Kanji Character Recognition Techniques

Etai Klein
Supervised by: Sravana Reddy

June 9, 2014

Abstract

This project attempts to create an accurate stroke-order-agnostic classification system for Japanese and Chinese characters (*kanji*). This project compares accuracy across 3 different classification methods across 2 data sets and 2 levels of Japanese *kanji* features. This survey is the first half of a project which will result in the creation of an app to help students of Japanese practice their handwriting. The project was coded in Java using a modular design.

1 Introduction

Current *kanji* classification methods inaccurately classify strokes that are out of order or combined. This causes problems for both new students and native speakers. This projet address this by surveying classification techniques to find the most accurate stroke-order agnostic classification system. This is important to improve the accuracy rates of classification.

2 What is Classification?

Classification is process of identifying to which group an unidentified observation belongs. Classification relies on machine learning and statistical algorithms. Classification is used for a wide number of applications including search engines, video tracking, biometric identification, and handwriting recognition.

This project focuses on Optical Character Recognition and Handwriting Recognition of Japanese *kanji*. A simple application of this kind of recognition is writing a *kanji* one might see on the street in a dictionary app to find it's meaning.

3 What is a *kanji* ?

Kanji are Japanese adopted Chinese logographic characters. In the 5th century, Japan imprinted China's writing system (*hanzi*) onto Japanese speech. Today, Japanese *kanji* and Chinese *hanzi* are still the same except for two exceptions: The stroke order for Chinese and Japanese characters are occasionally different, and some characters that have simplified over time have been simplified differently. These have no bearing on this project as each classification method will be trained and tested on data sets from the same language.

One way to classify *kanji* is to look at the sum of their strokes, the building blocks of *kanji*. There are 2,136 *joyo* (daily use) *kanji* as defined by the Japanese Ministry of Education in 2010. *Kanji* are made from groups of *bushu* (Radicals), which are in turn made from ordered groups of strokes. There are 214 total *bushu*. Strokes and radicals can be used help classify *kanji*.

4 Materials and Methods

This section is about the datasets, feature levels, and classification methods used in this survey.

4.1 datasets

This project used two data sets:

4.1.1 CASIA Online and Offline Chinese Handwriting Databases

This is a Chinese handwriting database which can be found at `<http://www.nlpr.ia.ac.cn/databases/handwriting/Home.html>`. The Chinese Handwriting Recognition competitions dataset contains 60 tokens of almost 4000 types of *kanji*.

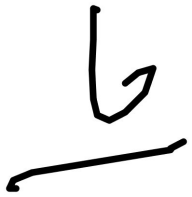

Figure 1: An example of the CASIA database of the *kanji* (ue)

The data was an encoded series of bytes and had to be decoded. Each file contained almost 4000 *kanji* with the values: size (bytes), GBK tagcode, number of strokes, concatenated xycoordinates, and an end marker. The decoding script can be found in `ChineseOnlineHandwritingToJPG.java`.

The *kanji*'s name was tagged using GBK, an encoding standard like unicode which is used in China, and required translation. The translation script can be found in `ChineseCharacterLookup.java`which parses the html from this online table `<http://www.cs.nyu.edu/~yusuke/tools/unicode_to_gb2312_or_gbk_table.html>`

4.1.2 Collected data

I created an app to collect data from students learning Japanese (as per the app's target audience). The app can be found in the project `datacollection`. The basic idea was that the app prompted users to write a character displayed at the top of the screen. When users hit the next button, their data was saved to dictionaries for compressed bitmaps and strokes. Then, when the user hit the save button, the dictionaries were saved to the phone to be collected later.

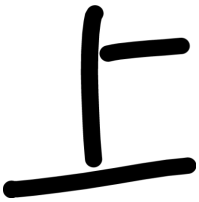

Figure 2: An example from the collected data of the *kanji*

The collected data was pre-processed by scaling each image to the same size.

