2016

Radical Recognition in Off-Line Handwritten Chinese Characters Using Non-Negative Matrix Factorization

Xiangying Shuai
Bard College

Radical Recognition in Off-Line Handwritten Chinese Characters Using Non-Negative Matrix Factorization

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Xiangying (Shar) Shuai

Annandale-on-Hudson, New York
May, 2016

Abstract

In the past decade, handwritten Chinese character recognition has received renewed interest with the emergence of touch screen devices. Other popular applications include on-line Chinese character dictionary look-up and visual translation in mobile phone applications. Due to the complex structure of Chinese characters, this classification task is not exactly an easy one, as it involves knowledge from mathematics, computer science, and linguistics.

Given a large image database of handwritten character data, the goal of my senior project is to use Non-Negative Matrix Factorization (NMF), a recent method for finding a suitable representation (parts-based representation) of image data, to detect specific subcomponents in Chinese characters. NMF has only been applied to typed (printed) Chinese characters in different fonts. This project focuses specifically on how well NMF works on handwritten characters. In addition, research in Chinese character classification has mainly been done using holistic approaches - treating each character as an inseparable unit. By using NMF, this project takes a different approach by focusing on a more specific problem in Chinese character classification: radical (sub-component) detection.

Finally, a possible application of radical detection will be proposed. This interactive application can potentially help Chinese language learners better recognize characters by radicals.

Contents

List of Figures

List of Tables

Dedication

I would like to dedicate this project to all of my family members - my grandparents in both China and the U.S., Chunxiu Tao, Guangcai Shuai, Amy and Jim Delaune Sr., my uncle, Gang Shuai, and my parents, Jim and Sophie Delaune. I am so very grateful for your guidance, support, and generosity all these years. Thank you!

I also want to thank Greg, for always being there for me.

Acknowledgments

First and foremost, I would like to thank my senior project advisers Amir Barghi and Sven Anderson for providing extraordinary guidance, support, patience, energy and encouragement not just this past year, but also throughout my time at Bard. Being a joint major is extremely difficult, and I want to thank both Sven and Amir for always being there for me. Without them, I would not have graduated with a joint degree. I also want to thank my senior project board members, Keith O'Hara and Ethan Bloch for their invaluable suggestions and encouragement. Also, I want to thank Sven and Ethan for giving me much encouragement as my academic advisers.

Furthermore, I want to thank Dean Bethany Nohlgren, Dean Mary Ann Krisa, and Fu-chen Chan for being such inspirational mentors this year. I also want to thank Dean Rebecca Thomas for being a wonderful role model - her success stories have strengthened my belief that as a woman I can be successful in S.T.E.M.

Lastly, I want to thank Kathleen (Katie) Burke, Alexandra Morris, and Marley Alford for being such inspirational co-clubheads of Women in S.T.E.M. @ Bard with me this year. I believe we have made a difference and will continue to make differences in improving gender equality in the S.T.E.M. fields.

1
Introduction

1.1 What is Character Recognition

Character recognition, or optical character recognition, is a field of research in computer vision, pattern recognition, and artificial intelligence that endeavors to recognize handwritten characters using computer algorithms. With the emergence of touch screen devices, the field of handwritten character recognition has received renewed interest in the past few years. Other related popular applications include signature verification, writer identification, on-line dictionary look-up, visual translation in mobile phone applications, etc.

In the research field of handwritten character recognition, "off-line" and "on-line" are important terms that describe the form of the data. The on-line case refers to the availability of trajectory data during the time of data collection, and the off-line case refers to data in the form of scanned images, or data in the form of pixels of images.

In this paper, we focus on Chinese characters due to their complex hierarchical structure and rich variations. Only off-line data is used in this project because real-time live user interaction is not required. Furthermore, this project focuses specifically on detect-

ing radicals/sub-components in Chinese characters, since this is a relatively unexplored problem.

1.2 Chinese Characters and the Significance of Radicals

The Chinese writing system is extremely hierarchical: words consist of individual characters, which in turn consist of a group of radicals ("偏旁部首", sub-components), which then in turn consist of a sequence of strokes ("笔划", the simplest components of each character). In general, a single Chinese character stands for at least one meaning, while a radical can also carry some semantic clues of the character. For example, the character "好"(h ǎ o, meaning "good"), consists of the two radicals "女" (n ǚ , meaning "female", or "daughter"), and "子" (z ǐ , meaning "son"). Figure 1.2.1 demonstrates how a Chinese character can be decomposed based on the "character-radical-stroke" hierarchical law.

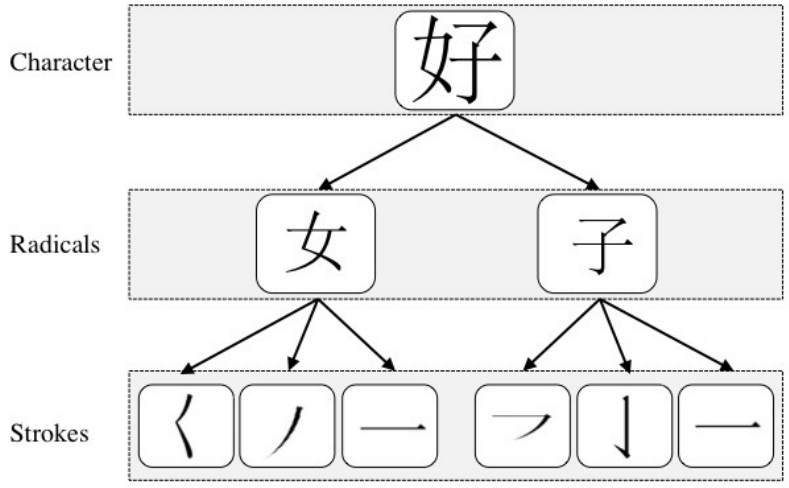

Figure 1.2.1. Hierarchical Composition of a Chinese Character: Every character can be decomposed based on the "character-radical-stroke" hierarchical law. For example, the character "好"consists of the two radicals "女", and "子", which in turn consist of a sequence of six simple strokes.

There are three known general approaches to Chinese character recognition: holistic, stroke-based, and radical-based.

Holistic Approaches: Most studies are done using holistic approaches, which recognize each character as an indivisible unit. No segmentation is performed and the whole character is recognized at once [15]. If we were to recognize characters holistically, we

would need to determine the size of the entire Chinese character set. According to statistics [11], there are over 400,000 unique Chinese characters, where 4,000 are used on a daily basis. If we were to classify them holistically and individually, the scale of usage obviously challenges holistic classification methods because naturally, even with the help of a good classification algorithm, the probability of selecting a correct character out of 4,000 is very small.

Stroke-Based Approaches: Another popular approach is to base classification on the extraction of low-level features such as strokes. Stroke-based methods focus on decomposing each character into a set of strokes, and then classify it based on the number, position, order, shape and orientation of the set of strokes [15]. However, because there are many variations (size, degree orientation, position, etc) of each type of stroke, this method essentially also has to deal with the problem of a large number of features. In addition, stroke-extraction is extremely difficult if a particular style of handwriting merges several strokes into continuous curves (similar to cursive handwriting in western languages). On the other hand, stroke-based approaches can be explored with on-line data since stroke order information would be available.

Radical-Based Approaches: Lastly, radical-based approaches decompose each character into its sub-components, or radicals, and classify the character based on the combination of the radicals and their positions within that character. There are 214 unique radicals in Chinese script, which is relatively small compared to the set of commonly used characters. Moreover, unlike strokes, each radical would only have one or two variations (vertical or horizontal). However, there are only a few existing methods that have focused on radical decomposition. Ideally, recognizing radicals is much easier than recognizing the whole character or its individual strokes. A native Chinese speaker recognizes characters based on high-level features such as the radicals (because they often contribute to the

meaning of the character), rather than low-level features such as the strokes. These observations motivate us to research on radical-based approaches to classify Chinese characters.

1.3 Databases of Handwritten Chinese

There are several databases of handwritten Chinese characters on-line, such as the CASIA On-line and Off-line Chinese Handwriting Databases built by the National Laboratory of Pattern Recognition (NLPR) and the Institute of Automation of Chinese Academy of Sciences (CASIA) [1]. In this study, we will use a relatively smaller but more recent database known as the Harbin Institute of Technology Opening Recognition Corpus for Chinese Characters (HIT-OR3C), which was collected in 2010 [4]. A sample of the data is shown in Figure 1.3.1.

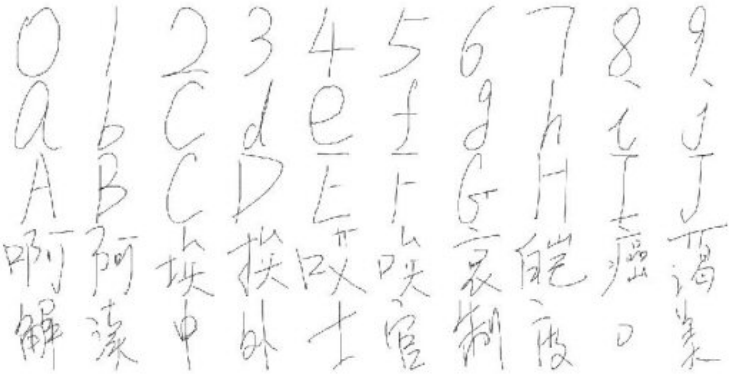

Figure 1.3.1. HIT-OR3C Data Example

The HIT-OR3C contains both on-line and off-line data of 6,825 unique classes, collected from 122 different writers, with 832,650 samples in total. The 6,825 character classes correspond to the 6,825 characters in the *Guojia Biaozhun* 2312 (GB2312, "国家标准") table, which is the registered name for a key official character set of China, used for simplified Chinese characters. The GB2312 table covers 99.75% of the characters used for daily Chinese input.

Furthermore, the 6,825 character classes are divided into two main sections: characters in the first section are arranged according to Pinyin ("拼音", the Chinese alphabet), and characters in the second section are arranged according to radicals (in increasing strokes). In this project, only the 3008 character classes in the second half of the data are used since the goal is to find radicals in the characters [4]. A complete mapping from radical classes to the characters in GB2312 can be found in Appendix A [2]. However, the map shows that only 168 out of the 214 radical classes are used in the second section of the data set.

In the HIT-OR3C data set, all the Chinese characters have been collected using a tablet with a handwriting document collection software: OR3C Toolkit, which is available for download on the website. The original individual character images are 128 by 128 grey scale.

In our off-line character recognition experiments, the image samples are converted to 128 by 128 binary matrices by averaging the RGB values of each pixel and setting a threshold of 128. That is, an average pixel value of less than 128 is considered as background and is converted to 0. Otherwise, it is considered as foreground and converted to 1. The data is converted to binary because the recognition methods we explore work well with sparse data, as explained in the next chapter.

1.4 Simple Approaches

1.4.1 Hamming Distance

Previously, we talked about how holistic approaches are quite challenging because of the large character class size. The most common and simple holistic approach in character recognition problems is template matching, where individual pixels are used as features. Classification is performed by comparing an input character with a set of templates (or prototypes) from each character class. Each comparison results in a similarity measure between an input character and a template.

We will show one holistic approach that fails to classify characters - Hamming distance, which compares the matrices of two character images and finds the number of positions at which the corresponding binary values are different. Hence, the smaller the Hamming distance is between two characters, the more similar they are to each other. To do so, we generated a set of 6,825 typed characters (which are used as templates/prototypes) corresponding to the GB2312 table, and for each input handwritten character, we calculated the Hamming distance between the input and each typed character, and picked the typed character that was "closest" to the input. We tested this method using 5,000 randomly picked input characters, and the accuracy rate was less than 1%. The explanation for the low classification rate is simple - poor alignment, as shown in Figure 1.4.1.

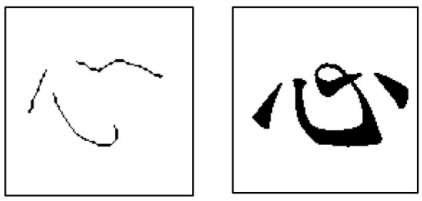

Figure 1.4.1. Poor Alignment of Characters in Hamming Distance: Simply matching handwritten characters with printed template characters results in poor alignment issues, as printed characters do not demonstrate most of the variations in handwritten characters: size, orientation, stroke density, and differences in handwriting.

Handwritten characters in general vary in size, orientation, and density, especially when there are multiple writers. Hence a simple holistic method such as this will not capture the similarities between handwritten characters.

1.4.2 Scale Invariant Feature Transform

There are some much more sophisticated methods to detect features in images, such as Scale-Invariant Feature Transform (SIFT). Hence we used SIFT to see how well it detects features in each character. A major part of SIFT is image feature generation, which transforms an image into a large collection of feature vectors, each of which is invariant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion. Fortunately, OpenCV (a library of computer vision functions)

has built-in SIFT functions to detect key points in images [6]. Figure 1.4.2 demonstrates how to use the OpenCV SIFT function to detect interesting points in an image [6].

```
import cv2
import numpy as np
img = cv2.imread('2charImg76.bmp')
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
sift = cv2.SIFT()
kp = sift.detect(gray,None)
img = cv2.drawKeypoints(gray,kp)
cv2.imwrite('sift_keypoints2.jpg',img)
```

Figure 1.4.2. SIFT Code Example. The code demonstrates how interesting features are detected in a given image. First it reads the input image and turns it into grayscale, and then it detects the points and draws them on an output image.

Figure 1.4.3 is an example of how SIFT fails to detect the same key points for two images of the same character ("爱", love) written by just two different people.

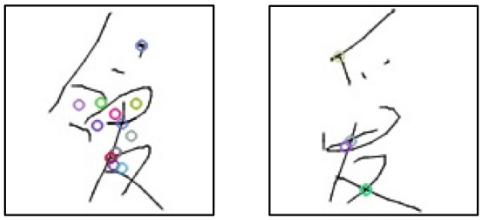

Figure 1.4.3. Incorrect Feature Detection by SIFT. The figure shows the key points detected in the same character written by two different people. It is clear that the key points in each image not only differ in the number, but also in the locations.

Poor classification results from the above experiments using holistic approaches do not show that all holistic approaches are not suitable for Chinese character recognition, nor that they are not effective. Rather, the results show the drawback of these approaches, that they require a high degree of correlation between the test and training images. In addition, holistic methods do not perform effectively under large variations in direction, scale and handwriting style. In the following chapter, we will explore a new radical-based method: non-negative sparse matrix factorization for automatically extracting radicals from Chinese characters.

2
Radical Extraction Using Matrix Factorization

2.1 Non-Negative Matrix Factorization

A fundamental problem in many pattern recognition tasks such as ours is finding a suitable representation of the data. Non-Negative Matrix Factorization (NMF), is a recently developed method for finding such a representation [3]. The NMF method has been used for many data analysis tasks such as face decomposition, font classification, gene expression clustering, and scalable Internet distance (round-trip time) prediction.

