Recognition of Handwritten Names

Dafna Bitton
Department of Computer Science
University of California, San Diego
dbitton@ucsd.edu

Abstract

Optical character recognition (OCR) is an important problem that has many applications. It has been widely studied and in many cases solved. We propose an offline method using a Hidden Markov Model (HMM) to take in as input an image of a handwritten name and output the corresponding name from a given lexicon.

1 Introduction

Class sizes at universities can be in the hundreds. The process of returning graded work to students can be time consuming and tedious. In order to avoid this process, we propose a method to return work to students online. This process provides feedback to students when it is available to them and avoids manual return of papers in class. The course staff will need to scan in all of the graded homework and tests, which can be automated, and label each document with the corresponding students' name. The laborious part of this process is labeling the documents with the students' names. We propose a method to automate this process by performing OCR on the name and using an HMM to determine which document belongs to which student.
Our idea originated from Jaety Edwards and David Forsyth [1] who proposed a method to use an HMM to transcribe a handwritten script. Their work focuses on finding where the characters are segmented and then transcribing the letters, using minimal training data.

2 Our Method

In order to simplify the process of finding and isolating the characters from the students' names, we provide character boxes for the students to enter their names, with one character for each box. Figure 1 is what a cropped input image looks like.

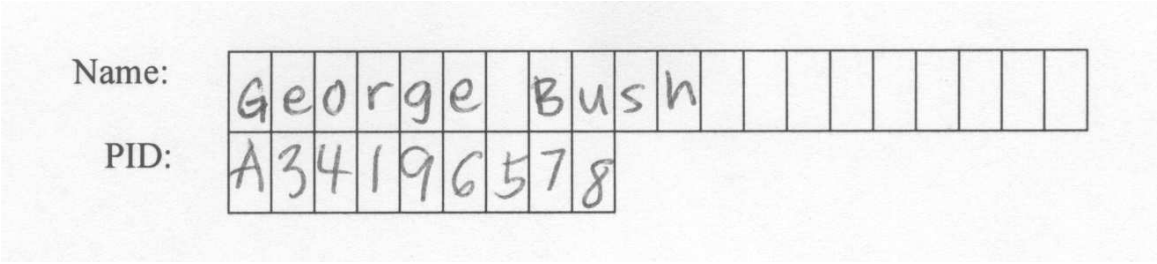

Figure 1: An example of a cropped quiz

If the length of a student's name exceeds the amount of character boxes available, we will just work with what can fit in the character boxes provided. We provide 20 character boxes and hypothesize

that the first 20 characters are distinct enough to identify the students.

Since we introduce character boxes, we need a mechanism for finding them. We run a line detection algorithm, called the Hough transform, on the input image where we expect the name to be, and infer where the character boxes are. We use prior knowledge that the lines from the character boxes are either horizontal or vertical and take the following steps:

1. Take the gradient of the input image.
2. Run a variation of Canny edge detection (edge.m in Matlab) on the vertical gradient of the image.
3. Run the Hough transform on the result of the edge detection and find the top two lines detected. From this infer where the horizontal character box lines are.
4. Run a variation of Canny edge detection (edge.m in Matlab) on the horizontal gradient of the image.
5. Run the Hough transform on the result of the edge detection and find the top lines detected. Infer where the 21 vertical lines of the character boxes are.
6. Cut out each character.

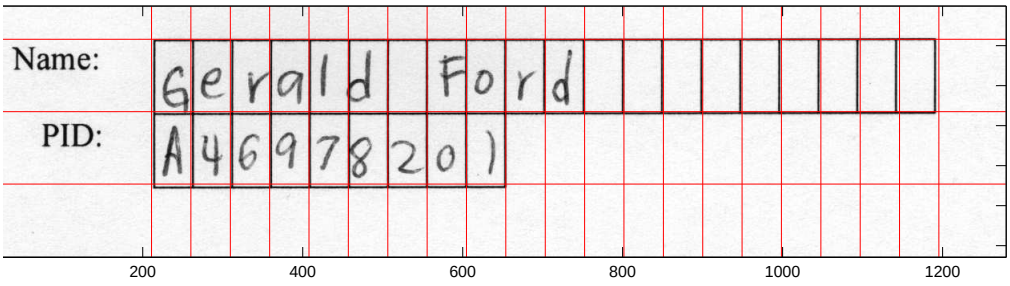

Figure 2: An example of the character box lines that were detected

The red lines in Figure 2 are the lines that were detected as a result of the steps taken above. Since the detected character box lines are often not exact, as in Figure 2, and the character box lines have a width, the cut out characters often have extra junk around the edges that needs to be removed.

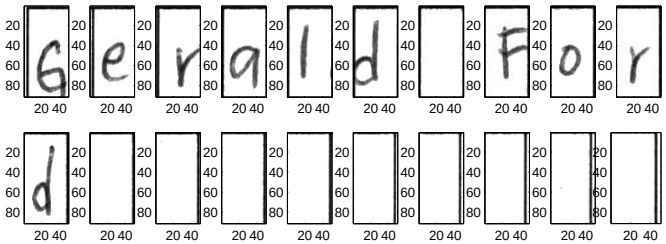

Figure 3: An example of extra junk around the edges of the cut out characters

In order to remove these extra lines, we expand a rectangle from the center of the character image until an optimal image is reached. We then center and scale the character image.

