "ni3hao3", but the student incorrectly types "ni3ho4", the tutor will put this item back into the pool for a future re-drilling. But the student did not demonstrate difficulty with initials "n" or "h", final "i", or the tone in the first syllable. The student did, however, demonstrate difficulty with the final "ao". Specifically, the student demonstrated difficulty with the final "ao", when in the second syllable, preceded by initial "h", with tone 3. What if we could give students more practice on the parts they have demonstrated difficulty? For our example, what if we could give more practice on items most similar to having a final "ao" in the second syllable, preceded by initial "h", with tone 3? We can in fact augment the Pinyin Tutor so that we can have such "intelligent" behavior by training a hidden Markov model for each possible feature and calculating the probability for each item in the lesson of being in the "Unlearned" state. This technique of estimating the probability a student knows a skill after observing their attempts is known as "knowledge tracing" in the intelligent tutor community (Corbett & Anderson, 1995).

4.1. Hidden Markov Model Basics

A hidden Markov model (HMM) is a statistical model that represents a Markov process with "hidden" state. A Markov process model can be thought of as a finite state machine where transitions between states are assigned probabilities and the probability of transition to state B at time t+1 depends only on the state the machine is in at time t. The "hidden" part is that we are not able to directly observe what state the machine is in currently, just the output dependent on the current state. The output symbols per state are assigned a probability distribution.

4.2. Hidden Markov Model Application

We can create an HMM for each skill necessary to master correctly spelling the pinyin of a spoken Chinese phrase. The skill set can be the 64 basic features. We can also supplement the basic 64 skill models with models for interactions between each, so a skill model for each interaction between initial-final, initial-tone, and final-tone (totaling 64 + 23*36 + 23*5 + 36*5 = 1187 HMMs (skill models)).

The HMM for each skill has two hidden states: "Learned" (L) and "Unlearned" (U). In each of these hidden states, we are able to observe whether a skill was answered correctly (C) or incorrectly (I). The transition probabilities between each state are:

- `P(learn)`, the prior probability of transitioning (U)->(L) at each step given that we start at (U)
- `P(forget)`, the prior probability of transitioning (L)->(U) at each step given that we start at (L)

And conditional observational probabilities within each "hidden" state are:

- `P(slip)`, the probability making an error even if the student is in the (L) state
- `P(guess)`, the probability of guessing correctly even if the student is in the (U) state

See *(figure 7).*

Now that we have the structure (states and transition paths) of the HMMs necessary to represent the skills to tutor, we need to calculate the transition and conditional observational probabilities between the two states. While there is no known way to optimally "train" an HMM (it's an NP-complete problem), fortunately there is a way to approximate transition probabilities via the Baum-Welch algorithm, which makes use of the forward-backward algorithm used frequently in HMM computations (Rabiner, 1989).

Using the data collected from the Pinyin Tutor in the Fall 2008 semester and considering only the student first attempts at each item (and thus not influenced by tutor feedback), we have approximately 250,000 student-tutor interactions for training our pinyin skill HMMs via the Baum-Welch algorithm.

4.3. Using Trained HMMs to Estimate Student Learning State

Now that we have HMMs trained for each skill, we can calculate the probability a skill is in the "Learned" or "Unlearned" state after observing correct/incorrect observations for that skill as the student works through the tutor lesson. Rabiner refers to this as "Problem 2" of HMM applications. Namely, given an observation sequence, O, (of correct/incorrect responses) and an HMM, λ, for the skill, we want to know the probability we are in a state S_i ((U) or (L)) at time t. Formally:

$$\gamma_t(i) = P(q_t=S_i \mid O_1,O_2,\ldots,O_t, , \lambda)$$

Where:

- $\gamma_t(i)$ is probability of being in state i at time t.
- S_i is state i, where i={Learned, Unlearned}
- O_t is the observation (Correct / Incorrect) at time t.
- λ is the trained HMM for a skill

Using the probabilities of being in the "Unlearned" state for each skill, we can now average the probabilities of each feature present in an item being in the "Unlearned" state and base our selection of the next item to present on this calculation. Our aim is that by choosing items with highest average probability of being in the "Unlearned" state, the tutor will tailor its instruction for the student precisely on skills he or she is having the most difficulty. Our aim is for the students to attain a level of mastery much faster than the previous tutor model.

5. Conclusions and Future Work

The analyses and statistical modeling of errors students make on this fundamental task for beginning students of Chinese have never been done at this level before. We hope through our efforts here we can illuminate in great detail what students find most problematic and provide insight for future language research.