The NMF method was originally proposed by Lee and Seung to find parts of individual objects: "Non-negative matrix factorization is distinguished from other methods by its use of non-negativity constraints. These constraints lead to a parts-based representation because they allow only additive, not subtractive, combinations." By contrast, other methods, such as principal component analysis and vector quantization, learn holistic, not parts-based, data representations [9].

Definition 2.1.1. Given an $n \times m$ non-negative input matrix V, a Non-Negative Matrix Factorization is one that aims to decompose it into an $n \times r$ basis matrix W, and $r \times m$ coefficient matrix H, respectively, so that V is approximately equal to the product of W

and H as follows

$$V_{ij} \approx (WH)_{ij} = \sum_{a=1}^{r} W_{ia} H_{aj} = \hat{V}_{ij},$$

where the rank r of the factorization is generally picked so that $(n+m)r < nm$, $W \geq 0$, and $H \geq 0$. △

2.1.1 The Basic NMF Algorithm in Detail

The goal of the NMF algorithm is to decompose an input matrix V into a basis set W and encoding set H specified in Definition 2.1.1, while satisfying the non-negativity constraint. We will talk about what W and H each represents in the context of radical detection in the next section.

To approximate W and H, the NMF algorithm first initiates two random positive matrices of dimensions $n \times r$ and $r \times m$, respectively. The next step is to repeatedly calculate the difference between the estimated matrix $\hat{V} = WH$ and the input matrix V, and at the same time update the elements in both W and H to minimize this difference iteratively, until some kind of a convergence criteria is met - the maximum number of iteration steps is reached, or the total error E is less than a predefined error threshold.

To converge V and WH is to minimize the construction error $||V - WH||^2$, which is the squared error (Euclidean distance) between V and WH. The general error function E, known as Frobenius Norm, can be described as follows:

$$E(W, H) = \frac{1}{2}||V - WH||_F^2 = \frac{1}{2}\sum_{ij}(V_{ij} - (WH)_{ij})^2. \tag{2.1.1}$$

Furthermore, the element-wise error between V and $\hat{V}$ is:

$$E_{ij}(W, H) = \frac{1}{2}||V_{ij} - \hat{V}_{ij}||_F^2 = \frac{1}{2}(V_{ij} - \sum_{a=1}^{r} W_{ia} H_{aj})^2. \tag{2.1.2}$$

The above error terms are squared because the difference between the actual input matrix and the estimated matrix can be either positive or negative.

In order to minimize the error, we need to know either to increase or decrease the current element values in W and H. Generally speaking, to find a local minimum of a function, one needs to take steps proportional to the opposite/negative direction of the gradient of the cost function at each iteration, and this method is known as Gradient Descent. Hence in our case, to find the directions of element values in W and H, we differentiate the function in (2.1.2) with respect to any pair of elements W_{ik} and H_{kj} (such that $W_{ik}H_{kj} = \hat{V}_{ij}$) separately [19]:

$$\begin{aligned} \frac{\partial}{\partial W_{ik}} E_{ij} &= \frac{1}{2}\frac{\partial}{\partial W_{ik}}(V_{ij} - \sum_{a=1}^{r} W_{ia}H_{aj})^2 \\ &= -(V_{ij} - \hat{V}_{ij})(H_{kj}) \\ &= -E_{ij}H_{kj}. \end{aligned} \tag{2.1.3}$$

$$\begin{aligned} \frac{\partial}{\partial H_{kj}} E_{ij} &= \frac{1}{2}\frac{\partial}{\partial H_{kj}}(V_{ij} - \sum_{a=1}^{r} W_{ia}H_{aj})^2 \\ &= -(V_{ij} - \hat{V}_{ij})(W_{ik}) \\ &= -E_{ij}W_{ik}. \end{aligned} \tag{2.1.4}$$

Notice that in (2.1.3), the partial derivative of $(V_{ij} - \sum_{a=1}^{r} W_{ia}H_{aj})$ with respect to any W_{ik} in W is only one term, $-H_{kj}$, not a sum. This is because W_{ik} only corresponds to one of the terms in $\sum_{a=1}^{r} W_{ia}H_{aj}$, where $1 \leq k \leq r$. Hence the other terms with $a \neq k$ cancel out in the derivation process. This explanation applies to (2.1.4) as well.

Having formulated the gradient for any pair of elements W_{ik} and H_{kj}, we can now take a step proportional to the opposite of the gradients and derive the update rules for them separately:

$$W'_{ik} = W_{ik} + \alpha\frac{\partial}{\partial W_{ik}} E_{ij} = W_{ik} + \alpha E_{ij}H_{kj}. \tag{2.1.5}$$

$$H'_{kj} = H_{kj} + \alpha\frac{\partial}{\partial H_{kj}} E_{ij} = H_{kj} + \alpha E_{ij}W_{ik}, \tag{2.1.6}$$

where W'_{ik} and H'_{kj} are new matrix elements that replace W_{ik} and H_{kj}.

In common Gradient Descent and linear regression problems, α is known as the "learning rate" whose value determines the rate the algorithm is approaching the local minimum at each iteration. Usually a very small value (such as 0.001) is chosen because we want to take small steps towards each local minimum to avoid the risk of missing it.

So far we have described a set of simple additive update rules, (2.1.5) and (2.1.10). A more popular approach, proposed by Lee and Seung [9] is to update the matrices multiplicatively [9]:

$$W'_{ik} \leftarrow W_{ik} \frac{(VH^T)_{ik}}{(WHH^T)_{ik}} \tag{2.1.7}$$

and

$$H'_{kj} \leftarrow H_{kj} \frac{(W^T V)_{kj}}{(W^T WH)_{kj}}. \tag{2.1.8}$$

The major advantage of the multiplicative approach is, as long as the initiation process of the matrices assigns positive values to all elements, multiplying each matrix element by a positive value makes sure the new element is also positive. In addition, there is no learning parameter α to tune.

The multiplicative rules can easily be derived from the additive ones. First, we will rewrite the additive update rules in (2.1.5) and (2.1.10) into the following forms:

$$\begin{aligned} W'_{ik} &= W_{ik} + \alpha \frac{\partial}{\partial W_{ik}} E_{ij} \\ &= W_{ik} + \alpha \frac{\partial}{\partial W_{ik}} (V_{ij} - \sum_{a=1}^{r} W_{ia} H_{aj})^2 \\ &= W_{ik} + \alpha (V_{ij} - W_{ik} H_{kj})(H_{kj}) \\ &= W_{ik} + \alpha (V_{ij} H_{ik}^T - W_{ik} H_{kj} H_{ik}^T) \\ &= W_{ik} + \alpha (VH^T - WHH^T)_{ik} \end{aligned} \tag{2.1.9}$$

and

$$\begin{aligned} H'_{kj} &= H_{kj} + \alpha \frac{\partial}{\partial H_{kj}} E_{ij} \\ &= H_{kj} + \alpha \frac{\partial}{\partial H_{kj}} (V_{ij} - \sum_{a=1}^{r} W_{ia} H_{aj})^2 \\ &= H_{kj} + \alpha (V_{ij} - W_{ik} H_{kj})(W_{ik}) \\ &= H_{kj} + \alpha (W_{kj}^T V_{ij} - W_{kj}^T W_{ik} H_{kj}) \\ &= H_{kj} + \alpha (W^T V - W^T W H)_{kj}. \end{aligned} \tag{2.1.10}$$

Next, Lee and Seung proposed to replace the α in rule (2.1.9) with the following term:

$$\alpha = \frac{W_{ik}}{(WHH^T)_{ik}}, \tag{2.1.11}$$

and rule (2.1.9) is then rewritten into rule (2.1.7):

$$\begin{aligned} W'_{ik} &= W_{ik} + \alpha (VH^T - WHH^T)_{ik} \\ &= W_{ik} + \frac{W_{ik}}{(WHH^T)_{ik}} (VH^T - WHH^T)_{ik} \\ &= W_{ik} + \frac{W_{ik}}{(WHH^T)_{ik}} (VH^T)_{ik} - \frac{W_{ik}}{(WHH^T)_{ik}} (WHH^T)_{ik} \\ &= W_{ik} + W_{ik} \frac{(VH^T)_{ik}}{(WHH^T)_{ik}} - W_{ik} \\ &= W_{ik} \frac{(VH^T)_{ik}}{(WHH^T)_{ik}}. \end{aligned} \tag{2.1.12}$$

Similarly, the α in rule (2.1.10) can be replaced by substituting the following term:

$$\alpha = \frac{H_{kj}}{(W^T WH)_{kj}}, \tag{2.1.13}$$

and now rule (2.1.10) can be rewritten into rule (2.1.8):

$$\begin{aligned} H'_{kj} &= H_{kj} + \alpha(W^T V - W^T W H)_{kj} \\ &= H_{kj} + \frac{H_{kj}}{(W^T W H)_{kj}}(W^T V - W^T W H)_{kj} \\ &= H_{kj} + \frac{H_{kj}}{(W^T W H)_{kj}}(W^T V)_{kj} - \frac{H_{kj}}{(W^T W H)_{kj}}(W^T W H)_{kj} \\ &= H_{kj} + H_{kj}\frac{(W^T V)_{kj}}{(W^T W H)_{kj}} - H_{kj} \\ &= H_{kj}\frac{(W^T V)_{kj}}{(W^T W H)_{kj}}. \end{aligned} \tag{2.1.14}$$

Finally, a pseudo code for the NMF can be found in Algorithm 2.1.1. Lee and Seung have proved that the algorithm under update rules (2.1.7) and (2.1.8) is guaranteed to reach at least a locally optimal solution [9].

Algorithm 1 NMF Algorithm

```
1:  procedure INIT
2:      initialize W+
3:      initialize H+
4:  procedure FACTORIZE
5:      for iteration in maxNumberOfIterations do
6:          for row = i,col = j in V do
7:              compute Eij
8:              compute gradients ∂/∂Wik Eij and ∂/∂Hkj Eij
9:              calculate updated elements W'ik and H'ik
10:             compute total error E
11:             if E < errThreshold then
12:                 break
13:
          return W, H
```

In the above algorithm, "*maxNumberOfIterations*", the maximum number of iterations the algorithm will run is generally specified by the user. "*errThreshold*", the tolerance threshold for the total error at each iteration, is also used to decide when to stop updating the elements. This value is usually set to a very small positive number to ensure a close approximation of $\hat{V}$ to V.

2.1.2 NMF Applications

In recent years, researchers have applied NMF to various problems. Among the most well-known applications are face decomposition, font classification, and document classification. The most original application of NMF is face decomposition - looking for localized features that correspond with intuitive notions of the parts of faces (the eyes, the nose, and the mouth) [9]. Now let's take another look at NMF based on Definition 2.1.1. The dimensions of each matrix can be expressed as follows:

$$V_{[n\times m]} \approx W_{[n\times r]} \times H_{[r\times m]}.$$

In Lee and Seung's research, the image database of faces is regarded as an $n \times m$ matrix V, with each column being an input image in the form of a size-n vector. There are m input images in total. The r columns of W are called basis images, and each column of H is called an encoding and has a one-to-one relationship with each face in V. What is even more important to know is that an encoding consists of the coefficients by which a particular face image is represented with a linear combinations of the basis images of W [9]. Once W and H are approximated, they can be used to reconstruct a new face, as demonstrated in Figure 2.1.1 [9].

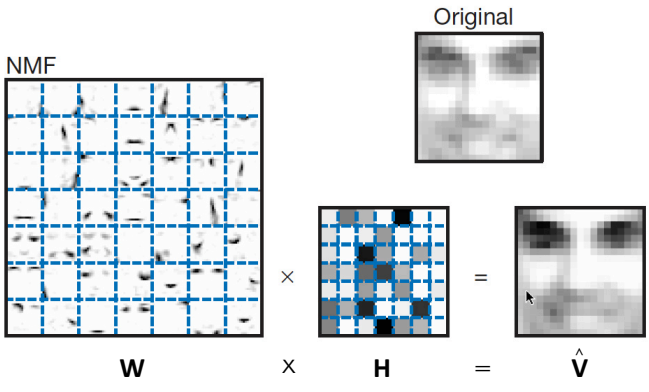

Figure 2.1.1. Reconstruction of a face using NMF. Non-negative matrix factorization learns a parts-based representation of faces: the W shown here is a set of $r = 7^2 = 49$ basis images (the eyes, the nose, the mouth, etc). A particular test image, tagged as "Original" here, is approximately represented by a linear superposition of the images in W, with encodings in H (the darker the color the bigger the coefficient matrix element is).

As can be seen from Figure 2.1.1, a large fraction of W, the NMF image basis, consists of vanishing coefficients. Hence both the basis images and image encodings are sparse

(see Definition 2.1.2). The basis images are sparse because they are not global, and they represent various versions of facial features, mouths, noses, eyes, etc, in different locations and forms. Any face in the data can be generated by combining these different parts, but the combination does not necessarily need all of these basis images [9] .

Definition 2.1.2. A sparse matrix is a matrix in which most of the elements are zero. By contrast, if most of the elements are nonzero, then the matrix is considered dense. △

Based on the above idea, many other applications including Chinese character classification can use NMF to decompose data into different features. Figure 2.1.2 and Table 2.1.2 summarize some of these applications.

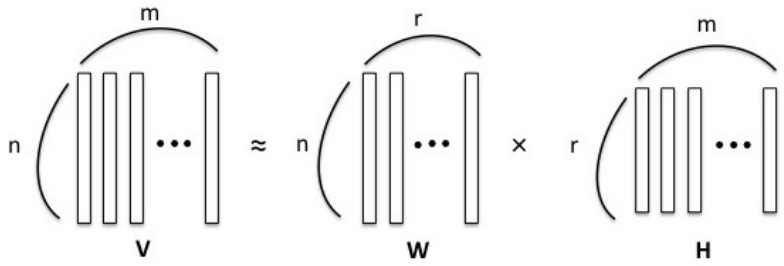

Figure 2.1.2. Visualization of NMF

Application	n	m	r
Face Decomposition	Total number of pixels in an image.	Number of face image samples.	Number of different facial features.
Font Classification	Total number of pixels in a letter/character image of a specific font.	Number of character image samples.	Number of different fonts.
Chinese Radical Detection	Total number of pixels in a character image. In this project, $n = 128^2$.	Number of character image samples.	Number of different radical classes.

Table 2.1.1. Use of Parameters in Different NMF Applications

The following section will explain in detail how NMF can be used to detect radicals in Chinese characters.

2.1.3 Outline of Radical Detection Using NMF

Tan, Xie, Zheng, and Lai were the first to use NMF on Chinese characters, and more specifically, they used the method to find radicals in printed characters. Unlike handwritten characters, the only variation in printed characters is the font [15]. Their method can be illustrated as below in two phases - training and testing, as shown in Figure 2.1.3.