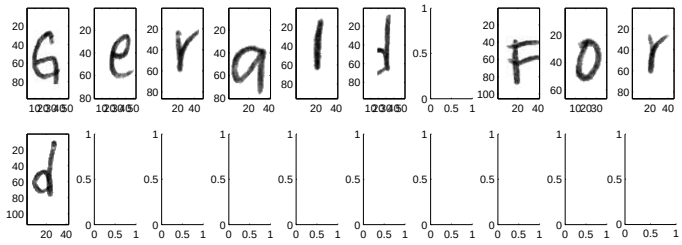

Figure 4: The character boxes after the extra lines have been removed

We used the ABCDETC dataset from NEC Labs as training data. NEC Labs had individuals fill out sheets with space for 5 examples of each character. We used 18 such sheets; in total we had 90 examples of each character a-z. We again used the Hough Transform to find the character boxes and cut out each training letter.

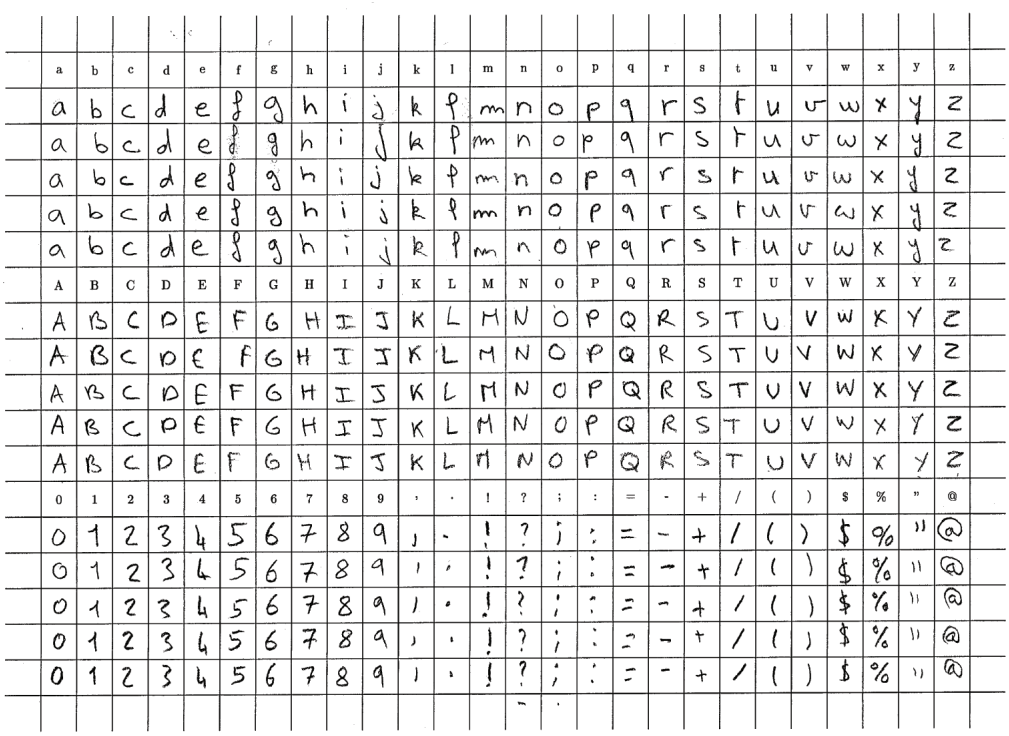

Figure 5: An example of a completed sheet from NEC Labs

3 The Model

We used an HMM where the hidden states were the true ASCII representation of the characters and the observations were the images of the characters. The transition probabilities we calculated from the class roster using a bi-gram model with the following equation:

$$P(c_t|c_{t-1}) = \frac{\text{Count}(c_{t-1}, c_t)}{\text{Count}(c_{t-1})}$$

Where Count(c_t) is the number of times character c_t appears in the roster and Count(c_{t-1}, c_t) is the number of times the sequence c_{t-1}, c_t appears in the roster.

The emission probabilities are supposed to be the probability of a character looking a certain way. For example, the probability of seeing some character image given that you know it is of an 'a'. Instead of calculating this, for each character image we find the top 10 nearest neighbors from the training data. We then say that the probability that this character image is a particular character is the number of votes for this character + 1 divided by 10 + 26. We add 1 to each vote tally because the nearest neighbor algorithm in this case is very flawed and cannot be trusted. We don't want a character to not be considered just because it had no votes.

With this framework in place we run the Viterbi algorithm to find the most likely sequence of hidden states that produced the observations (character images). We then take this sequence of characters we believe to be the truthful name and run nearest neighbor on it against all of the names in the class roster. This way we guarantee that our guess of the name is actually in the roster. We also require that the name that is returned as the nearest neighbor have the same length as the query name.

4 Experiments

We had a test set with 12 different names, where each name was between length 9 and length 14, inclusive. 10 of the 12 names were classified correctly using our algorithm. The results would not have been quite as good had all of the names in the test set been the same length. This is because in the final step where we run nearest neighbor on the predicted name against the names in the roster, we only consider names of the same length. Therefore, if there is a name in the roster that has unique length, then it will always be classified correctly regardless of what happens in the OCR step. In order to have a better idea of how the algorithm is performing, we need to have a larger test set with more names of the same length.

5 Future Work

The next step in this project is to improve the emission probabilities. The outcome of the nearest neighbor step is just slightly better than random chance, and this is not acceptable. Even though the training data has been centered and scaled, it is not enough. We need to both jitter the training data and get a lot more training data. There should be enough training data to cover most representations of each character, but this is certainly not the case in our current training data set. Another option is to not use nearest neighbor anymore but rather go with something like boosting. Furthermore, the HMM model should be reconsidered. It is not enough to only take into account the previous state. An HMM may still be appropriate, if we expand it to be an n-gram HMM, where we take into account the last n-1 states, instead of only the previous state like we do now.

References

[1] J. Edwards, D. Forsyth. *Searching for Character Models*. NIPS 2005.