Chinese Character Recognition: Naïve Bayes Net vs Hidden Markov Model

Meg Campbell
CSE 573

Abstract

Chinese and Japanese character recognition remains a subject of ongoing study due to the size and complexity of the problemspace. Focusing on simple models of characters and features, I compared the performance of a Naïve Bayes Net classifier to a Hidden Markov classifier I had written for a previous AI course. My intent with the Bayes Net was to produce a classifier that was more tolerant to an input with incorrect stroke order. While the Markov model performed consistently better than the Bayes for character recognition and individual stroke recognition with in-order stroke data, the Bayes recognized characters with incorrect stroke order much more accurately than the Markov.

Introduction

For new students of Japanese and Chinese, the alphabet is one of the most daunting attributes of the language. Unlike English's pseudophonetic alphabet, Chinese uses a logographic alphabet where each character represents a word or concept; Japanese uses three alphabets, two phonetic and one logographic. The amount of memorization inherent in learning either language makes them very difficult for beginners.

These logographic alphabets also present a challenge to a user who is translating unfamiliar text. Because of the size of the alphabet, it's very possible to encounter a character you've never seen before during translation. Dictionary entry is difficult when the character is wholly unfamiliar. The most common approach to unfamiliar character lookup is an electronic "search by radical". The user selects subpieces of the character they want to find and searches for composite characters that meet those criteria. Radical search is very reliable for an intermediate user, but still challenging for beginners who may not yet know how to recognize what sections of a character compose a recognized radical.

Figure 1. A small subpiece of a kanji search by radical, jisho.org

By far the most intuitive means of character lookup is drawing the unfamiliar character and having a dictionary find lookalikes. Recognizing the drawn character from OCR remains a difficult problem, as each character is hard to normalize and segment. Recognizing the drawn character from tablet input is somewhat simpler, as this format is already implicitly segmented and contains temporal information.

Data and Features

Generally speaking, a given character can be described by a few attributes: the number of strokes, their order, and relative positioning. A stroke can be characterized by its length, directionality, and number of changes in directions.

For this project a character was represented by xml input containing a list of points and defined and labeled substrokes.[1] This input provides Cartesian coordinates for each point as well as information about time, pressure, and which points comprise a stroke (from pen down to pen up). The code processes these coordinates to determine three distinctive features.

Feature	Discrete/Continuous	Meaning
Length	Continuous[2]	Sum of the Euclidean distance between each pair of consecutive points in the stroke.
Directionality	Discrete	The overall direction of the stroke as determined between the first and last points. This is grouped into eight directions (cardinal and intermediate) with each classification being determined by the slope and a slight error range.
Number of turns	Discrete	The number of changes observed in the directionality of the stroke, sampling at intervals of every ten points. Range in value from 0 to 7.

Figure 2. Stroke features and attributes

Character Recognition

I tested character recognition using two very minimalistic models. The first, which I implemented four years ago for an undergraduate AI class, used a Hidden Markov Model to describe the characters. The "hidden states" in question were the individual strokes, with transition probabilities derived from the likelihood of seeing a stroke given the preceding stroke. The emissions probabilities were calculated using training data and the features described in Figure 2. The final character recognition was determined by dictionary comparison on the identified strokes.