4.2 Feature levels

Kanji classification essentially is about finding the difference between two or more kanji. One way to computationally classify an unknown *kanji* is to quantify some features about the *kanji* and compare it to other *kanji* to find the most similar one. Humans are notoriously worse at choosing features than machines.

4.2.1 Pixel Level

Optical Character Recognition can classify images using the lightness of pixels as features.

Below I've included my distance method for pixel level analysis. The basic idea is that the difference between the lightness of each pixel of two kanjis (one of which being an average of many kanji) are summed up. That sum is the 'distance' used to classify the kanji.

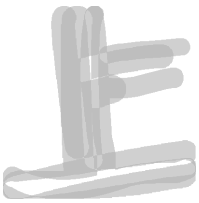

Figure 3: An example of the averaged Collected data for the *kanji*

```
  /**
   * distance
   *
   * @summary Computes a score based
 on how different one \textit{kanji} is from this \textit{kanji}
   *
   * @param other\
textit{kanji}Pixels - The pixels of the \textit{kanji} to Compare to
   * @return sum - the difference score
   *
```

```
   * @pseudocode
   * 1. loop through each \textit{kanji}'s pixel array
   * 2. sum the difference between pixels
   *
   */
  public int distance(int[][] other\textit{kanji}Pixels){

    double sum = 0;

    for (int row = 0; row < averagePixels.length; row++) {
      for (int col = 0; col < averagePixels[0].length; col++) {
        //sum the difference between each pixel
        sum += Math.abs((double) averagePixels
[row][col] - (double)other\textit{kanji}Pixels[row][col]);
      }
    }

    return (int)sum;
  }
```

4.2.2 Stroke Level

Strokes, unlike pixels, don't have obvious features. I used the following features:

- stroke length (start point to end point)
- stroke angle (start point to end point)
- direction between strokes (end point to start point)

The daily use *kanji* all have fewer than 30 strokes, soI use an array of length 30 to represent the features of each stroke.

Below I've included my distance method for stroke level analysis. The basic idea is that the difference between the angle and length for each stroke of two kanjis are summed up along with the movement between each stroke. That sum is the 'distance' used to classify the kanji.

```
/**
   * distance
   *
   * @summary Computes a score based on how different one \textit{kanji} is from this \
     textit{kanji}
   *
   * @param \textit{kanji} - The pixels of the \textit{kanji} to Compare to
   * @return sum - the difference score
   *
   * @pseudocode
   * 1. loop through each \textit{kanji}'s strokes
   * 2. Sum the difference between lengths, angles, and moves
   *
   * @note: moves is weighted by 100 since it is a more useful feature
   *
   */
  public double distance(Stroke\textit{kanji} \textit{kanji}) {

    int maxStrokes = 30;
    int weight = 100;

    int sum = 0;

    int i = 0;
    while (i < maxStrokes){

      sum+= Math.abs(this.avelengths[i] - \textit{kanji}.lengths[i]);
```

```
    sum+= Math.abs(this.aveangles[i] - \textit{kanji}.aveangles[i]);

    //moves are the difference between strokes, so there will be one less maximum move
        than maximum stroke
    if (i < maxStrokes-1){
      sum+= weight * Math.abs(this.avemoves[i] - \textit{kanji}.moves[i]); //weighted
    }

    i++;
  }

  return sum;
}
```

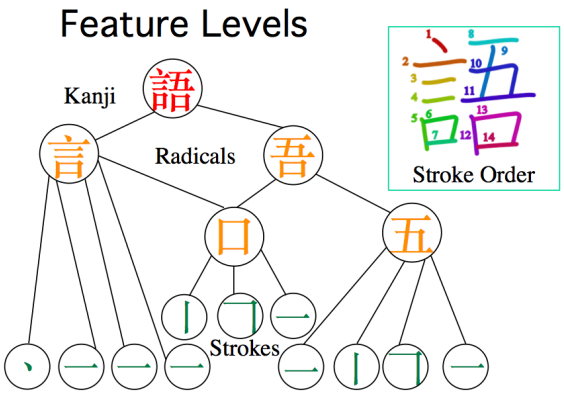

Figure 4: The Feature levels of a *kanji* beyond the pixels level.