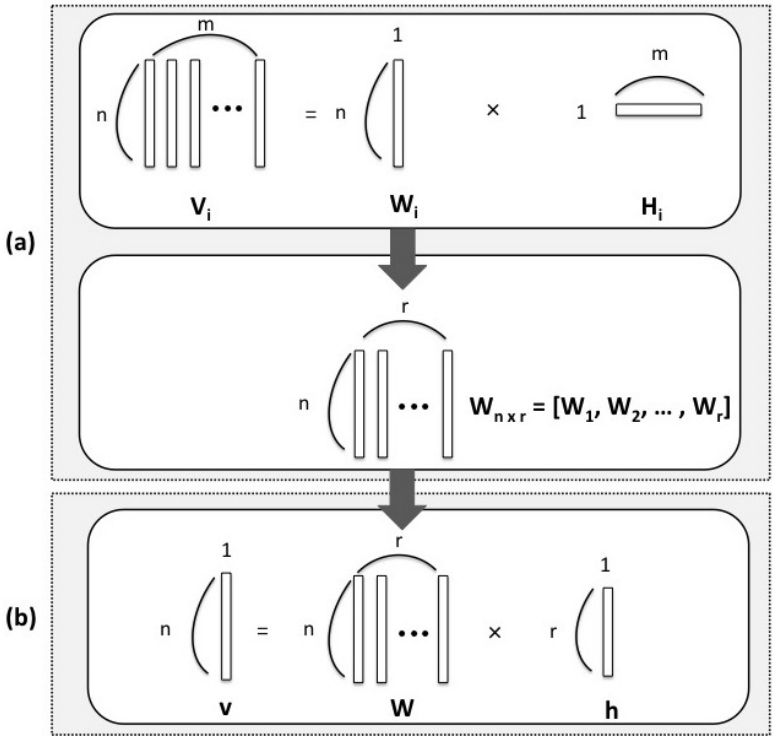

Figure 2.1.3. Illustration of the Training and Testing Phases. In (a), the training phase, each radical W_i is learned using m characters (in their column vector form) that contain the radical. W_i and H_i are estimated to best fit V_i. When all 168 radicals have been learned, together they form the radical dictionary, W. Then in (b), the testing phase, for every test image v, W is used to estimate h, a one-column coefficient matrix, which is used to determine what radicals are most prominent in the reconstruction of v.

In general NMF applications, the W matrix is known as a dictionary, and in this problem, it is known as a radical dictionary. Each column of W represents a basis radical class [15]. In other NMF applications where the features are unknown, r is usually estimated in order to better approximate W and H. In this problem, we specify $r = 1$ at each training step, because we want to isolate radical classes in order to learn them individually. As demonstrated in Figure 2.1.3, in the training phase (a), we train each radical class W_i separately from the others using NMF, with m characters that contain this radical. The training is done when we have r column vectors trained for the $r = 168$ radical classes in

the data. Then W stays constant and is passed into (b), the testing phase. Each column of W corresponds to one of the learned radical classes. Hence each testing character image (in its column vector form), can again be estimated as:

$$v \approx W \times h, \tag{2.1.15}$$

where W is known, and h is a coefficient matrix that is estimated using least squares (because W is not a square matrix, we cannot solve for h using the inverse of W). Since most characters are structured by no more than four or five components, and each $h_i \in h$ represents how important a radical class i in W is in the formation of a character, we can see if particular radicals exist in a character simply by examining the top four or five coefficients in h. In this paper, we make the assumption that all characters have no more than five components. Figure 2.1.4 demonstrates how a character in the testing phase can be reconstructed using the estimated W and H.

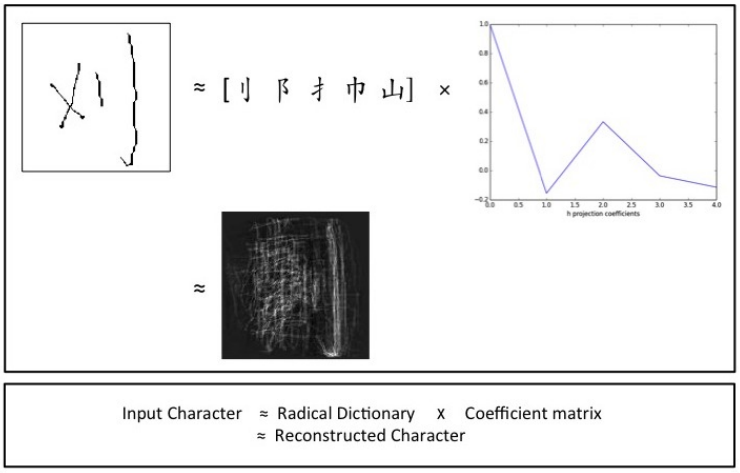

Figure 2.1.4. An Example of a Reconstructed Character: As demonstrated, the input character "刈(yì, [verb] to regulate)" is used as a testing image sample. In this example, only five radical classes are used in the radical dictionary W for demonstration purposes, with "刂" being the first radical class in W. The coefficient matrix h correctly predicted that the most prominent sub-component in "刈" is "刂" (the peak at $x = 0$). Although the left part of the reconstructed image appears to be a blur (since it contains radical classes that are not learned), the right side clearly demonstrates that "刂" is a prominent part of the test image.

2.2 NMF Results

In this section, we will record the initial results of radical detection using the following variants of NMF: standard, probabilistic, projected gradient, non-smooth, sparse (imposed on the W matrix), sparse (imposed on the H matrix), and penalized NMF. A short description of each NMF variant (except standard NMF, which is previously described in depth in Section 2.1.1) can be found in Appendix B. These are the most popular NMF methods used today. Since NMF has not been applied to handwritten character classification, we want to apply all of these methods to see which algorithms would produce good results.

The basis for comparison at this point is only the average accuracy over all radical classes. After we get a general idea of how each variant algorithm performs, we will then compare some of the algorithms in depth in the following section.

Another idea we want to explore here is how the accuracy of NMF changes as a function of the training-set size. As seen in Appendix A, not all radical classes map to the same number of characters (we will refer the number of characters a radical class maps to as its *character variability*). For example, radical class number 4, "儿(ér, son)", only maps to itself, where as radical class number 24, "阝(ěr, ear)", maps to 61 characters, such as "阢", "阡", "阱", "阪", "阽", "阼", etc. Hence for a radical class such as "儿", we get 1 character sample from each of the 122 writers in the data, leading to a total of only 122 samples representing the radical class (hence it has very little noise in the training set). On the other hand, "阝" would have $122 \times 61 = 7442$ samples (which we assume would lead to more noise in the training set). Hence, different radical classes might require training sets of different sizes for them to be better represented in the dictionary.

2.2.1 Preliminary Results

To explore how the accuracy is related to the training set size, we need to first establish a basis for comparison, using a minimum training set that is equal for all radical classes. For each radical class, we pick only 102 random samples from the pool of all 122 writers, and the testing phase will test on the remaining 20 samples. This way, all the radical classes are learned on training sets of equal size.

We will then use the seven variants of NMF to see how the results differ. It is important to state that there is no overlap between the training and testing image samples. In addition, for each variant NMF algorithm we use, the experiment is run 10 times (using identical training and testing sets for each algorithm) and returns the average measurements. The results are recorded below in Table 2.2.1. The average training and testing times are useful measures calculated to determine the efficiency of the various NMF algorithms. The training error, known as Frobenius Norm, is our error function described in Equation 2.1.1.

NMF Variant	**Avg. Training Time Per Radical Class (Seconds)**	**Avg. Training Error (Frobenius Norm)**	**Avg. Testing Time (Seconds)**	**Avg. Accuracy Per Radical Class (%)**
NMF-Standard	4.48	265.44	0.75	22.80
PMF-Probabilistic	4.26	265.62	0.77	23.21
LSNMF-Projected Gradient	2.45	265.45	0.21	21.16
NSNMF-Nonsmooth	3.21	265.71	0.21	22.38
SNMF-Sparse W	3.88	265.44	0.25	22.07
SNMF-Sparse H	7.00	265.34	0.75	22.07
PMFCC-Penalized	3.80	265.37	0.24	21.99

Table 2.2.1. Results from Different NMF Variants Using a Minimum Training Set

All of the results shown in Table 2.2.1 suggest the poor performance of NMF when using the same number of samples to train each radical class. Because of the small training set size, the W dictionary is not a good representation of character variability.

A good question to ask here is, whether using all of the available data to train each rad-

ical would lead to a better representation of character variability. To answer this question, we used standard NMF to train on all available data, and tested the resulting dictionary on the same testing samples used above (20 testing samples per radical class), and obtained an average classification accuracy of 27.08% over 10 runs. By using all of the available data to learn the radical, the average classification accuracy improved by less than 5%. It is possible that this is the best result, but it is more likely that using all of the available data impacts the dictionary's ability to generalize the radical classes and their features - that by using all of the data, too much irrelevant variation is added to the training set. This is quite a common issue in machine learning.

In the following subsection, we will make use of a concept known as learning curves to see if incrementally increasing the training size (yet without using the entire data set) will affect the results in a positive way. Plotting the learning curves for the radical classes can show us how to fit the data efficiently.

2.2.2 The Learning Curves of NMF

A learning curve is a measure of predictive performance on a given domain as a function of varying amounts of learning effort. The most common form of learning curves in machine learning shows predictive accuracy on the testing samples as a function of the number of training samples. In this radical recognition problem, the accuracy for each radical class might vary depending on the number of training samples we use. The more training samples we use, the more variations of characters are added to the training set. This improves the generalization of the dictionary due to observing more character variability.

One idea is to group all of the radical classes into equal subsets and plot learning curves for each subset. Note that it is impossible to graph a learning curve for each radical class, as we need a substantial number of radicals in a dictionary to determine the efficiency of the algorithm. Because a radical exists in a test character if it is one the top five coefficients

of h, a W matrix with $r = 1$ (r is the number of radicals/columns in the W dictionary) is extremely biased and would definitely always return a classification accuracy of 100%. Hence it is important each subset contains more than 5 radical classes.

First we need to see the distribution of the 3008 characters among the 168 radical classes. Let $R = \{r_1, r_2, \ldots, r_{168}\}$ be the set of 168 radical classes such that radical class r_i maps to $|r_i|$ characters. Appendix A shows that the smallest $|r_i|$ is 1, and the largest $|r_i|$ is 204. We plot $\sum_{i=1}^{168} r_i$ such that $|r_i| = x$ for all $x \in [1, 204]$. That is, each column in the graph represents the number of radical classes that map to x characters. The sum of all the columns is 168 as we have 168 radical classes.

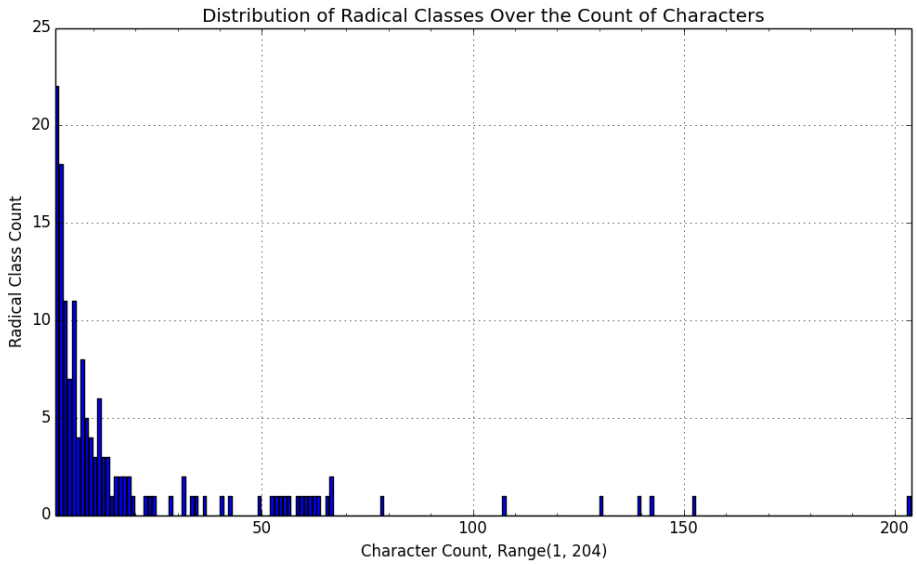

Figure 2.2.1. Distribution of Radicals Over the Count of Characters (Character Variability): The x-axis represents the number of characters a radical class maps to (character variability), and the y-axis shows the sum of the radical classes that map to x characters. It is shown that most radical classes map to fewer than 50 characters, while there are only a few radical classes that map to more than 100 characters.

Figure 2.2.1 shows that most radical classes map to fewer than 50 characters, while there are only a few radicals that map to more than 100 characters. We cannot plot the learning curves for radical classes that contain fewer than 5 characters, because the range is not wide enough to show improvement in accuracy. This excludes 197 characters out of

the set of 3008 characters. We then group the rest of the elements in R (adding the bars by multiplying them with their corresponding x value in Figure 2.2.1 in a left-to-right fashion) into three subsets of equal size, S_1, S_2, and S_3. Hence when we test a random character in a certain radical class, the probability of that radical being already trained in any one of the three subsets is equal.

After grouping the three subsets, we find that $|S_1| \approx |S_2| \approx |S_2| \approx \frac{1}{3}(3008 - 197)$, and

$$\begin{cases} S_1 = \{r_i \in R \mid 5 < |r_i| \leq 42\} \\ S_2 = \{r_i \in R \mid 43 < |r_i| \leq 78\} \\ S_3 = \{r_i \in R \mid |r_i| \geq 79\}. \end{cases}$$

In a sense, S_1 can be thought of as a subset of radical classes that have low character variability (all of them map to less than 42 characters). Similarly, radical classes in S_2 have medium character variability, and radical classes in S_3 have high character variability.

We also double-checked that each subset contains more than 5 radical classes. Now we have extracted the domains to plot the learning curve for each subset. Since each character is written by 122 different writers, it means each character sample has 122 copies. We can use 102 of those for training and 20 for testing. Hence each number in the domain is actually multiplied by 102 in training. The resulting learning curves are shown in Appendix C.

The learning curve for S_1 shows that using around 300 samples for training is ideal for radical classes in S_1. The S_2 and S_3 learning curves are not minimally monotonous (they contain many peaks), but they show some peaks which can still suggest some good numbers to use for training. The highest peaks are $x \approx 1400$ and $x \approx 1900$ for S_2 and S_3, respectively. It is clear that radical classes with high character variability require more training samples than the ones with low character variability.

Let N_i be the ideal number of samples used for training radical category r_i. Based on

our observations from the learning curves, N_i can be determined as follows:

$$N_i = \begin{cases} |r_i| \times 102 & \text{if } |r_i| \leq 5 \\ 3 \times 102 & \text{if } 5 < |r_i| \leq 42 \\ 14 \times 102 & \text{if } 43 \leq |r_i| \leq 78 \\ 19 \times 102 & \text{otherwise } (|r_i| \geq 79), \end{cases}$$

where for $|r_i| \leq 5$, all of radical class r_i's available training samples are used. It is important to emphasize that partitioning the 3008 characters into three equal subsets (in increasing character variability) is only one way to plot the learning curves. While our partitioning methodology yields good results (shown below), other methods might also lead to improvement in learning, such as forming subsets of radical classes with similar number of strokes, or making sure each subset has the same number of radical classes.