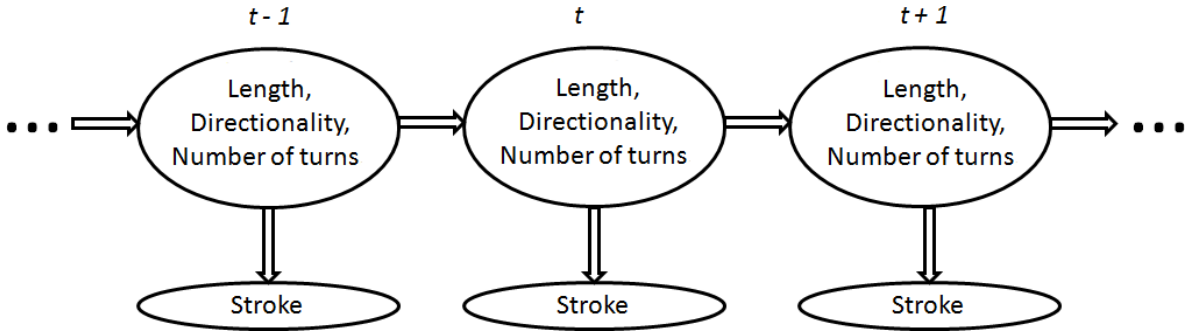

Figure 3. HMM design for character recognizer

This model assumed that the strokes were entered in the correct order and neatly, meaning the user lifted the pen at the end of every stroke before drawing the next. This type of input excludes cursive, messy handwriting, and the less precise behavior of a user looking up an unfamiliar character; often, these users will use the wrong stroke order, fail to end each stroke at the correct point, or break a continuous stroke.

Hoping to build a system that was more robust to wrong stroke order—something the HMM proved particularly bad at, due to biasing from the stroke transition probabilities—I decided to implement a naïve Bayesian classifier to recognize the characters.

[1] For sample xml data see Appendix A or the "Data" folder of the submitted project.
[2] Though inherently continuous, this feature was binned and treated as discrete for the HMM approach.

The Bayes model used the same basic attributes as the HMM classifier. However, where length in the HMM classifier was treated as a discrete value and binned into "short" and "long", the Bayes treated length as a continuous variable and used a Gaussian distribution to characterize the length of each stroke from observable data. Directionality and number of turns were both inherently discrete. To keep the model simple, all variables were assumed to be independent of each other. This may have been an oversimplification; a stroke with a high number of turns is likely to be longer. Similarly, a "long" diagonal stroke will naturally be longer than a "long" vertical or horizontal stroke. Treating the length as a continuous variable was intended to compensate for some of these potential dependencies by providing a more precise measure of the expected length of each given stroke.

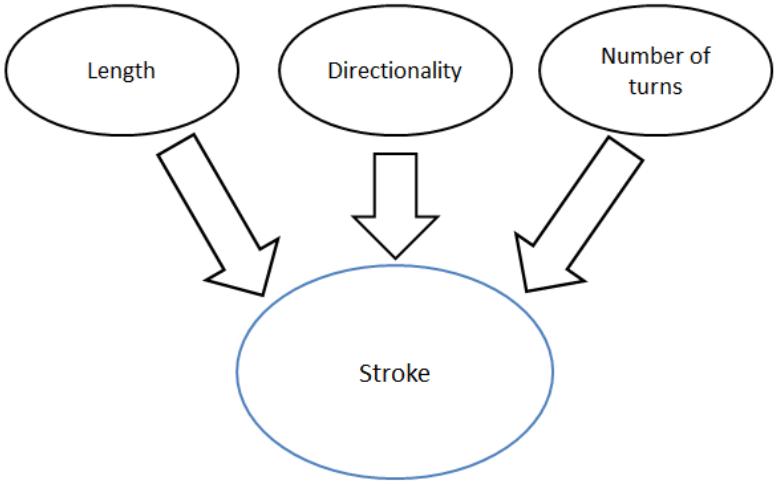

Figure 4. Bayes model for character recognizer

Like the HMM, the Bayes model was trained on fully labeled sample data in order to construct a probability matrix of the likelihood of each trait given the stroke identity. Then the algorithm calculated the maximum likelihood estimate assuming uniform priors. This decision was made in order to support the relaxation of the stroke order requirement; any stroke may occur at any step in the character.

Both classifiers provided the same basic functionality. Given a directory containing training files, they trained the classifier and then identified each stroke in an input character, finally producing an estimate for the total character. Both can output confusion matrices for the individual stroke data. For character classification, they output the name of the file to be classified and the best estimate of the classifier, as shown in Figure 5.