4.3 classification Methods

The basic idea for any classification technique is to score all possible classifications based on the test *kanji*'s features and pick the best score. This project used data split into training and testing sets in order to score the testing *kanji* against the trained *kanji*.

4.3.1 K-Nearest-Neighbors

K-Nearest-Neighbors (KNN) works by assigning scores to all *kanji*s in the training datasets. Then, the training dataset kanji are scored and classified as the mode of the K best scores.

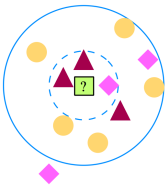

Figure 5: If K is 3 (the smaller circle) then we would classify the unknown shape as a red triangle. If K is 10 (the larger circle) then we would classify the unknown shape as a yellow circle

4.3.2 Gaussian Distribution

The Guassian distribution works by computing the average and the standard deviation of every feature for every *kanji* type. Then, unknown *kanji* are classified as the *kanji* with the lowest sum z-score.

4.3.3 Univariate Distribution

The Guassian distribution works by computing the average of every feature for every *kanji* type to compute a single average *kanji* for each type. Then, each average *kanji* is scored and the unknown *kanji* is labeled as the average *kanji* with the best score.

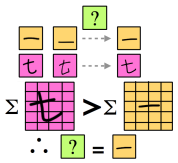

Figure 6: The sum of the pixels z-scores for the unknown shape lower for the yellow box.

4.3.4 WEKA Multivariates

WEKA (Waikato Environment for Knowledge Analysis) is a data analysis and machine learning tool. Multivariate data was calculated using WEKA's java plugin. A multivariate classifier uses complex algorithms involving the relationship between features for classification. Two useful WEKA tests are IBK, a K-Nearest-Neighbors classifier and SMO, a Sequential Minimal Optimization algorithm for training a support vector classifier (which I am choosing to think of as a multivariate distribution classifier).

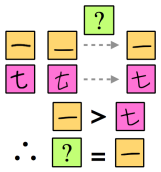

Figure 7: The unknown shape is closer in similarity to the average yellow box on a pixel by pixel level.

5 Code Structure

This project was coded with a modular design so any dataset, feature level, classification method could be easily swapped for another. I organized my classes into the following packages:

- `classifierInterfaces`- Interfaces for the `PixellevelClassifier`and `strokelevelClassifiers`.
- controller - The main controller from where you can run all tests
- `\textit{kanji}Classes`- Defines a `\textit{kanji}`class and its specialized subtypes, `Pixel\textit{kanji}`and `stroke\textit{kanji}`.
- `pixellevelClassifiers`- defines the `KNN,Gaussian`and `WEKA`classifiers.
- `strokelevelClassifiers`- defines the `KNN,Gaussian`and `WEKA`classifiers.
- `utils`- Contains classes related to image and file processing.

6 Results

The collected data outperformed the CASIA database data on every test. On average, stroke-level classification outperformed Pixel-level classification.

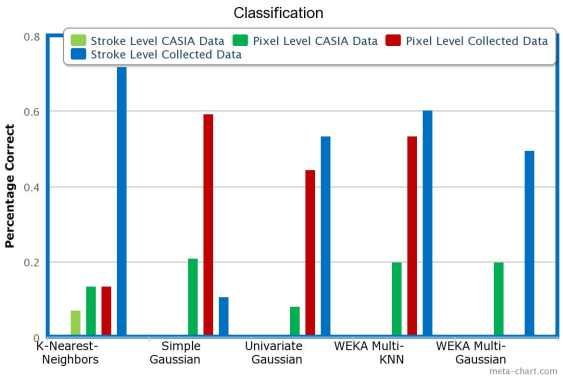

Figure 8: Classification results across feature levels and data sets.