By using the above partition and the 7 NMF variants for training, we get the results in Table 2.2.2 from testing on 3,360 character images (exactly 20 characters from each of the 168 radicals). Appendix E contains our Python code for radical classification using the training data partitioning methodology described in this section.

NMF Variant	**Avg. Training Time Per Radical (Seconds)**	**Avg. Training Error (Frobenius Norm)**	**Avg. Testing Time (Seconds)**	**Avg. Accuracy Per Radical (%)**
NMF-Standard	44.20	523.35	0.22	45.56[1]
PMF-Probabilistic	12.25	523.54	0.22	41.67
LSNMF-Projected Gradient	8.70	523.25	0.21	39.26
NSNMF-Nonsmooth	50.29	523.64	0.26	41.94
SNMF-Sparse W	11.01	523.30	0.21	44.53[2]
SNMF-Sparse H	11.37	523.30	0.21	39.38
PMFCC-Penalized	28.09	523.32	0.78	38.76[7]

Table 2.2.2. Results from Different NMF Variants After Studying the Learning Curves. We boxed and ranked the accuracies of the top two and also the least efficient algorithms.

The above results show significant improvement in accuracy compared to the ones listed in Table 2.2.1. It is clear that using more training samples generally leads to better results because a higher level of character variability is expressed.

In real time machine learning applications, training is usually done beforehand, so the

training time does not matter significantly even though the training times for the 7 algorithms vary greatly. Hence we can say that the differences between the training errors are negligible. All of the testing times are under one second. Hence the differences are also not significant. What we need to focus on is the accuracy - Standard NMF and SNMF (with imposed sparsity on W) show similar accuracies, whereas the penalized model shows the lowest accuracy. In the next subsection, we will analyze how statistically different these results are.

2.2.3 Statistical Comparisons of Two Pairs of Algorithms

In machine learning, an overall classification accuracy alone is typically not enough information to help determine what is the best algorithm. The slight difference we saw in the accuracies produced by Standard NMF and SNMF might be a result of the variance in the training or testing data and other reasons. In this problem, since our classification system consists of as many as 168 classes, we cannot use common measurements such as a confusion matrix or a ROC (receiver operating characteristic curve, which is used for binary classification) to examine the performance of NMF. However, we can apply two commonly used statistical methods on our NMF results: One-Sample Paired t-Test for Population Means and One-Sample Paired F-Test for Population Variances. We will first apply the two tests once to the standard NMF and SNMF pair, to see how statistically different the two best algorithm results are. Then we will apply the two tests to the standard NMF and penalized NMF pair, to see how statistically different the best and the worst algorithm results differ.

Standard NMF & SNMF - One-Sample Paired t-Test for Population Means: The paired sets of data would be the two sets of accuracies provided by the two NMF algorithms. Each set contains 168 averaged accuracies, one for each radical class.

Let X be the set of accuracies returned by Standard NMF, and Y be the set of accuracies returned by SNMF. Since we want to see if the average results are significantly different, we want to set up two hypotheses to examine the accuracies. The null hypothesis is that the overall means of the two sets of results are equal:

$$H_0 : \overline{X} - \overline{Y} = 0 \tag{2.2.1}$$

The alternative hypothesis is that there is a difference in the overall means of the two sets of results:

$$H_1 : \overline{X} - \overline{Y} \neq 0 \tag{2.2.2}$$

The first step of the t-test is to calculate the difference ($D_i = Y_i - X_i$) between the two observations on each of the $N = 168$ pairs. Calculating the mean ($\overline{D}$) and the square of standard deviation value (s_D^2) of the differences gives:

$$\overline{D} = \frac{\sum_i D_i}{N} \approx 0.0104 \tag{2.2.3}$$

and

$$S_D^2 = \frac{\sum_i D_i^2}{N-1} - \frac{N(\overline{D})^2}{N-1} \approx 0.0526. \tag{2.2.4}$$

Then, the standard error of the mean difference ($SE(\overline{D})$) is:

$$SE(\overline{D}) = \frac{S_D}{\sqrt{N}} \approx 0.0513, \tag{2.2.5}$$

and the test statistic t ratio is found from:

$$t = \frac{\overline{D}}{SE(\overline{D})} \approx 0.2003, \tag{2.2.6}$$

with $N - 1 = 167$ degrees of freedom under the null hypothesis. Using the t-value, we can find the p-value (probability distribution) using the pt command in R (a software for statistical computing):

$$p = pt(t = 0.2003, 168 - 1) \approx 0.5793. \tag{2.2.7}$$

Suppose we use a significance level of $\alpha = 0.05$ in this test, we have $p > \alpha$, which means we cannot reject the null hypothesis, H_0. This means there is insufficient evidence to conclude that the two sets of results have different means.

Although the overall improvement of NMF over SNMF in average accuracy is only 1%, it is not a negligiable difference. It would be useful to calculate a confidence interval for the mean difference to tell us within what limits the true difference is likely to lie. A 95% confidence interval for the true mean difference is:

$$\overline{D} \pm t_{\alpha/2} \times SE(\overline{D}), \tag{2.2.8}$$

where $t_{\alpha/2}$ is the 2.5% point of the t-distribution on $N - 1 = 167$ degrees of freedom. Using R again, we can find that the confidence interval is:

$$[-0.0245, 0.0453], \tag{2.2.9}$$

which means if we do the same experiment to compare NMF and SNMF 100 times, 95 times the true value for the average difference would lie in the 95% confidence interval shown above.

Standard NMF & SNMF - One-Sample Paired F-Test for Population Variances: To build a more solid analysis of the difference between the two sets of results, we can also compare their variances, σ_X^2 for SNMF and σ_Y^2 for NMF. To do so, we can make the following hypothesis:

$$H_0' : \sigma_X^2 - \sigma_Y^2 = 0 \tag{2.2.10}$$

and

$$H_0' : \sigma_X^2 - \sigma_Y^2 \neq 0. \tag{2.2.11}$$

Suppose we choose a significance level of $\alpha = 0.05$ again. The test statistic F is:

$$F = \frac{S_X^2}{S_Y^2} \approx \frac{0.2208^2}{0.2603^2} \approx \frac{0.0488}{0.0678} \approx 0.7198. \tag{2.2.12}$$

Then using R again, we can find the probability distribution for the F-statistic is:

$$p \approx 0.0091, \tag{2.2.13}$$

which means, with 95% confidence, the null hypothesis H_0' is rejected. Furthermore, we find that the confidence interval for the difference in the two population variances is:

$$[0.4917, 0.9039]. \tag{2.2.14}$$

That is, we are 95% confident that the ratio of the two population variances is between 0.4917 and 0.9039. Since the interval does not contain the ratio value 1, we can conclude that the population variances differ. Table 2.2.3 and Table 2.2.3 sum up the comparisons between the two algorithms.

Table 2.2.3. t-Test for Population Means - Standard NMF and SNMF

	t-value	N-1	Significance (α)	p-value	Confidence Interval
Difference	0.2003	167	0.05	0.5793	[-0.0245, 0.0453]

Table 2.2.4. F-Test for Population Variances - Standard NMF and SNMF

	F-value	N-1	Significance (α)	p-value	Confidence Interval
Difference	0.7198	167	0.05	0.0091	[0.4917, 0.9039]

Based on the above tests, we can conclude that although there is insufficient evidence to conclude which algorithm yields a better overall accuracy, we see that there is lower variance among the average accuracies of the radical classes in standard NMF. That is, the 168 accuracies in SNMF disperse further from their population mean than those of NMF. As described in Appendix B, SNMF imposes sparseness (more zero elements) on W. It is possible that the level of sparseness is uneven among the radical classes (columns) in W, leading to more variance among the performance of its radicals.

In conclusion, standard NMF should be used if there there is a preference for accuracies closer to the expected values, whereas SNMF (with imposed sparseness on W) can be used

when there is a preference in a shorter training time.

Standard NMF & Penalized NMF - One-Sample Paired t-Test for Population Means and F-Test for Population Variances: We will follow the same calculations described previously to derive the two test statistics for standard NMF and penalized NMF. The details are omited and we will just show the results in Tables 2.2.3 and 2.2.3.

Table 2.2.5. t-Test for Population Means - Standard NMF and Penalized NMF

	t-value	N-1	Significance (α)	p-value	Confidence Interval
Difference	0.7137	167	0.05	0.7618	[0.0031, 0.1330]

Table 2.2.6. F-Test for Population Variances - Standard NMF and Penalized NMF

	F-value	N-1	Significance (α)	p-value	Confidence Interval
Difference	1.7138	167	0.05	1.663e-07	[1.6786, 3.0854]

Again, the One-Sample Paired t-Test shows that we cannot reject the null hypothesis that the populations means of standard NMF and penalized NMF are the same. There is insufficient evidence to conclude that the two sets of results have different means. What we can conclude is that the standard NMF results are much more variant than that of penalized results.

Appendix D contains plots that compare the means and variances of all 168 radical classes between the two pairs of algorithms: standard NMF and SNMF (the two best algorithms), and standard NMF and penalized NMF (the best and the worst). Figure D.0.1 and Figure D.0.2 are great illustrations that show although standard NMF and SNMF yield similar accuracies among all radical classes, there is greater variance among the performance of radical classes in SNMF. Furthermore, Figure D.0.3 and Figure D.0.4 demonstrate that although there is greater variance among the performance of individual radicals in standard NMF, it yields much better results than the penalized method.

3
Conclusion

3.1 Discussions and Comparisons

Among the existing methods for handwritten Chinese character classification, the radical-based approach may be the most similar to human cognition, given that most native Chinese speakers recognize characters by their radicals (because they give semantic clues about the meanings of the characters). However, radical extraction in Chinese characters is still relatively unexplored because of the challenges it presents. In this paper, we used Non-Negative Matrix Factorization (NMF) to extract the radicals in handwritten characters. NMF is a very recent methodology proposed to find "parts-based" representation of objects (usually imagery or textual objects). This method had only been previously used on printed characters [15].

In this paper, we find that learning each radical using a training set with a size determined based on the radical's character variability (the number of characters it maps to) yields better results than using a minimum or maximum training set. After learning all 168 radicals (as column vectors), a radical dictionary matrix is formed, which we can use to predict what radicals are present in new testing image samples. Furthermore, because

we did not know which NMF variants would yield good results, we used seven different variants of NMF (including the standard one) to learn the dictionary. Based on the average classification accuracies (using the same training and testing samples for all algorithms over 10 runs/executions), we find that the standard NMF algorithm and Sparse NMF (with an imposed sparseness on the dictionary matrix) yield the best results. While the standard algorithm performs well on all radicals, the individual radical class performance in SNMF shows more variance.

Without modifying the input data images, Non-Negative Matrix Factorization already produces positive results (a 45% accuracy rate and a testing time that is less than a second) that exceed our expectations, also given that the data has so many variations (size, rotation, and different handwriting styles). We believe that with some modifications in the data itself, NMF will perform more efficiently.

The original research (by Tan, Xie, Zheng, and Lai) that applied NMF to printed Chinese characters made two more modifications: Taking away the non-negative constraint and normalizing all images to a bounding shape (which we will describe in the future work section) [15]. Their methods produced a character classification rate of 99.2%. It is difficult to compare our results with theirs since they focused on character classification and we focused on radical classification. However, since they assumed each character has at most two radicals, we can estimate their radical classification rate to be approximately $\sqrt{99.2}\% \approx 99.6\%$ (assuming that both radicals in each character are independent), which is a lot better than ours. On the other hand, the scale of their experiments was much smaller. They learned only 59 radicals from 648 printed characters, and used only 1029 testing samples. It is natural that classification of handwritten characters is more challenging than that of printed characters because of the handwriting style variations in handwritten characters.

There is only one paper that we know of that used the same data set, HITPU [13].

The author, Daming Shi, reported a radical classification rate of 96.5% and a character classification rate of 93.5%. His experiments were conducted using 200 radical classes on a test set of 430,800 characters from 2,154 character classes [13] (whereas we used 168 radicals, 3,360 testing samples from 3008 character classes), which is on a much larger scale than ours and produced much better results. Shi's approach is entirely different. For training, he uses kernel principle-component analysis to find "landmark" points in the training samples and capture the main variations around the mean radical. He makes the assumption that each character consists of up to four unique radicals. For testing, chamfer distance minimization is used to match radicals within a character using the dynamic tunneling algorithm to search for the best shape parameters to describe the deformation of an active model to fit the test image [13]. The author used a combination of algorithms to best capture the many variations in Chinese characters.

Based on the above comparisons, it is clear that although the standard NMF algorithm already produces meaningful results on handwritten characters produced by different writers, alone it is not enough to produce comparable results. Nevertheless, it has been proved that NMF performs well in finding parts and sub-components of objects. Furthermore, radical classification can be used in many useful real-world applications, and we will make a proposal for one in the following section.

3.2 A Proposal for a Character Learning Application

As mentioned in the introductory sections, handwritten Chinese character recognition has received renewed interest with the emergence of touch screen devices. An on-line dictionary called Line Dictionary, which bills itself as "more than a dictionary", has a handy feature that allows users to find Chinese characters by drawing them with a mouse, a track pad or a tablet. Figure 3.2.1 is an illustration of the feature [10].

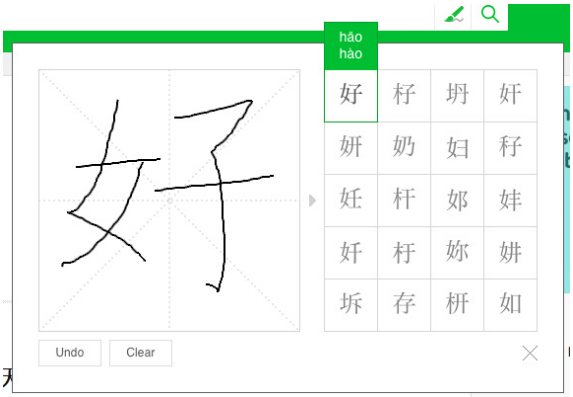

Figure 3.2.1. Dictionary Look-Up Drawing Application. This on-line application allows the user to draw a character using a mouse, and then it makes suggestions in real time about what that character might be. The example shown here is a hand drawn character, 好(good), and it is clear that the application makes a correct prediction.