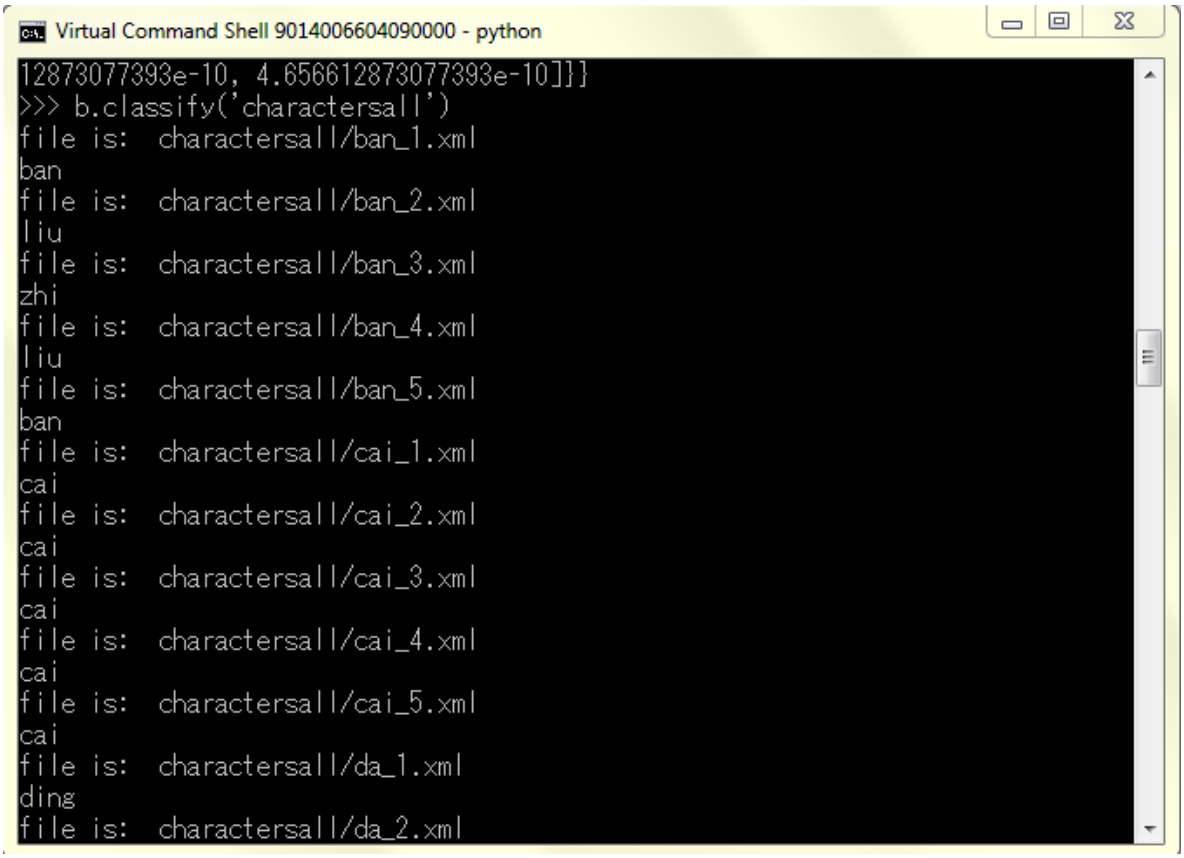

Figure 5. The Bayesian classifier running on a directory of test data

Results

The two classifiers were tested on three sets of data. The first, a simplified training set, broke strokes on reflex curves to produce characters made only of straight lines. This was the dataset used for the presentation. However, while simpler and helpful for debugging, this dataset is not representative of the way any user would naturally write a character.

The final results were instead obtained from datasets written in a more natural manner, without segmenting the strokes. This produces more complex strokes. While the Bayesian classifier performed comparably to the Markov on classifying the segmented strokes, on the unsegmented dataset it was consistently weaker than the Markov (Fig 6). This performance was particularly bad on the complex strokes (see “kohat”, “fishhook”, “Lku”). Because both systems use the same features to recognize strokes, it seems likely that these complex strokes—also the rarest occurrences—benefitted the most from the Markov model’s transition probabilities.