Running analyses on the entire data set has been a recurring challenge, as has been running analyses with large numbers of covariates due to computer memory constraints and analysis programs crashing. While in this paper we have only discussed our success at the basic model with 64 skills, we are making inroads to overcome the technological hurdles that make larger analyses difficult. We believe completing these extra-large analyses with ever more descriptive models is achievable in the coming months.

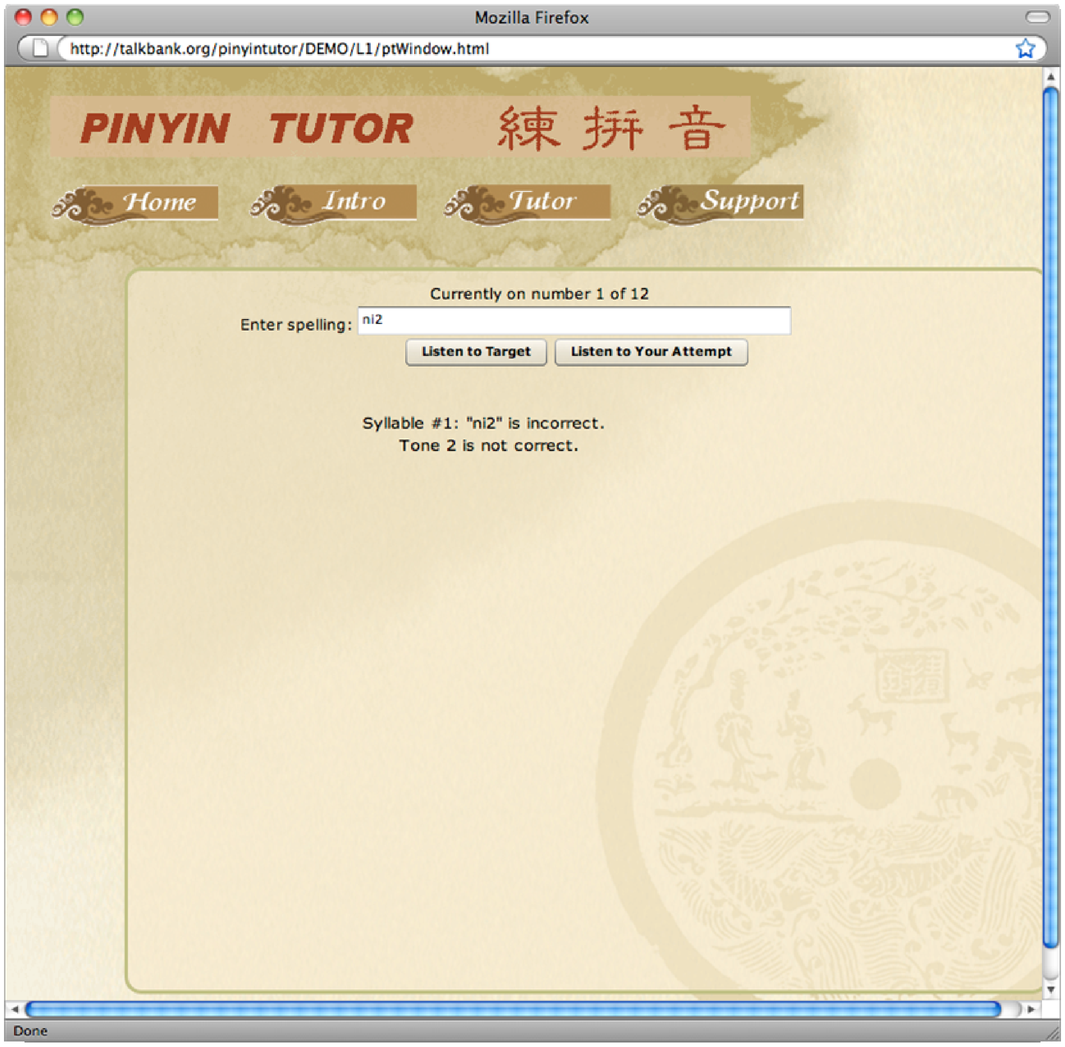
Once the Pinyin Tutor is equipped with the new knowledge tracing module, we expect students will learn more quickly and robustly than with the previous model. Preliminary plans are being made for an in-lab study at the The Chinese University of Hong Kong (CUHK) to test this theory. We expect the full version of the tutor as described in this paper to be online by summer 2010, and used in schools for the Fall 2010 semester. It is currently being advertised on the Pinyin Tutor website and a beta version is currently being tested by at least two sites.