Unlike English, the sound (Pinyin) of a Chinese character does not correlate with its shape, and to type a character, one must know its sound. The application in Figure 3.2.1 is useful when the user does not know the Pinyin of the character, which happens frequently to non-native speakers. While an application like this is becoming more and more popular, there still does not exist a similar application that extracts radicals from a hand drawn character. Thereby we will make a proposal for such an application here (Due to time constraints, we will not be able to actually implement the idea). We propose the following layout for the radical learning application:

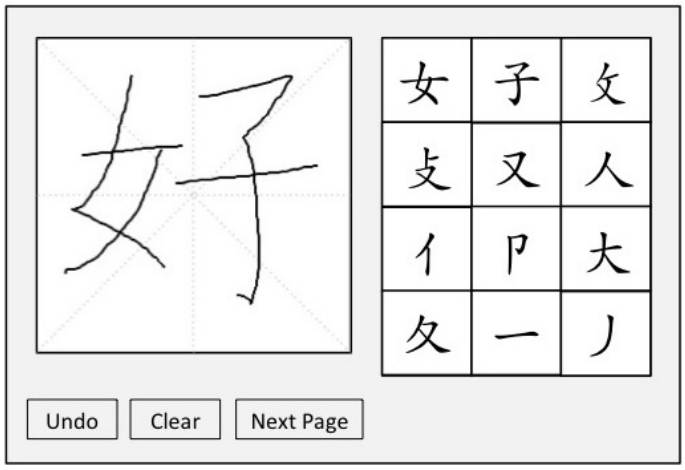

Figure 3.2.2. A Proposed Layout for a Character Learning Application. We model the new application based on the layout of the previous one, with an additional "Next Page" button to allow the user to look at more options.

The initial step of building the application requires that we learn the radical dictionary W beforehand and save the matrix into a file. Then, whenever a user finishes writing, we convert the character image into a matrix of binary elements. Next, we use W to estimate

the coefficient matrix h, pick the top 12 coefficients in h, and display the corresponding radicals. Finally, when the user picks the desired radicals, the application should display the following information about the radical: meaning, number of strokes, etc.

We model the new application based on the layout of the previous one, with an additional "Next Page" button to allow the user to look at more options. Recall that in our experiment, we defined that a radical is found in a testing sample if its index appears to be in the top five elements of the estimated coefficient matrix, h. The layout of the proposed application allows there to be at least 12 suggestions of radicals, and the user is allowed to look at more suggestions if the desired radicals are not on the first page. This means the actually radical classification rate might be much higher than 45%. However, when building and testing the model, we still need to take into account of new styles of handwriting being introduced to the testing set. One solution is to update the W dictionary constantly as new styles of handwriting appear.

3.3 Conclusion

In this project, we used a novel approach known as Non-Negative Matrix Factorization (NMF), a recent method for finding a suitable parts-based representation of imagery data, to detect sub-components (known as radical characters) in handwritten Chinese characters. Whereas most researchers focus on holistic approaches, by using NMF, this project takes a different approach and a human cognitive perspective by focusing on a more specific problem - radical detection.

In our experiments, seven variants of the NMF algorithm were used to learn a dictionary that represents 168 radical classes. We find that learning each radical using a training set of a size most suitable to represent its character variability produces better results than using a minimum or maximum training set. Furthermore, by testing on 3,360 character image samples, the best accuracies are 45.56% and 44.53%, produced by standard NMF

and sparse NMF (SNMF, with enforced sparseness on the dictionary matrix), respectively. We have also found that while standard NMF performs well on all radical classes, SNMF is much faster and shows more variance. We assume the variance is a result of uneven sparseness among the learned radical classes.

Finally, we proposed a character learning model which can potentially be built into a mobile phone or web application. The application would allow the user to first hand draw a Chinese character, then it predicts what radicals may be present in the character image, and after the user selects the desired radicals, the application displays the following information about the chosen radicals: meaning, number of strokes, etc. From a linguistic point of view, we believe that learning radicals and their shapes and meanings would help a learner better recognize Chinese characters and memorize their meanings.

NMF has shown to be a dynamic method that produces meaningful results for many different types of problems. For future study, it would be interesting to see NMF being applied to other Eastern Asian languages that also have hierarchical character structures, such as Korean and Japanese.

3.4 Future Work

The researchers that applied NMF to printed Chinese characters also proposed two modification of the problem: changing the non-negative constraint in NMF, and normalizing the input data using Affine transform. We will briefly describe these two proposals in the following subsections.

3.4.1 Constrained Sparse Matrix Factorization

Due to the non-negative restraint, NMF meets many challenges and requirements in real applications. In Tan, Xie, and Zheng's study, they propose to drop the non-negative constraint, because although it may affect the sparseness, is unnecessary for character de-

composition [15]. They introduced Constrained Sparse Matrix Factorization (CSMF) as an improvement of NMF. CSMF guarantees the sparseness by dropping the non-negative constraint, and at the same time it brings in penalized functions. Usually, penalty terms are added to the new method in order to replace a constrained optimization problem by unconstrained problems whose solutions ideally converge to the solution of the original constrained problem.

In CMSF, V corresponds to the character set, W corresponds to the radical set, and H is a matrix of unique decomposition coefficients specific to V and with respect to basis W [15]. Finding the point of convergence between V and WH in CMSF is the same as NMF - we need to minimize the construction error $||V-WH||^2$, which is the squared error (Euclidean distance) between V and WH. Then the CSMF can be described as follows:

$$\min_{W,H} E(W,H) = \frac{1}{2}||V - WH||_F^2 + g_1(W) + g_2(H) \tag{3.4.1}$$

$$\text{s.t.} W \in D_1, H \in D_2$$

where D_1 and D_2 are domains of W and H, and g_1 and g_2 are penalized functions of W and H, respectively [15] [18]. The penalty functions are described as follows:

$$g_1(W) = \alpha \sum_{j_1=1}^{r} \sum_{j_2=1}^{r} \sum_{c, j_2 \neq j_1}^{n} |W_{j_1}(c)||W_{j_2}(c)| + \beta \sum_{j=1}^{r} \sum_{c=1}^{n} |W_j(c)|, \tag{3.4.2}$$

and

$$g_2(H) = \lambda \sum_{k=1}^{m} \sum_{j=1}^{r} |h_k(j)|, \tag{3.4.3}$$

$$\text{s.t. } W \in \mathbb{R}^{n \times r}, W \geq 0 \text{ and } H \in \mathbb{R}^{r \times m}$$

where α, β, and λ are non-negative weights/coefficients specifying the importance of each term [15] [18]. It is easy to see that CSMF evolves from NMF because when $D_1 = \{W \geq 0\}$, $D_2 = \{H \geq 0\}$ and $\alpha = \beta = \lambda = 0$, CSMF is essentially NMF.

With the penalty functions $g_1(W)$ and $g_2(H)$ defined, it is easier to see that that penalty

terms are placed where the original constraints are violated to compensate for the violation. Ideally, the solutions of the unconstrained problems will eventually converge to that of the original constrained problem.

3.4.2 Affine Sparse Non-Negative Matrix Factorization

Because of the complexity of Chinese characters, and the variations of handwriting in our data set, NMF might not perform well as it may result in radical extraction variations in location, scale, or direction. To overcome this problem, Tan, Xie, Zheng, and Lai proposed to apply Affine transformation on all characters in the date set, and called this modified method Affine Sparse Matrix Factorization (ASMF) [15].

The Affine transformation procedure can be seen as the normalization step of the data set, so that all characters in the data set can have an identical shape, which is bounded by a rectangle and resembles a TBLR quadrilateral, as shown in Figure 3.4.1. [15].

(a) (b)

Figure 3.4.1. Normalization of Data: (a) Bounding rectangle, (b) TBLR quadrilateral.

ASMF Overview: Let f denote the Affine transformation. Then we can combine the previously described CSMF with Affine transformation and propose the following ASMF definition:

$$\begin{aligned} \min_{W\in D_1, H\in D_2} E(W,H) &= \frac{1}{2}||A - WH||_F^2 + g_1(W) + g_2(H) \\ &= ||f(V) - WH||^2 + g_1(W) + g_2(H) \end{aligned} \tag{3.4.4}$$

where

$$A_{iu} = f(V_{iu}) \approx (WH)_{iu} = \sum_{a=1}^{r} W_{ia}H_{au}$$

and D_1, D_2, g_1 and g_2 are described in the previous section.

Now the Affine tranformation can be described as follows:

$$\begin{aligned} A_i &= f(V_i) = AV_i + b = A_s A_t A_u A_\theta V_i + b \\ &= \begin{pmatrix} s & 0 \\ 0 & s \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & t \end{pmatrix} \begin{pmatrix} 1 & u \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} V_i + \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} \\ &= \begin{pmatrix} s \cdot \cos(\theta) & (stu) \cdot \cos(\theta) - (st) \cdot \sin(\theta) \\ s \cdot \sin(\theta) & (stu) \cdot \sin(\theta) - (st) \cdot \cos(\theta) \end{pmatrix} V_i + \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}, \end{aligned} \tag{3.4.5}$$

where A_s, A_t, A_u, A_θ, and b denote the scaling, stretching, skew, rotation, and translation parameters, respectively. The Affine transformation will improve the alignment of characters by adjusting them into a common normal structure.

For future work, it would be interesting to see if the modifications that work well on printed characters would also improve NMF's performance on handwritten characters.

Appendix A

Map of Radicals to GB2312

The following is a radical index for the 3,008 GB 2312-80 Level 2 hanzi, arranged according to a reduced set of 186 radicals (this index also applies to GB/T 12345-90 Level 2 hanzi). The GB 2312-80 Row-Cell codes for the radicals themselves are also provided under the "Radical"column. Note that some radicals do not have a corresponding range—hanzi categorized under such radicals are in GB 2312-80 Level 1 hanzi.

Number	Radical	GB 2312-80
1	50-27 c	5601–5612
2	56-13 丨	5613–5614
3	56-15 丿	5615–5627
4	56-28 丶	5628
5	50-50 乙	5629–5632
6	22-94 二	5633
7	42-14 十	5634–5637
8	19-07 厂	5638–5645
9	56-46 匚	5646–5651
10	18-23 卜	5652–5653
11	56-54 刂	5654–5670
12	56-71 冂	5671–5672
13	56-73 亻	5673–5757
14	40-43 人	5758–5765
15	16-43 八	5766–5771
16	57-72 勹	5772–5775
17	28-24 几	5776–5777
18	22-89 儿	5778
19	57-79 亠	5779–5790
20	57-91 冫	5791–5801
21	58-02 冖	5802–5804

Number	Radical	GB 2312-80
22	58-05 讠	5805–5863
23	58-64 卩	5864–5865
24	58-66 阝	5866–5926
25	21-22 刀	5927–5928
26	33-06 力	5929–5936
27	51-54 又	5937–5939
28	59-40 厶	5940
29	59-41 凵	5941–5943
30	59-44 厶	5944–5946
31	25-04 工	5947
32	45-33 土	5948–6015
33	42-31 士	6016–6018
34	60-19 艹	6019–6234
35	62-35 艹	6235–6236
36	20-83 大	6237–6243
37	62-44 尢	6244–6247
38	62-48 扌	6248–6313
39	20-71 寸	
40	63-14 弋	6314–6317
41	31-58 口	6318–6476
42	64-77 囗	6477–6487

Number	Radical	GB 2312-80
43	29-77 巾	6488–6506
44	41-29 山	6507–6559
45	65-60 彳	6560–6573
46	65-74 彡	6574
47	65-75 犭	6575–6621
48	47-06 夕	6622–6625
49	66-26 夂	6626
50	66-27 饣	6627–6646
51	25-67 广	6647–6663
52	66-64 门	6664–6736
53	35-37 门	6737–6759
54	67-60 丬	6760–6762
55	67-63 讠	6763–6917
56	69-18 宀	6918–6932
57	69-33 辶	6933–6969
58	69-70 彐	6970–6973
59	42-12 尸	6974–6981
60	25-13 弓	6982–6987
61	28-26 己	
62	69-88 屮	6988
63	37-14 女	6989–7055
64	48-01 小	7056–7057
65	55-51 子	7058–7063
66	34-77 马	7064–7088
67	70-89 纟	7089–7158
68	71-59 幺	7159–7160
69	71-61 巛	7161–7163
70	45-85 王	7164–7223
71	46-04 韦	7224–7226
72	36-30 木	7227–7363
73	40-14 犬	7364–7365
74	20-85 歹	7366–7376
75	19-21 车	7377–7406
76	24-74 戈	7407–7416
77	17-40 比	
78	45-63 瓦	7417–7422
79	54-25 止	
80	74-23 攴	7423
81	40-53 日	7424–7457
82	17-20 贝	7458–7471
83	28-91 见	7472–7479
84	37-03 牛	7480–7491
85	42-54 手	7492–7502
86	35-11 毛	7503–7512

Number	Radical	GB 2312-80
87	38-88 气	7513–7521
88	75-22 攵	7522–7524
89	38-12 片	7525–7527
90	29-79 斤	
91	55-06 爪	7528–7529
92	52-34 月	7530–7602
93	39-23 欠	7603–7608
94	23-71 风	7609–7614
95	76-15 殳	7615–7618
96	46-36 文	7619–7621
97	23-29 方	7622–7629
98	22-23 斗	
99	27-80 火	7630–7664
100	24-24 父	
101	76-65 灬	7665–7668
102	27-07 户	7669–7673
103	76-74 礻	7674–7692
104	48-36 心	7693–7716
105	77-17 丰	7717–7718
106	43-14 水	7719–7721
107	46-67 毋	
108	42-30 示	
109	42-15 石	7722–7771
110	33-90 龙	7772
111	50-21 业	7773–7775
112	36-31 目	7776–7813
113	44-79 田	7814–7822
114	43-36 四	7823–7832
115	35-83 皿	7833–7835
116	78-36 钅	7836–7981
117	42-24 矢	7982–7984
118	26-44 禾	7985–8006
119	16-55 白	8007–8011
120	25-47 瓜	8012–8013
121	51-35 用	8014
122	36-81 鸟	8015–8057
123	80-58 疒	8058–8119
124	33-02 立	8120–8121
125	49-08 穴	8122–8133
126	81-34 衤	8134–8165
127	81-66 疋	8166–8167
128	38-04 皮	8168–8169
129	35-12 矛	8170
130	81-71 耒	8171–8182

Number	Radical	GB 2312-80
131	32-47 老	8183
132	22-90 耳	8184–8190
133	19-28 臣	
134	46-87 西	8191
135	50-19 页	8192–8213
136	82-14 虍	8214–8215
137	19-70 虫	8216–8329
138	83-30 缶	8330–8333
139	41-64 舌	8334
140	54-81 竹	8335–8406
141	30-42 臼	8407-8410
142	55-52 自	8411
143	49-10 血	8412
144	54-59 舟	8413–8431
145	50-34 衣	8432–8437
146	49-82 羊	8438–8443
147	35-55 米	8444–8461
148	84-62 艮	8462–8463
149	51-80 羽	8464–8472
150	84-73 糸	8473–8478
151	34-83 麦	8479–8480
152	55-63 走	8481–8485
153	19-64 赤	8486–8487
154	22-25 豆	8488–8489
155	51-47 酉	8490–8524
156	19-29 辰	
157	85-25 豕	8525
158	34-17 卤	8526

Number	Radical	GB 2312-80
159	32-79 里	
160	55-67 足	8527–8583
161	41-77 身	
162	18-41 采	
163	85-84 豸	8584–8589
164	29-39 角	8590–8603
165	49-52 言	8604–8605
166	48-33 辛	
167	25-40 谷	
168	39-64 青	8606
169	38-68 其	
170	51-74 雨	8607–8618
171	19-61 齿	8619–8627
172	86-28 黾	8628–8630
173	86-31 隹	8631–8637
174	29-80 金	8638–8646
175	51-67 鱼	8647–8715
176	24-79 革	8716–8725
177	25-39 骨	8726–8739
178	25-77 鬼	8740–8746
179	42-19 食	8747–8751
180	50-84 音	
181	87-52 髟	8752–8764
182	34-73 麻	8765–8767
183	34-25 鹿	8768–8775
184	26-58 黑	8776–8786
185	42-83 鼠	8787–8791
186	17-39 鼻	8792–8794

Appendix B
Brief Descriptions of the NMF Variants

All of the NMF variant algorithms used in this project are implemented in Nimfa, a Python library for non-negative matrix factorization. In Nimfa, both dense and sparse matrix representation are supported [20]. While the standard NMF algorithm is easy to implement and usually yields good results, researchers have also come up with other methods that are effective. In this appendix, we will give a short description (including the objective function if it differs from the standard one) for the other five algorithms and the motivation to use each of them. The individual update functions and their derivations will be omitted.