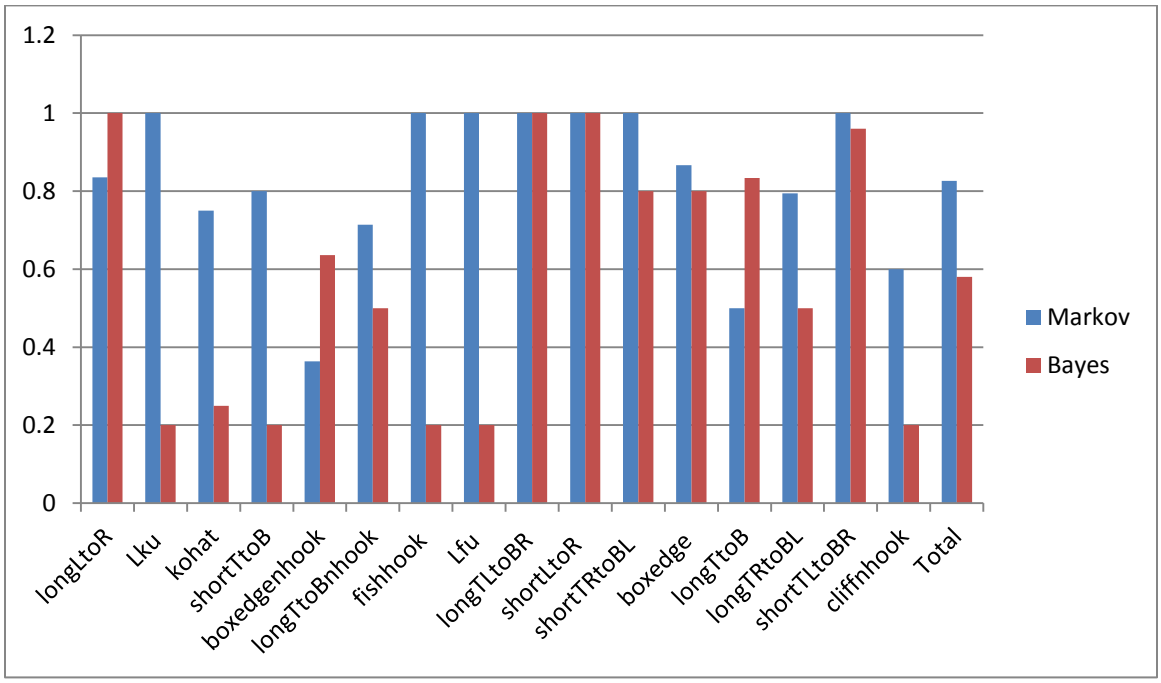

Figure 6. Stroke recognition percentage (in order dataset)

The overall character recognition was slightly better in the Markov model than the Bayes for characters with in-order strokes. However, the character recognition on out of order strokes was very low for the Markov (25%, compared to 66% for in order) but comparable to the in-order performance for the Bayes (55% for in order compared to 65% for out of order).