7 Discussion

Native speakers and students have different difficulties in learning Japanese. Native speakers aren't careful about seperating their strokes. Students are write rigidly and frequently get the stroke order wrong. The difference can be seen in `Figure1` and `Figure2`. It is unsurprising that neat handwriting makes classification easier.

Pixel-level data was difficult to use since there were 400*400 features for each image. The whole CASIA dataset, contains over 35 billion features. It was practical to use a reduced dataset for the CASIA data.

The orginal datafiles for both datasets need to be reprocessed to get stroke-level data. The Collected data contains 103 datapoints per file, while the Collected data contains almost 4000 datapoints. These will be added to the Results file later when they finish processing.

Results show promise for stroke-level data collection. However, the stroke-level data is not order-agnostic. In the `FutureWork`section I talk about my plans for future order-agnostic stroke-level algorithms.

7.1 Future Work

The next step in the project is to build a bootstrapping algorithm that classifies strokes into radicals and radicals into *kanji*. This skates is stroke-order-agnostic since it has a good probability of classifying the correct radical even if the stroke order is wrong. Likewise, it has a good probability of classifying the correct *kanji* even if the radical order is wrong!

There are prior probabilities regarding the spatial ordering of the radicals can be used to inform my probabilties i.e. east, west or enclosure.

Another idea is to force stroke-order-agnostic classification is to do stroke-level preprocessing. For example, knowing how many strokes should be in a *kanji*, can inform how to recreate strokes by classifing all points the user's pen passes through. This will be informed by the rules around drawing strokes in Japanese.

8 Conclusion

In 2009, Google acquired the reCAPTCHA system, which digitizes books while protecting web services from abusive bots, effectively solving two problems at once. This study to find which recognition models work best, was the first step in solving two problems: helping Japanese-language students learn written *kanji* and improving the unsupervised learning model for Japanese character recognition.

Ultimately, I would like to crowdsource handwriting data and use it to create an Optical Character Recognition model using both radical sequences and their spatial positions. This model could then be used to create an application which would teach student to write better Japanese while also crowdsourcing data to improve the recognition model.

9 Acknowledgements

Thanks to my advisor Sravana Reddy, Dartmouth Lingustics and Cognitive Science department.

10 Sources

- Luis von Ahn, et al., reCAPTCHA: Human-Based Character Recognition via Web Security Measures
- Mitsuru NAKAI, Substroke Approach to HMM-based On-line Kanji Handwriting Recognition
- Mathieu Blondel, Unsupervised Learning of Stroke Tagger for Online Kanji Handwriting Recognition
- Junko Tokuno, et al., On-line Handwritten Character Recognition Selectively Employing Hierarchical Spatial Relationships among Subpatterns

- Mark Stamp, A Revealing Introduction to Hidden Markov Models
- Korpi Kiia, et al., Kanji Snap an OCR-based Smartphone Application for Learning Japanese Kanji Characters
- Soon-kak Kwon, et al., Character Recognition System Based On Android Smart Phone
- Honggang Zhang, et. al, HCL2000A Large-scale Handwritten Chinese Character Database for Handwritten Character Recognition
- Yohei Sobu, Hideaki Goto, Binary Tree-Based Accuracy-Keeping Clustering Using CDA for Very Fast Japanese Character Recognition
- T. M. Breuel, The OCRopus open source document analysis and OCR system
- R. Smith, An overview of the Tesseract OCR engine
- Mark L. Murphy, The Busy Coder's Guide to Android Development

- CASIA database, `http://www.nlpr.ia.ac.cn/databases/handwriting/Home.html`
- GBK table, `http://www.cs.nyu.edu/~yusuke/tools/unicode_to_gb2312_or_gbk_table.html`
- WEKA, www.cs.waikato.ac.nz/ml/weka