B.0.3 Probabilistic Model (PMF)

We usually see the NMF problem from an optimization point of view. However, there is also a probabilistic interpretation of the NMF models. Given the probabilistic characterization, classical multiplicative update rules can be derived as an maximum likelihood estimation algorithm (MLE, which is a method of estimating the parameters of a model given some data). In their original NMF article, Lee and Seung proposed another objective function:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{m} [V_{ij} log(WH)_{ij} - (WH)_{ij}], \tag{B.0.1}$$

which also subjects to the non-negativity constraints. This objective function can be derived by interpreting NMF as a way to construct a probabilistic model, in which each pixel element V_{ij} is generated by adding Poisson noise to the product, $(WH)_{ij}$. It follows that this objective function is related to the likelihood of generating the images in V from the basis W and encodings matrix H [9].

B.0.4 Alternating Least Squares with Projected Gradient (LSNMF)

The projected gradient approach is also commonly used because it converges much faster than the popular multiplicative update approach, as shown in Table 2.2.1 and Table 2.2.2.

The algorithm uses the same objective function in Equation 2.1.1. However, it updates entire matrices instead of individual elements (hence the faster speed). It alternates in the sense that it fixes either W or H while updating the other:

$$\text{Find } W^{k+1}, \text{such that } E(W^{k+1}, H^k) \leq E(W^k, H^k), \text{ and} \tag{B.0.2}$$

$$\text{Find } H^{k+1}, \text{such that } E(W^{k+1}, H^{k+1}) \leq E(W^{k+1}, H^k), \tag{B.0.3}$$

where E is the error function described in Equation 2.1.1, and k is the iteration number.

B.0.5 Non-smooth Model (NSNMF)

It is common that the basis and encoding vectors in W and H produced by NMF display a high degree of overlapping among basis vectors, which contradicts the intuitive nature of the "parts". For example, in Figure 2.1.1, there are many similar parts that resemble the same facial feature. Hence an NMF variant capable of producing more localized and less overlapped feature representations of the data is desired. A new variant, named as Non-smooth NMF, has been proposed to produce more centralized factorizations.

The new model differs from the original one in the use of an extra smoothness matrix for imposing sparseness: the target matrix V is estimated as the product, $V \approx WSH$. The positive symmetric square matrix S, is a smoothing matrix defined as:

$$S = (1 - \theta)I + \frac{\theta}{r}\mathbf{1}\mathbf{1}^T, \tag{B.0.4}$$

where I is the identity matrix, 1 is a vector of ones, and the parameter θ satisfies $0 < \theta < 1$.

The interpretation of S as a smoothing matrix can be explained as follows: Let X be a positive and nonzero vector. Consider the transformed vector $Y = SX$. If $\theta = 0$, then $Y = X$ and no smoothing on X has been performed. However, as θ approaches 1, the vector Y tends to be the constant vector with all elements almost equal to the average of the elements of X. This means X is the smoothest possible vector because all entries are almost equal to the same nonzero value. Due to the multiplicative nature of the objective function, strong smoothing in S forces strong sparseness in both the basis and encoding matrices, with the parameter θ controlling the sparseness of the model [20]. This method does not produce good results because it imposes sparseness on both W and H, and we will see in the model description in SNMF that sparseness in H is not needed.

B.0.6 Enforced Sparseness (SNMF)

Although standard NMF is effective most of the time, it does not always result in part-based representations. Some researchers have shown that incorporating the notion of "sparseness" (the number of zero elements) improves the found compositions. Furthermore, the degree of sparseness in W and H can even be controlled and imposed. Researchers Kim and Park introduced two formulations to impose sparseness on the W and H matrices separately [7].

To apply sparseness constraints on W, they proposed the following objective function:

$$E(W, H) = \frac{1}{2}||V - WH||_F^2 + \eta||H||_F^2 + \alpha\sum_{i=1}^{m}||W(i,:)_1^2||, \tag{B.0.5}$$

where $W, H \geq 0$, $W(i,:)$ is the i-th row vector of W, $\alpha > 0$ is a parameter to suppress $||H||_F^2$, and $\eta > 0$ is a regularization parameter to balance the trade-off between the accuracy of the approximation and the sparseness of W.

On the other hand, they proposed the following objective function for H:

$$E(W,H) = \frac{1}{2}||V - WH||_F^2 + \eta||W||_F^2 + \beta\sum_{j=1}^{n}||H(:,j)_1^2||, \tag{B.0.6}$$

where $W, H \geq 0$, $H(:,j)$ is the j-th column vector of H, $\eta > 0$ is a parameter to suppress $||W||_F^2$, and $\beta > 0$ is a regularization parameter to balance the trade-off between the accuracy of the approximation and the sparseness of H [7].

The results in our experiments show that imposing sparseness on the dictionary, W, is quite beneficial since it produces more localized and less overlapped features in W. On the other hand, imposing sparseness on H does not produce good results because having more zero elements does not help elements in H to express the importance of each part in W.

B.0.7 Penalized Model (PMFCC)

Some researchers claim that since a simple objective function such as Frobenius Norm does not involve any guidance from the data labels or prior knowledge of the data (if there is any), it makes the results unreliable. In their paper "Semi-Supervised Clustering via Matrix Factorization", researchers Wang, Li, and Zhang proposed a new objective function with a penalty term [17]:

$$E(W,H) = ||V - WH||_F^2 + tr(W\theta W^T), \tag{B.0.7}$$

where θ is a constraint matrix. In their paper, prior knowledge on data refers to the existence of some pairwise constraints indicating similarity or dissimilarity relationships between training samples. The knowledge that indicates two points belong to the same class is defined as a must-link constraint, M, where as the knowledge that indicates two points belong to different classes is defined as a cannot-link constraint, C. Hence θ is defined as

$$\theta_{ij} = \begin{cases} \theta_{ij}, & (V_i, V_j) \in C \\ -\theta_{ij}, & (V_i, V_j) \in M \\ 0, & \text{otherwise,} \end{cases}$$

where V_i and V_j are the i-th and j-th columns in the input data matrix V.

The reason why PMFCC performed poorly in our experiments is perhaps this model is more suitable for problems with an input matrix V with some columns that are of different classes. The way we formated our problem is that V contains character samples that are of the same radical class.

There are many other variations of the penalty function, and all of them add penalty values to the objective function for violating some given constraints. The penalized model written in Nimfa does not seem to be suitable for our problem, but it provides a good basis for comparison.

Appendix C
Plots of Learning Curves

Each of the following 2 figures shows the comparisons between the learning curve for each of the 3 subsets of radical classes formed in Section 2.2.2. As mentioned in that section, the learning curves can suggest what amount of training data is good for learning the radical dictionary. The y-axis in the first figure shows the accuracy or classification on a set of testing samples (20 samples per radical). The y-axis in the second figure is scaled based on each learning curve to show if there is a peak (or multiple peaks).

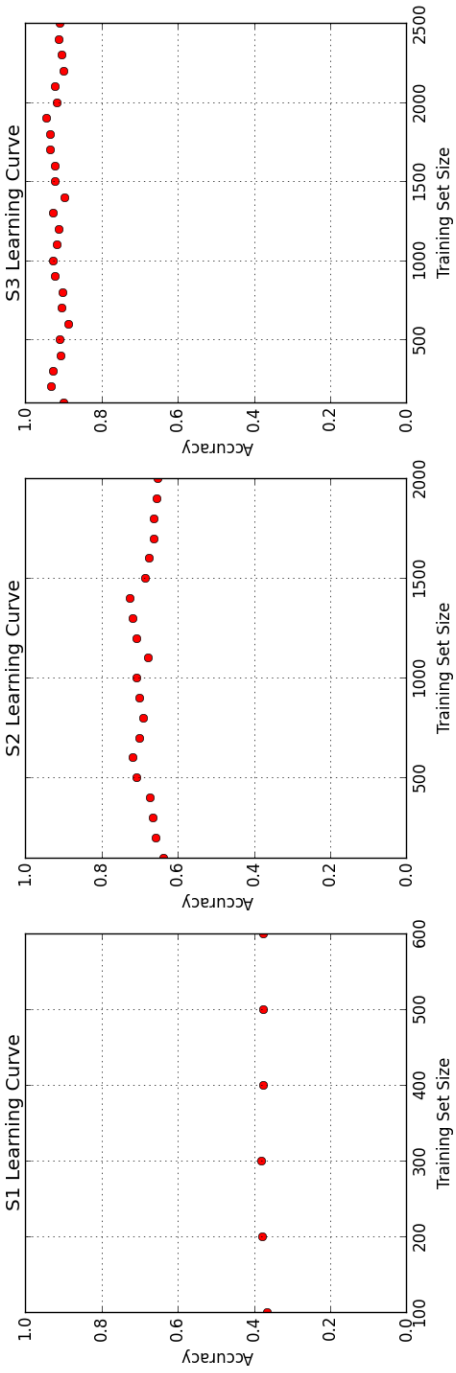

Figure C.0.1. Learning Curves (x-axis: Number of Training Samples; y-axis: Accuracy). In this figure, the range for the y-axis of each learning curve is the same: 0 to 1. Recall that the radical classes in subset S_1 have low character variability and all 3 subsets contain the same number of characters. This means S_1 contains more radical classes than S_2 and S_3, which explains the low accuracies. On the other hand, S_3 has the least amount of character classes, which explains the high accuracies.

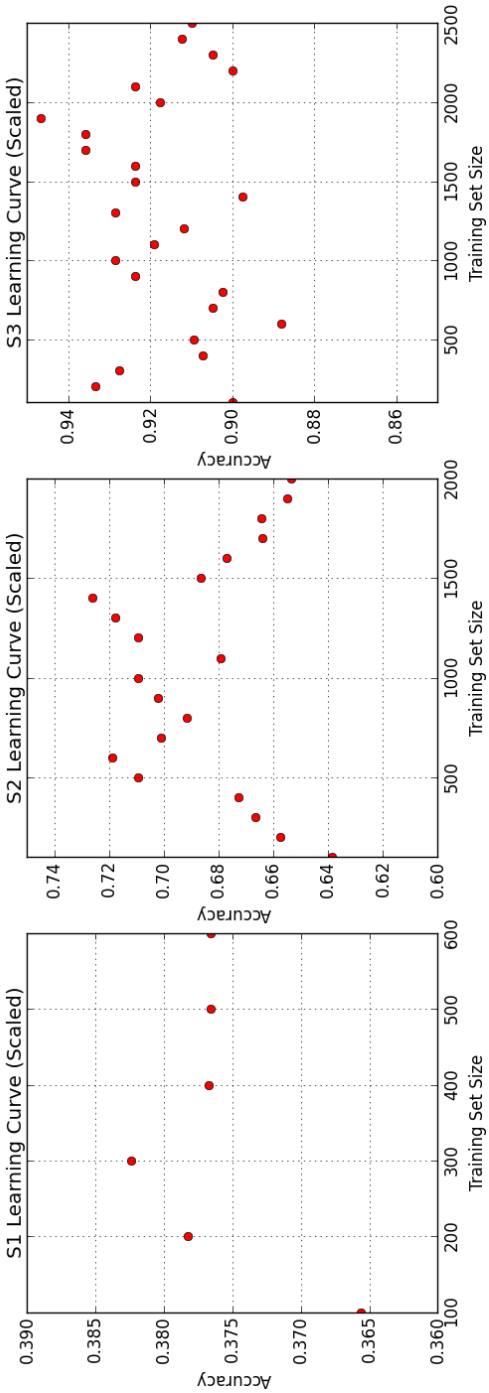

Figure C.0.2. Scaled Learning Curves (x-axis: Number of Training Samples; y-axis: Accuracy). The first scaled plot shows that the radical classes in subset S_1 are learned best when the training set size is $x = 300$. Recall that the radical classes in S_1 are the ones that have least character variability. The lack of variations of characters (which results in less noise in the training data) seems to have contributed to the fact that the learning curve only has one peak. The learning curve for S_3 shows a lot of noise (many peaks). The S_2 and S_3 learning curves are not minimally monotonous (too many peaks), but they show some peaks which can still suggest some good numbers to use for training. The highest peaks are $x \approx 1400$ and $x \approx 1900$ for S_2 and S_3, respectively.

Appendix D

Paired Comparisons of Means and Variances

The following plots compare the means and variances of individual radicals between two pairs of algorithms: standard NMF and SNMF (the two best algorithms), and standard NMF and penalized NMF (the best and the worst). Because we have to plot data for as many as 168 radical classes (too many to fit into one plot), each comparison is broken into two parts so that the first part contains radical classes 1 to 84, and the other one contains 85 to 168.

It is shown in Figure D.0.1 and Figure D.0.2 that although standard NMF and SNMF yield similar accuracies among all radicals, there is greater variance among the performance of each radical in SNMF. We assume this is because of uneven sparseness on the column vectors of W.

Furthermore, Figure D.0.3 and Figure D.0.4 demonstrate that although there is greater variance among the level of performance of individual radical in standard NMF, it yields much better results than the penalized method.