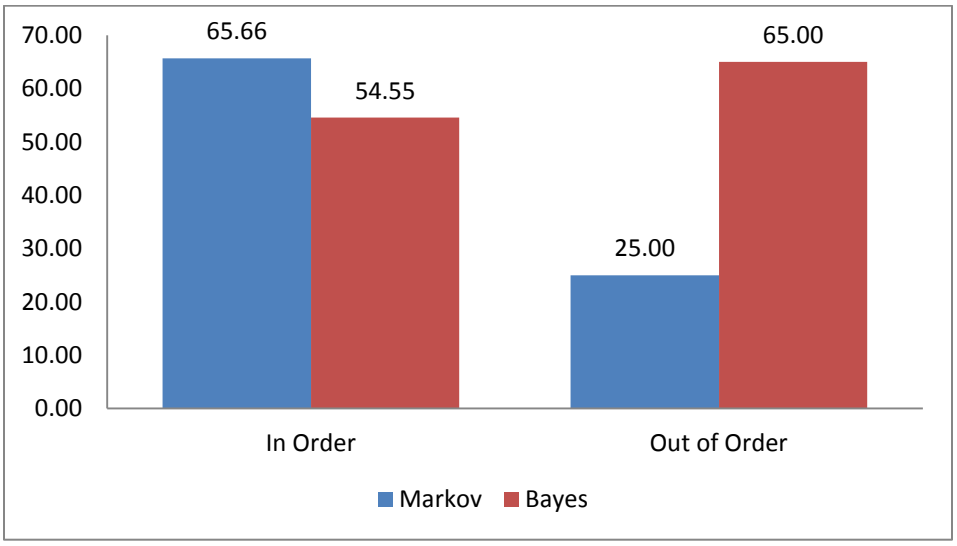

Figure 7. Character recognition percentage

Surprises

When I drafted my initial proposal for the project, I expected to see the Bayesian classifier perform comparably to the HMM and that both would fall in the 80-90% accuracy range. In fact, both were substantially less accurate than that despite my results four years ago reaching above the 90% range. In part this is because the code I inherited from that project was incomplete. The final draft of my original code included a fourth attribute, the "degree of change" in a stroke in addition to the number of directional changes. The degree of change represents the magnitude of a directional change at each switch. For example, a horizontal stroke that turns sharply into a vertical would have a change degree of two; there is one intermediate position (diagonal) and then the vertical. The degree of change varies from one to seven, depending on how sharply reflex a curve is. This additional attribute substantially improved the performance of the old model. While I repaired a large amount of the other missing code, I didn't have time to add in the final attribute, leading to a lower overall performance than I had expected.

It was also surprising that the out of order Bayesian classifier performed *better* than the in order Bayesian classification. Partially I attribute this to the smaller data size of the out of order strokes. Because of the temperamental nature of the sketch panel code (see Appendix B) it was necessary to reorder the strokes for the out of order set by hand, manually changing the timestamps on the substrokes and their labels. Because of the time consuming nature of the job, the out of order dataset was relatively smaller and so the resolution of the results is less meaningful than the in-order set. I suspect that with a larger dataset the Bayes classifier would perform identically on the in-order and the out of order, since the stroke order doesn't factor into its probability model at all. The Markov, however, would still perform substantially worse on the out of order.

Conclusions and future work

As hypothesized originally, a very simple naïve Bayes classifier that disregards stroke order performs much better than the original Markov model for recognizing characters with strokes out of order. It also performs comparably to the Markov model running on in-order strokes. Both, however, have overall low performance. Ideally this work would be extended to factor in more stroke features for greater accuracy, particularly the degree of change attribute that was lost from the old code, and potentially degree of curvature.

Additionally, *neither* model deals with the case of recognizing characters whose strokes are the same but whose stroke positioning creates a different character. For example, both 力 and 刀 possess the same two strokes (by the classifiers' terminology, 'boxedgenhook' and 'longTRtoBL') but should be identified as different characters. This heavily simplified model doesn't take into account the stroke positions or points of intersection, which on a larger character set would cause it to make incorrect identifications.

The paper that I was initially planning to model my work on[3] uses *both* Bayes and Markov to create an extremely robust model of character recognition that can tolerate incorrect stroke order, malformed strokes, not picking up the pen, and other errors while maintaining high (above 80%) accuracy. For future work I would try to draw more from this model, using the strengths of each classifier and potentially comparing each stroke's recognition to take the label of higher certainty at each step. It could potentially be more accurate to classify using the multiradical lookup method (Fig 1); however, this would require identifying complex subsets of the character that contain multiple strokes. These dictionaries are highly accurate but the identification problem becomes correspondingly more complex (and the labeling more difficult) when looking at multistroke compounds that make up a single character.