And since the Pinyin Tutor will now be keeping track of the level of each skill for each student, we can provide teachers and students with highly detailed reports of their progress. For the first time, learners will know precisely what sounds and pinyin they are having most trouble with and can pay more attention to them in the classroom and homework assignments.

Another area for future work is to explore if there are other interesting analysis methods available from the machine learning community. For instance, we would like to find precise methods for comparing results of our HMM training with the Lars-Lasso or

L1-regularized logistic results and see if these very different techniques arrive at the same conclusions. Also, we would like to explore methods that are more computationally efficient ways of modeling and training success at identifying pinyin syllables. One such method we're currently looking into is a way for computing the lasso using coordinate descent (Friedman, et al 2010).

Figure 1.
Pinyin Tutor screenshot



Currently on number 1 of 12

Enter spelling: ni2

Syllable #1: "ni2" is incorrect.
Tone 2 is not correct.

Figure 2.
Below are the 64 covariates along with their numeric label we used for the "basic" model (without interaction terms).

Initials:

1	b
2	p
3	m
4	f
5	d
6	t
7	n
8	l
9	g
10	k
11	h
12	j
13	q
14	x
15	z
16	c
17	s
18	r

Finals:

19	zh
20	ch
21	sh
22	w
23	y
24	a
25	e
26	i
27	o
28	u
29	v
30	ai
31	ao
32	an
33	ei
34	en
35	ia
36	ie
37	iu
38	in
39	ou
40	ua
41	uo
42	ui
43	un
44	ve
45	vn
46	ue
47	ang
48	eng
49	ian
50	ing
51	iao
52	ong
53	uai
54	uan
55	van
56	iang
57	iong
58	uang
59	ueng

Tones:

60	Tone 1
61	Tone 2
62	Tone 3
63	Tone 4
64	Tone 5

Figure 3.
The graph below represents the coefficient learning progress of LARS-Lasso on the 64-covariate syllable model. The x-axis represents the fraction of the abscissa values at which coefficients are computed, as a fraction of the saturated |beta|. The y-axis represents the standardized values of the coefficients of each covariate. As described earlier, the Lars-Lasso method can be thought of as a linear regression with an "s" parameter that allows one to choose how many covariates to include in the model. The graph starts at the left with few covariates that are most predictive of whether a student will answer a syllable correctly (corresponding to using a small "s"). As we increase "s" to its maximal value (100% saturation, |beta| / max(|beta|)=1.0), we're including all covariates, which is essentially a regular linear regression. These are the values listed on the rightmost side of the graph.