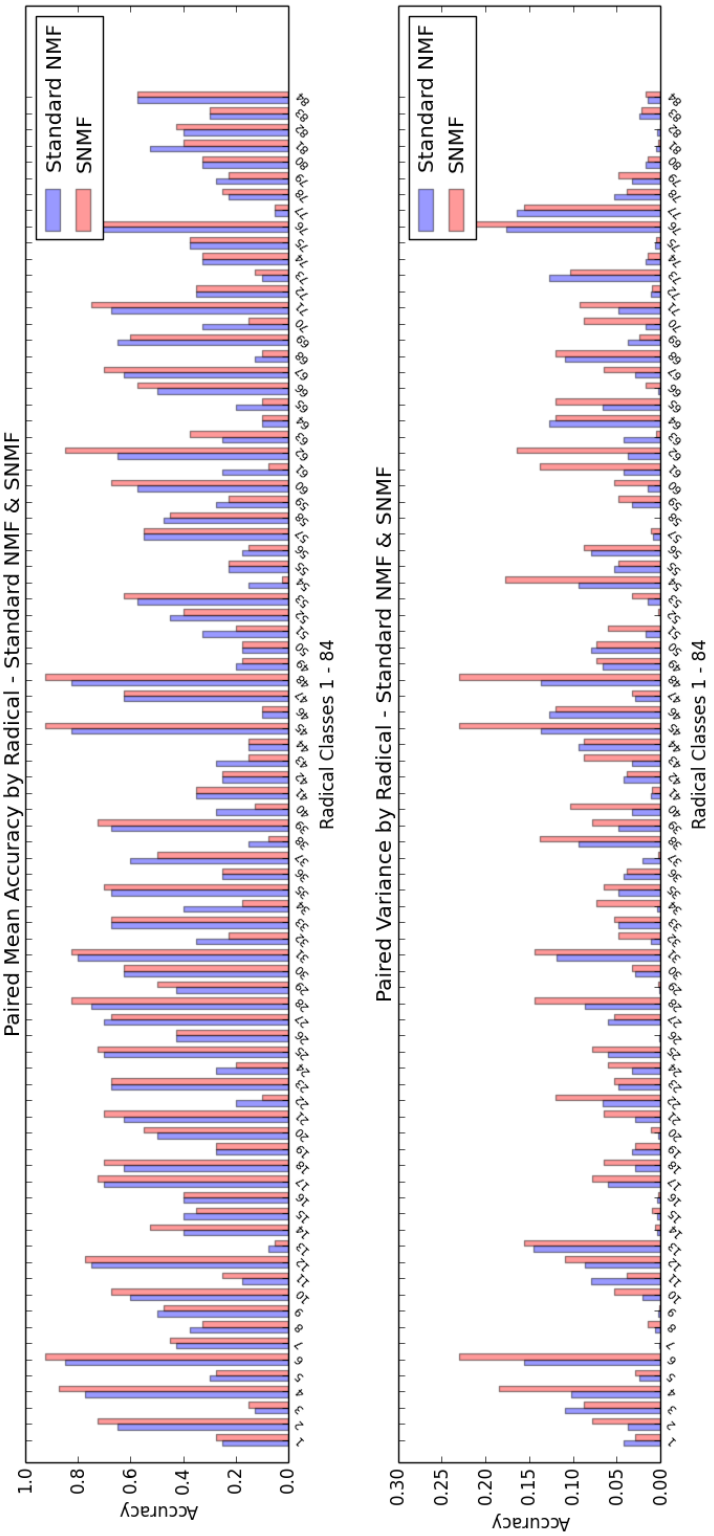

Figure D.0.1. Paired Mean Accuracy and Variance Comparison between Standard NMF and SNMF (First Half). The first part of the figure compares the mean accuracies of standard NMF and SNMF side by side. The second part of the figure compares the radical variances of standard NMF and SNMF side by side. The plots show that although the pairs of accuracies are similar, the paired variances actually vary by a great amount. We assume this is because of uneven sparseness on the column vectors of W learned by SNMF.

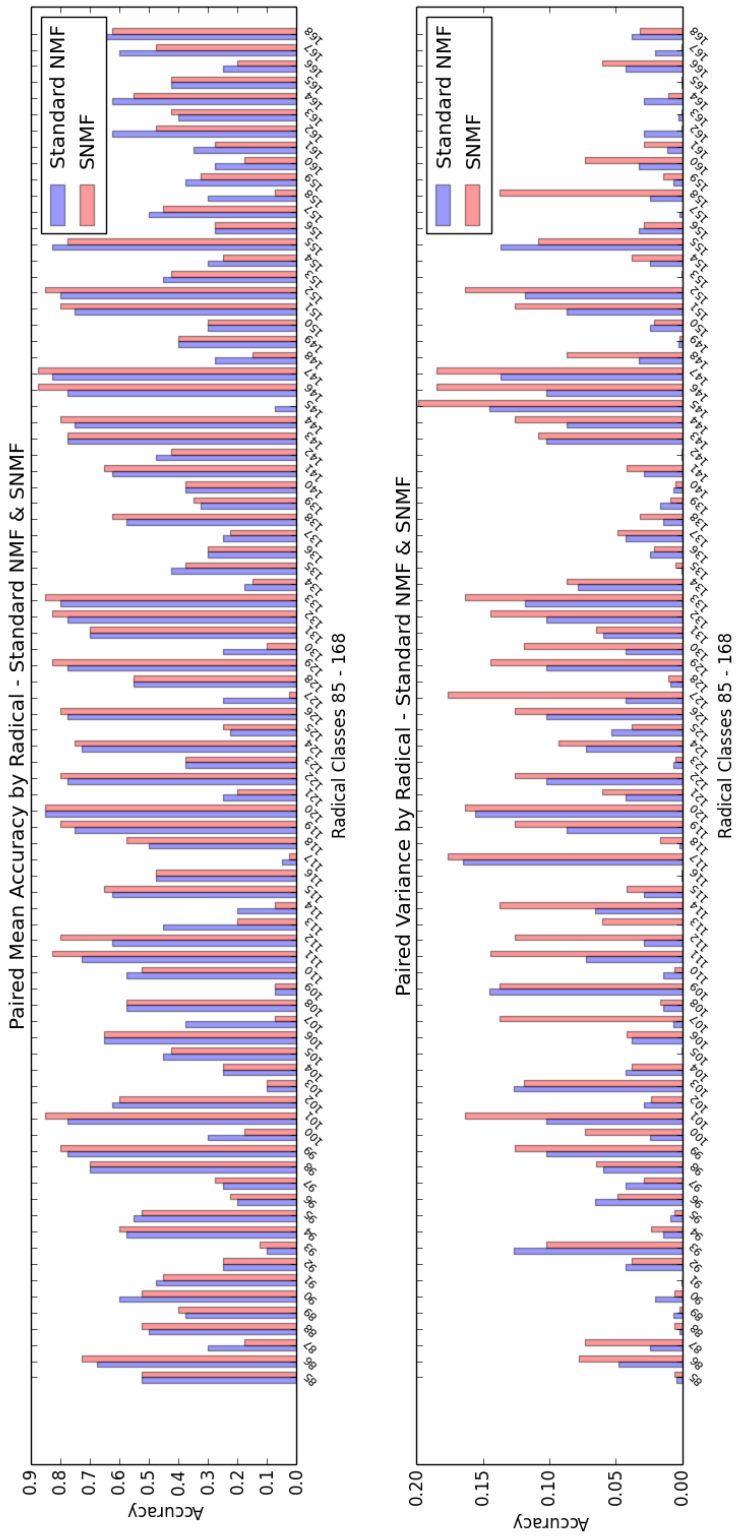

Figure D.0.2. Paired Mean Accuracy and Variance Comparison between Standard NMF and SNMF (Second Half). The first part of the figure compares the mean accuracies of standard NMF and SNMF side by side. The second part of the figure compares the radical variances of standard NMF and SNMF side by side. The plots show that although the pairs of accuracies are similar, the paired variances actually vary by a great amount. We assume this is because of uneven sparseness on the column vectors of W learned by SNMF.

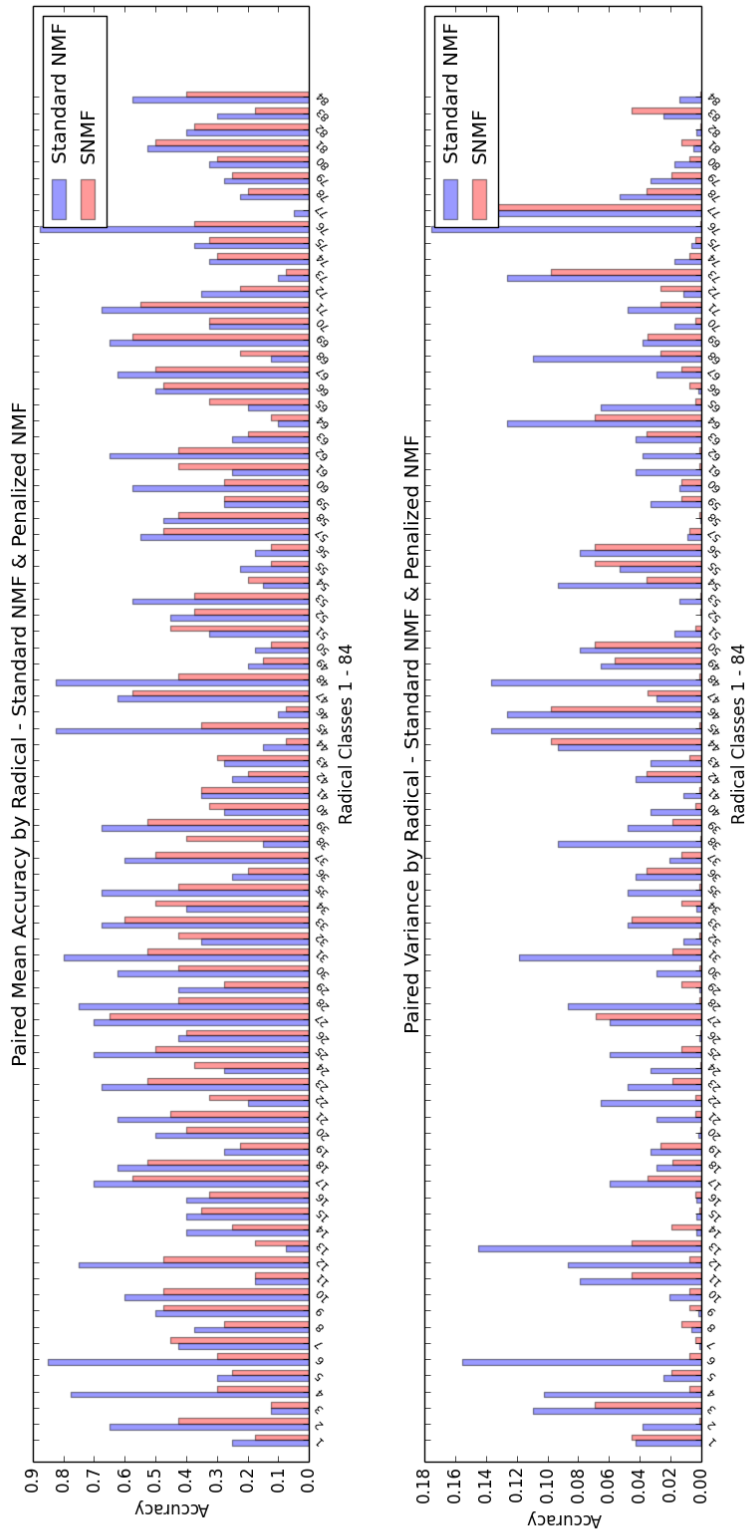

Figure D.0.3. Paired Mean Accuracy and Variance Comparison between Standard NMF and Penalized NMF (First Half). The first part of the figure compares the mean accuracies of standard NMF and penalized NMF side by side. The second part of the figure compares the radical variances of standard NMF and penalized NMF side by side. The plots show that although the pairs of accuracies are similar, the paired variances actually vary by a great amount.

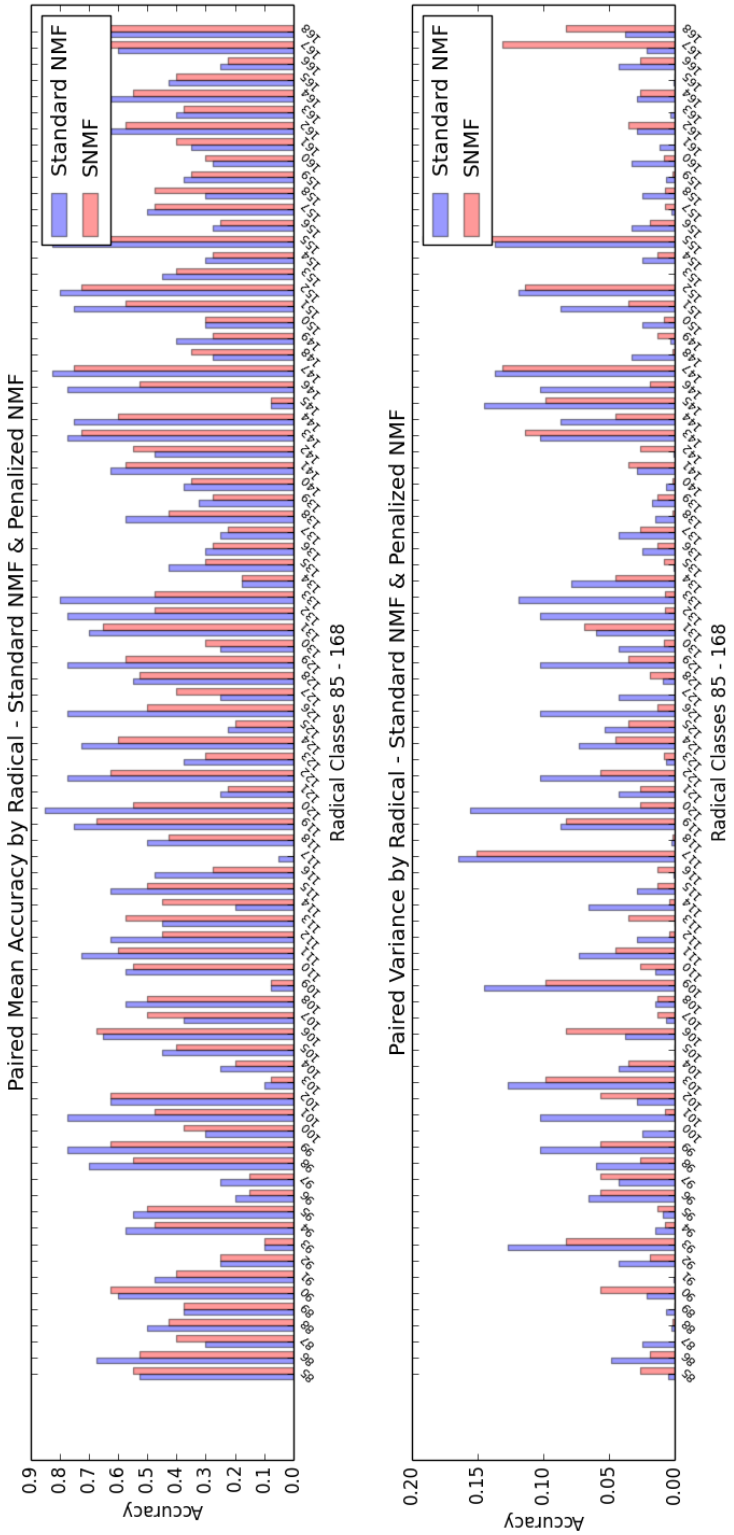

Figure D.0.4. Paired Mean Accuracy and Variance Comparison between Standard NMF and Penalized NMF (Second Half). The first part of the figure compares the mean accuracies of standard NMF and penalized NMF side by side. The second part of the figure compares the radical variances of standard NMF and penalized NMF side by side. The plots show that although the pairs of accuracies are similar, the paired variances actually vary by a great amount.