[3] http://hal.inria.fr/docs/00/10/47/51/PDF/cr1097190992028.pdf

Appendix A: Data Model

```
<?xml version="1.0" encoding="utf-8"?>
<sketch id="ff204993-50a4-4a1e-9eb8-f25ea65f4b5c" units="himetric">
  <point x="824" y="839" pressure="24" time="1241730733603" name="point" id="7debf261-088f-4fab-9a10-
1165ed59713f" />
  <point x="824" y="839" pressure="50" time="1241730733611" name="point" id="b4161414-778b-48f5-8c16-
21aa8d6e702d" />
  <point x="824" y="839" pressure="77" time="1241730733618" name="point" id="3f3d2aa3-9dd9-464c-8289-
f0d9c641b416" />
  <point x="824" y="839" pressure="97" time="1241730733626" name="point" id="40e2c10b-5f91-411d-bcee-
7c589926af75" />
  <point x="824" y="839" pressure="114" time="1241730733633" name="point" id="79d847af-935d-4ae4-8d2e-
3bf6e529d859" />

…

  <shape type="substroke" name="substroke" id="faf10694-1467-44fd-bacc-70cf33c05bb1" time="1241730734505"
x="819" y="667" color="-16777216" height="1485" width="1007" penTip="Ball" penWidth="53" penHeight="1"
raster="CopyPen" start="7debf261-088f-4fab-9a10-1165ed59713f" end="9764cd1a-e432-4694-8b8b-a8658acad1a4"
source="ConverterJnt">
    <arg type="point">7debf261-088f-4fab-9a10-1165ed59713f</arg>
    <arg type="point">b4161414-778b-48f5-8c16-21aa8d6e702d</arg>
    <arg type="point">3f3d2aa3-9dd9-464c-8289-f0d9c641b416</arg>
…
  </shape>

  <shape type="stroke" name="stroke" id="0c64349b-c586-4097-a3da-1c8bcdf8ae80" time="1241730734505" x="819"
y="667" height="1485" width="1007" start="faf10694-1467-44fd-bacc-70cf33c05bb1" end="faf10694-1467-44fd-bacc-
70cf33c05bb1" source="Converter">
    <arg type="substroke">faf10694-1467-44fd-bacc-70cf33c05bb1</arg>
  </shape>

…

  <shape type="boxedgenhook" name="shape" id="d8be2430-2cbc-4c54-80b7-c57686f3bf10" time="1241730734505"
x="819" y="667" probability="0" color="-16777216" height="1485" width="1007" penTip="Ball" raster="CopyPen"
start="faf10694-1467-44fd-bacc-70cf33c05bb1" end="faf10694-1467-44fd-bacc-70cf33c05bb1"
source="Sketch.AddLabel">
    <arg type="substroke">faf10694-1467-44fd-bacc-70cf33c05bb1</arg>
  </shape>
</sketch>
```

Appendix B: Code Readme

- **Data**: contains the three data subsets, xml format. "Segmented data" is the data in which the strokes are broken on reflex curves. "Unsegmented data" contains the characters with unbroken strokes. "Unsegmented wrong order" contains the data with the strokes manually reordered. All data is fully labeled in order to computer confusion matrices and may be used as a training set.
- **SketchPanel and Labeler**: contains the applications for collecting character xml (SketchPanel) and labeling it (Labeler). These applications can be launched from the executables in the Programs/SketchPanel subfolder. This is work from Christine Alvarado's sketch recognition, done in part at Harvey Mudd and in part at MIT. Further documentation is available at: http://www.cs.hmc.edu/~alvarado/research/download.html

  PLEASE NOTE THE FOLLOWING CAVEATS FOR THESE PROGRAMS: SketchPanel will only accept input on a tablet PC running either Windows XP or Windows Vista and will only correctly save xml in XP. The Labeler will behave correctly on either. Neither will run reliably on a non-tablet PC or on a more current OS in compatibility mode.
- **guid**: a supporting file used by the Bayesian classifier to save files with labels. Copyright Conan C. Albrecht.
- **CharacterReco** and **BayesCharacterReco**: the Markov and Bayesian character recognizers, respectively. Usage noted below. Functions take a directory name as input and walk the directory to train the model/classify and evaluate, respectively.

CharacterReco (Markov)

```
>>a = StrokeLabeler()
>>a.trainHMMDir('training')
>>a.classify('characters')
>>a.evaluate('characters')
```

BayesCharacterReco (Bayes)

```
>>b = Bayes('training')
>>b.classify('characters')
>>b.evaluate('characters')
```