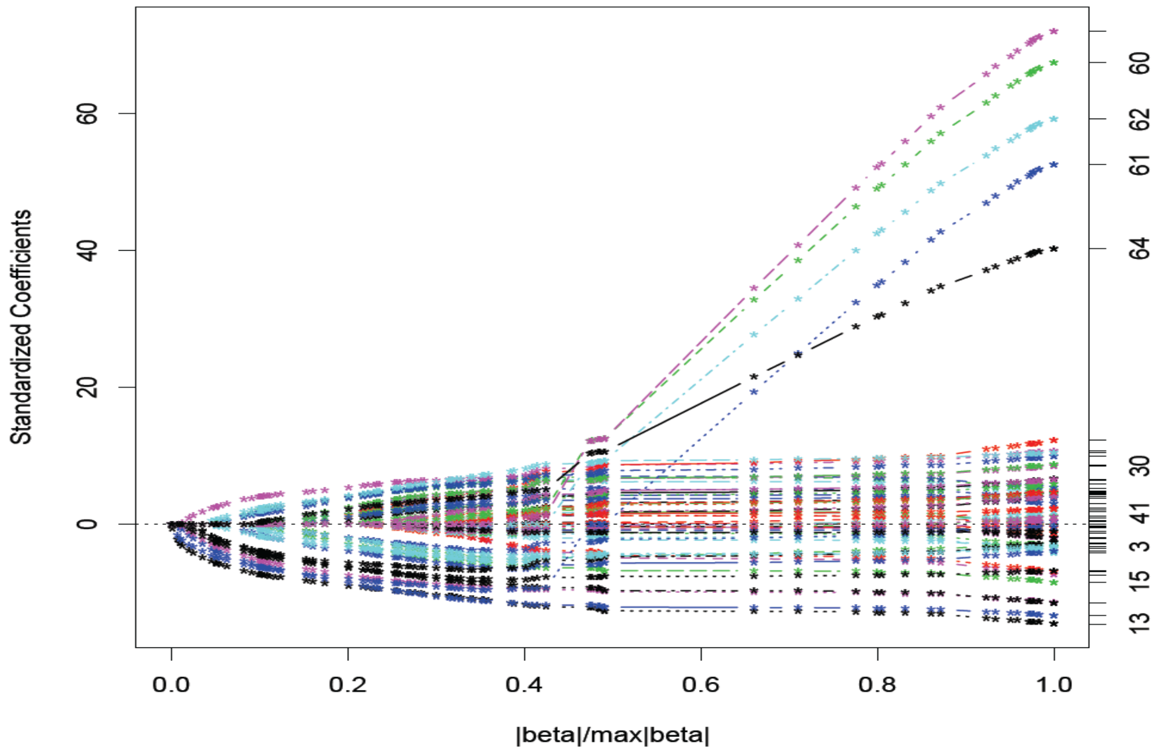


Figure 4.
The graph below represents K-fold cross-validated mean squared prediction error of the 64-covariate syllable model as modeled by LARS-Lasso. As described in figure 3, the x-axis represents the fraction of the abscissa values at which the CV curve should be computed, as a fraction of the saturated |beta|. Here we see that the 64-covariate model maximizes its predictive power after about 1/3 of the most important covariates (in terms of predicting syllable correctness) are included.

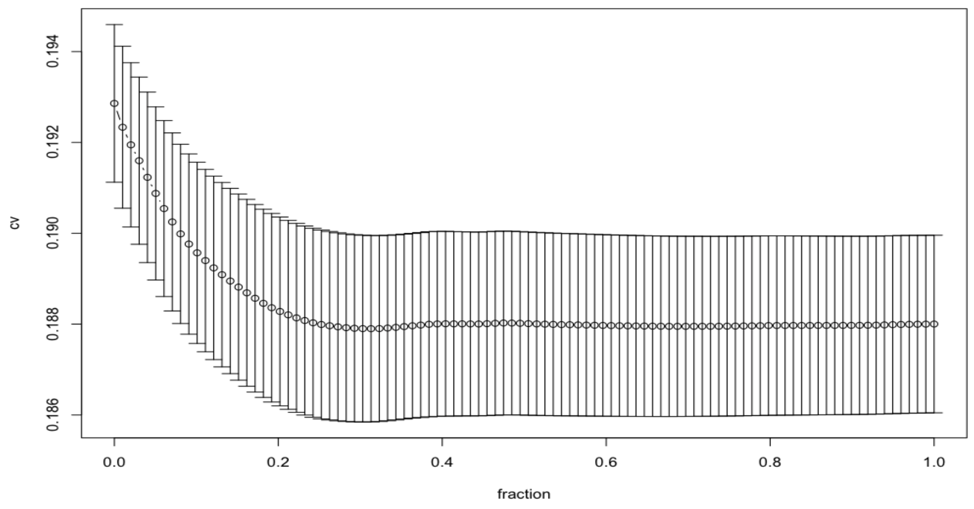
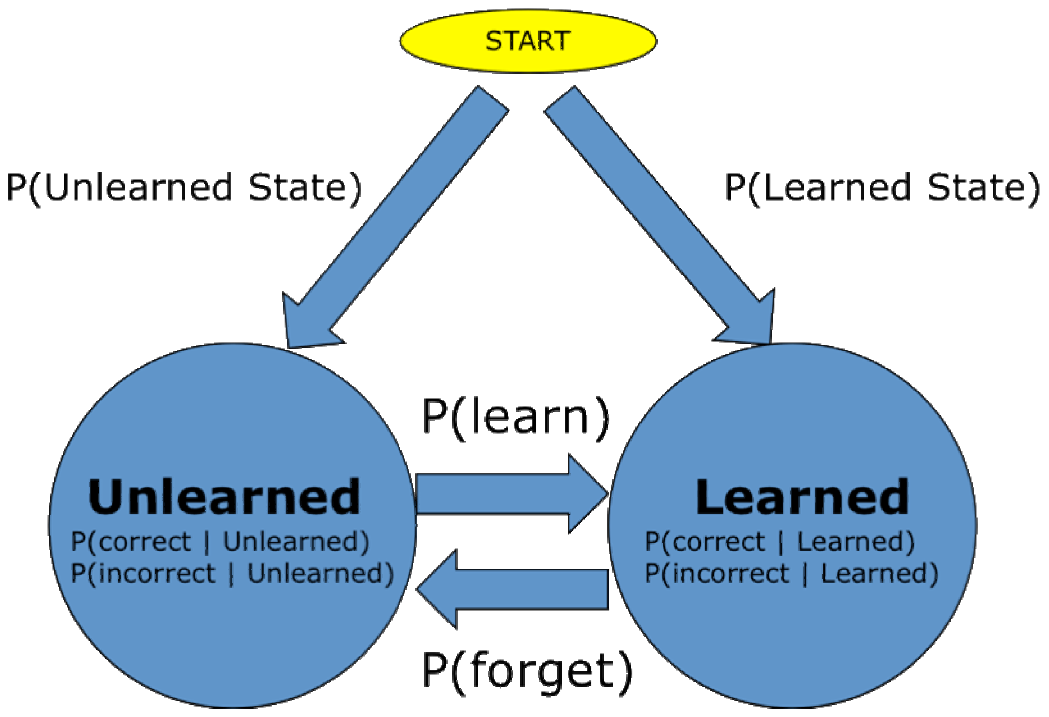