Appendix E
Python Code for Radical Classification

The following Python code demonstrates the usage of each of the 7 algorithms. The user must make sure that the data is grouped by radicals, so that there are 168 data folders. To run only one of the algorithms each time, the user should un-comment the rest of them. The training set size varies for each radical, which is based on the results of the learning curves. The testing data size is 20 for each of the 168 radicals. There is no intersection/overlap between the training and testing data. In the Python language, lines that start with '#' are comments and are therefore not executed. Similarly, paragraphs that are surrounded by triple apostrophes are comments for functions, which include expainations of function parameters and return values.

```
from __future__ import division
import random
import numpy as np
import numpy.linalg as la
import character_variability
import time
import nimfa
from numpy import linalg as LA

############## Initiation Stage & Adjustable Parameters ############

# countList = counts how many characters each radical maps to.
  # This is omitted in this file.
  # The user can regenerate this list from Appendix A.
countList = character_variability.count()

RAD_SET = np.arange(168) # Index of all radicals.

NUM_RAD = len(RAD_SET) # The total number of radical classes.

NUM_RUN = 10 # Number of runs. Default = 10.
```

```
# Number of samples used for training per radical.
  # Will be multiplied by a factor based on the learning curve later.
NUM_TRAINING = 102

# Number of samples used for testing / per radical, this number stays constant.
NUM_TESTING = 20

# Initiate an array of arrays to record the indices of the character samples.
    # used for training, so later on we can pick test samples that won't
    # overlap with the training samples.
picked = np.full((NUM_RAD,NUM_TRAINING*19), -1, dtype=np.int)

############## File-Reading Function ##############

def fileToVector(file):
  '''
  :param file: A character image file in its binary form.
  :return: The matrix in a vector form.
  '''
  data = open(file,'r')
  vector = [row.strip().split(' ') for row in data]
  data.close()
  vector = np.asarray(vector).astype(np.int) # Turning data into a numpy array
  vec = vector[0]
  return np.asarray(vec)

############## Basic Training Function ##############

def radicalTraining(index, r, num):
  '''
  This method is used for training individual radical column vectors.
    For the easy of computation, it will return a row vector instead.
    Later on we will convert it to the transpose of it.

  :param index: Index of the radical if the dictionary is not of size 168.
  :param r: Actual index of the radical.
  :param num: The character variability of the radical class.
  :return: The learned radical column and the Forbenius Norm error.
  '''

  # Multiply the number of training samples by a factor decided based on the
    # learning curves.
  if num <= 5: num_instances = num
  if num > 5 and num <= 42: num_instances  = 3
  if num >= 43 and num <= 78: num_instances  = 14
  if num >= 79: num_instances = 19

  # Initiate input data matrix of different samples.
    # Note that the dmat here is a transpose of V.
    # The actual data matrix V is of dimension 128^2 x NUM_TRAINING.
```

```
dmat = np.zeros([NUM_TRAINING * num_instances, 128 * 128])

# The set of samples we can pick for training.
# We use +1 because the actual data files are 1-based.
set = range(1, num * 122 + 1)

# Pick the training samples.
trainSet = random.sample(set, NUM_TRAINING * num_instances)

# Difference in length between picked array and the initial picked array.
diff = NUM_TRAINING * 19 - len(trainSet)
addon = np.asarray(np.full((1,diff), -1, dtype=np.int))[0]

# Update the picked array.
picked[index] = np.asarray(np.concatenate((trainSet, addon)))

# Add each image data (in its column vector form) into the V data matrix.
i = 0
for s in trainSet:
  testfile = "/user_specified_path/radical" + str(r+1) + "/" + str(s) + ".txt"
  vec = fileToVector(testfile)
  dmat[i] = vec
  i += 1

# Convert the transpose back into V.
dmat = np.transpose(dmat)

#### 1. Basic NMF
nmf = nimfa.Nmf(dmat, seed="nndsvd", rank=1, max_iter=50, update='euclidean',
  objective='fro')
nmf_fit = nmf()
W = nmf_fit.basis()
H = nmf_fit.coef()

#### 2. Probabilistic (PMF)
# pmf = nimfa.Pmf(dmat, seed="random_vcol", rank=1, max_iter=50, rel_error=1e-5)
# pmf_fit = pmf()
# W = pmf_fit.basis()
# H = pmf_fit.coef()

#### 3. Projected Gradient (LSNMF)
# lsnmf = nimfa.Lsnmf(dmat, seed="random_vcol", rank=1, max_iter=50, sub_iter=10,
  inner_sub_iter=10, beta=0.1)
# lsnmf_fit = lsnmf()
# W = lsnmf_fit.basis()
# H = lsnmf_fit.coef()

#### 4. Non-smooth (NSNMF)
# nsnmf = nimfa.Nsnmf(dmat, seed="nndsvd", rank=1, max_iter=50, theta=0.5)
# nsnmf_fit = nsnmf()
# W = nsnmf_fit.basis()
```

```
  # H = nsnmf_fit.coef()

  #### 5. Sparse NMF with Imposed Sparseness on W (SNMF)
  # snmf = nimfa.Snmf(dmat, seed="random_vcol", rank=1, max_iter=50, version='l',
    eta=1., beta=1e-4, i_conv=10, w_min_change=0)
  # snmf_fit = snmf()
  # W = snmf_fit.basis()
  # H = snmf_fit.coef()

  #### 6. Sparse NMF with Imposed Sparseness on H (SNMF)
  # snmf = nimfa.Snmf(dmat, seed="random_c", rank=1, max_iter=50, version='r',
    eta=1.,beta=1e-4, i_conv=10, w_min_change=0)
  # snmf_fit = snmf()
  # W = snmf_fit.basis()
  # H = snmf_fit.coef()

  #### 7. Penalized NMF (PMFCC)
  # pmfcc = nimfa.Pmfcc(dmat, seed="random_vcol", rank=1, max_iter=50,
    theta=np.random.rand(dmat.shape[1], dmat.shape[1]))
  # pmfcc_fit = pmfcc()
  # W = pmfcc_fit.basis()
  # H = pmfcc_fit.coef()

  # Calculate the Frobenius Norm error term
  WH = W.dot(H)
  D = np.subtract(dmat,WH)
  fro_norm = LA.norm(D, 'fro')

  return W, fro_norm

############## Basic Testing Function #############

def radicalTesting(index, rad, num, dict):
  '''
  Function that tests all 20 samples of a single radical class.

  :param index: Index of the radical if the dictionary is not of size 168.
  :param r: Actual index of the radical.
  :param num: The character variability of the radical class.
  :param dict: Learned dictionary of 168 radical classes.
  :return: The number of false classifications, and the total number of tests.
  '''

  err = 0 # Number of false predictions.
  total = 0 # Total number of tests.

  # Pool of samples to choose from.
  # We use +1 because the actual data files are 1-based.
  set = range(1, num * 122 + 1)

  # Find the array of samples that were not used in training.
```

```
  remainingSet = np.setdiff1d(set, picked[index])

  # Pick 20 samples that haven't been used.
  testSet = random.sample(remainingSet, NUM_TESTING)

  # Iterate over the testing samples.
  for s in testSet:
      testfile = "/user_specified_path/radical" + str(rad+1) + "/" + str(s) + ".txt"
      vec = fileToVector(testfile)

      # Approximate coefficient matrix using least squares.
      h = la.lstsq(dict,vec)[0]

      # Check to see if the correct radical class of the test image is in
        the top 5 coefficients in h.
      if np.argsort(np.asarray(h))[-1] != index
        and np.argsort(np.asarray(h))[-2] != index
            and np.argsort(np.asarray(h))[-3] != index
                and np.argsort(np.asarray(h))[-4] != index
                    and np.argsort(np.asarray(h))[-5] != index:
                        err += 1

      total += 1 # Increment the total number of testing samples.

  return err, total

################ Main Training Function #################

# Initiate the radical dictionary.
    # This is actually the transpose, for the ease of computation.
radDict = np.zeros([NUM_RAD, 128*128])

def train(dict):
  '''
  The following function uses the basic training function to generate
    the radical dictionary, W.

  :param dict: Initianization of the main dictionary W.
  :return: Learned dictionary, average training time, and training error.
  '''

  tolErr = 0 # total training err
  print "Start training..."
  start_training = time.time()
  for i,r in enumerate(RAD_SET):
    print "-- Training radical ", i, r
    W, err = radicalTraining(i, r, countList[r])
    dict[i] = np.transpose(W)
    tolErr += err
  end_training = time.time()
  print "End training..."
```

```
  avg = (end_training-start_training)/NUM_RAD
  print "Average training time : ", avg, " seconds / radical."
  dict = np.transpose(dict)
  print "Radical dictionary shape ", np.shape(dict)

  return dict, avg, tolErr/NUM_RAD

################ Main Testing Function ################

def test(dict):
  '''
  Calls the basic testing function to test on all 168 radical classes.

  :param dict: Learned W dictionary.
  :return: Overall accuracy,
           average testing time,
           and the array of 168 individual accuracies.
  '''

  accRadicals = np.zeros([1,NUM_RAD])[0] # Accurary for each of the radicals.

  t_err = 0
  t_total = 0

  start_testing = time.time()
  for i,r in enumerate(RAD_SET):
    print "testing -> ", i,r
    num = countList[r] # number of instances
    err, total= radicalTesting(i,r,num,dict)
    t_err += err
    t_total += total
    accRadicals[i] += (total-err)/total

  end_testing = time.time()
  avg = (end_testing-start_testing)/(NUM_RAD*NUM_TESTING)
  print "Average testing time : ", avg, " seconds / character."

  for j in range(len(accRadicals)): # Accuracy for each radical
    print "radical, ", j, accRadicals[j]

  return (t_total-t_err)/t_total, avg, accRadicals


################ Main Function ################

def main(num_run=10):
  '''
  Main function that calls all sub-functions to calculate final measurements.

  :param num_run: Number of runs. Default = 10.
  :return: None. All results are printed.
```

```
  '''

  tp = 0 # Total number of true positives.
  train_time = 0
  train_err = 0
  test_time = 0

  # Initiate an array of individual radical class accuracies.
  acc_rad = np.zeros([1,NUM_RAD])[0]

  # Run the experiment several times.
  for i in range(num_run):
    dict_init = np.zeros([NUM_RAD,128*128])
    radDict, avg_train_time, avg_train_err = train(dict_init)
    accu, avg_test_time, accRadicals = test(radDict)
    tp += accu/num_run
    train_time += avg_train_time/num_run
    train_err += avg_train_err/num_run
    test_time += avg_test_time/num_run
    acc_rad = np.add(acc_rad, accRadicals/num_run)

  print "############ Final Results ############"
  print "true positive ", tp
  print "avg training time ", train_time
  print "avg train err ", train_err
  print "avg testing time ", test_time
  print "accuracy for each radical class", acc_rad

main(NUM_RUN)
```

Bibliography

[1] *CASIA Online and Offline Chinese Handwriting Databases*, http://www.nlpr.ia.ac.cn/databases/handwriting/Home.html.

[2] *Chinese Character Sets-China: GB 2312-80 Level 2 Radical Index*, http://examples.oreilly.com/9780596514471/cjkvip2e-appG.pdf.

[3] C. Lee, H. Kang, and Kim H, *Font Classification Using NMF With Hierarchical Clustering*, International Journal of Pattern Recognition, 2005.

[4] Qingcai Chen, *Harbin Institute of Technology Opening Recognition Corpus for Chinese Characters (HIT-OR3C)*, http://www.iapr-tc11.org/mediawiki/index.php/Harbin_Institute_of_Technology_Opening_Recognition_Corpus_for_Chinese_Characters_(HIT-OR3C).

[5] *Frobenius Norm*, http://mathworld.wolfram.com/FrobeniusNorm.html.

[6] *Introduction to SIFT (Scale-Invariant Feature Transform)*, http://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html#gsc.tab=0.

[7] Hyunsoo Kim and Haesun Park, *Sparse Non-Negative Matrix Factorizations via Alternating Non-Negativity-Constrained Least Squares for Microarray Data Analysis* (2007), 1496–1497.

[8] Daniel Lee and Sebastian Seung, *Algorithms for Non-negative Matrix Factorization.*

[9] ———, *Learning the Parts of Objects by Non-negative Matrix Factorization*, Nature **401(6755)** (1999), 788-91.

[10] *Line Dictionary*, http://ce.linedict.com/dict.html#/cnen/home.

[11] L. Shan, *Passport to Chinese: 100 Most Commonly Used Chinses Characters*, EPB Publishers, 1995.

[12] *Nonnegative Matrix Factorization*, http://www.almoststochastic.com/2013/06/nonnegative-matrix-factorization.html.

[13] Daming Shi, *Offline Handwritten Chinese Character Recognition by Radical Decomposition*, ACM Transactions on Asian Language Information Processing **2** (March 2003), 27–48.

[14] Rosie Shier, *Statistics: 1.1 Paired t-Tests*, `http://www.statstutor.ac.uk/resources/uploaded/paired-t-test.pdf`.

[15] T. Jun, X. Xie, W. Zheng, and J. Lai, *Radical Extraction Using Affine Sparse Matrix Factorization For Printed Chinese Characters Recognition*, International Journal of Pattern Recognition and Artificial Intelligence, 2012.

[16] T. Zheng, Fang B, Liu W, Y. Tang, G. He, and J. Wen, *Total Variation Norm-based Non-negative Matrix Factorization for Identifying Discriminant Representation of Image Patterns*, Elsevier, 2008.

[17] Fei Wang, Li Tao, and Zhang Changshui, *Semi-Supervised Clustering Via Matrix Factorization* (January, 2008).

[18] W. Zheng, Li S, Lai J, and S. Liao, *On Constrained Sparse Matrix Factorization*, Proceedings/IEEE International Conference on Computer Vision., 2007.

[19] Albert Au Yeung, *Matrix Factorization: A Simple Tutorial and Implementation in Python*, `http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/`.

[20] Marinka Zitnik, *Nimfa*, `http://nimfa.biolab.si/index.html`.