Figure 5.
Below we list the coefficients of the basic 64-covariate syllable model learned by LARS-Lasso on random 10,000 attempts.

-0.227838233	ve (üe)	0.00486158	ui
-0.205062847	c	0.010542571	ou
-0.146153347	un	0.013836567	iang
-0.13763165	q	0.015997153	uo
-0.128128634	r	0.020129841	an
-0.106388461	zh	0.031502608	e
-0.085701215	z	0.036566456	ian
-0.081860869	iong	0.037555292	d
-0.078574248	v	0.038106812	k
-0.055606824	j	0.043877758	g
-0.051050815	x	0.046953773	f
-0.04879625	o	0.053512103	in
-0.04779997	s	0.058413901	w
-0.043854635	uan	0.060323097	iao
-0.041513312	ang	0.060392861	ao
-0.029042265	eng	0.061035341	ue
-0.027708019	m	0.064187286	i
-0.026345493	y	0.068191224	iu
-0.026324417	ing	0.0734974	ei
-0.020049983	t	0.076838215	uang
-0.017968735	n	0.087835728	ong
-0.01500347	b	0.089788545	ie
-0.012971185	uai	0.092804293	ai
-0.012052117	ch	0.096112072	en
-0.002689831	l	0.099807003	a
-0.000855295	sh	0.126843133	ua
0	p	0.202237942	ia
0	vn (ün)	0.243847379	Tone 2
0	van (üan)	0.289695259	Tone 3
0	ueng	0.295659682	Tone 4
0.000740474	u	0.300028026	Tone 1
0.002231802	h	0.333281778	Tone 5

Figure 6.
Hidden Markov Model representation (one for each pinyin skill).



References

[1] Efron, B., Hastie, T., Johnstone, I., Tibshirani, R. (2004), Least Angle Regression, Annals of Statistics, Volume 32, Number 2, 407-499.

[2] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," Proc. of the IEEE, Vol.77, No.2, pp.257-286, 1989.

[3] Tibshirani, R., (1996), Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society, Series B (Methodological), Volume 58, Issue 1, 267-288.

[4] Zhang, Y. (2008), Cue Focusing for Robust Phonological Perception in Chinese.

[5] Corbett, A. and J. Anderson, Knowledge tracing: Modeling the acquisition of procedural knowledge., 1995.

[6] Casella, George, and Berger, Roger L. (2002). Statistical Inference, Second Edition. Duxbury Press, Pacific Grove, California.

[7] Hastie, Tibshirani, and J. H. Friedman. Elements of Statistical Learning New York: Springer, 2009

[8] Koh, K., Kim, S., Boyd S. (2007) An Interior-Point Method for Large-Scale L1-Regularized Logistic Regression, Journal of Machine Learning Research Number 8, 1519-1555.)

[9] Cen, H., Generalized Learning Factors Analysis: Improving Cognitive Models with Machine Learning.

[10] Friedman, J., Hastie, T., Tibshirani, R. (2010) Regularization Paths for Generalized Linear Models via Coordinate Descent.

[11] Hastie T, Efron B (2007). lars: *Least Angle Regression, Lasso and Forward Stagewise.* R package version 0.9-7, URL http://CRAN.R-project.org/package=Matrix.

[12] Koh K, Kim SJ, Boyd S (2007b). l1logreg: *A Solver for L1-Regularized Logistic Regression.* R package version 0.1-1. Avaliable from Kwangmoo Koh (deneb1@stanford.edu).