Recognizing Patient Names in Handwritten Clinical Notes in the Absence of Training Data

Bethany Percha
Stanford University
Biomedical Informatics Training Program
318 Campus Drive, Clark Center S240
Stanford, CA 94305
blpercha@stanford.edu

Abstract

As physicians and hospitals in the United States switch to electronic forms of record keeping, one of the greatest challenges they face is how to deal with the thousands of handwritten clinical notes already in their possession. Currently the best way to incorporate these notes into electronic databases is to convert them into digital images via scan or fax. However, staff must then manually curate these pages by attaching them to the correct patient files. Optical character recognition of handwritten text images like these fails in most cases because of the variety of handwriting styles involved and the complete lack of labeled, handwritten training data. Here we investigate an approach that addresses this problem by manufacturing pseudo-handwritten training data from multiple computer fonts, which is used to train hidden Markov models for each of 32 different possible patient names. The models are then used to classify handwritten word images of patient names. The handwritten names are classified correctly 64% of the time using this method. We discuss future improvements to the method, as well as the practical details of its implementation within an EMR system.

Future Distribution Permission

1. Introduction

Debate about the future of the U.S. health care system currently pervades the halls of politics and the front pages of major newspapers. One thing almost everyone can agree on, however, is that medicine in the United States will benefit from the judicious use of electronic forms of patient record-keeping, such as electronic medical record (EMR) and prescription ordering systems. With the passage of 2009's HI-TECH Act, physicians can receive up to $40,000 as an incentive for using EMR systems that meet certain criteria, and the use of these systems is on the rise. However, there are still some major roadblocks to their universal adoption.

One of the greatest challenges health providers face when switching to an EMR is how to deal with the thousands of handwritten clinical notes already in their possession. All of these records must somehow be introduced into the system, either by scanning or manual data entry. Currently the easiest way to do this is simply to fax in all of the old records. Staff must then manually assign individual documents to the appropriate patient files. Optical character recognition (OCR) usually fails on handwritten documents because these algorithms typically need to be trained to recognize a particular person's handwriting using a large volume of handwritten text. In the clinical setting, such training data is unavailable, and patient documents may also contain handwriting from many different individuals.

For my CS231A final project, I decided to focus on the problem of how to build a classifier for handwritten patient names, under the assumption that absolutely *no* handwritten training data was available. This is not an unrealistic assumption in the clinical setting, for reasons I discuss below. I developed the classifier using "morphed" computer fonts and hidden Markov models. In this paper, I discuss the construction of the classifier, its accuracy (surprisingly good) when classifying handwritten patient names, and how it might be modified in the future to improve its performance.

2. Background

2.1. Keyword Spotting

Recognizing handwritten patient names represents a subdomain of a larger computer vision problem called *keyword spotting* [1]. Keyword spotting differs from a related field, handwriting recognition, because it does not attempt to translate entire documents into digital blocks of text. It only seeks to determine whether a certain word or phrase is *present* in a document. Most keyword spotting algorithms treat keywords as images and find a set of words in the document whose appearance is most similar to the query word.

There are two main variants of this word spotting technique, which Rodriguez and Perronnin outline in their 2009 paper [2]. The *holistic* approach describes an entire word image with a single feature vector and defines a vector distance measure to quantify word similarity [1, 3, 4]. The *part-based* approach describes a word image as a set of local points of interest, or keypoints, which are then mapped onto keypoints from another image. The amount of deformation it takes to map the keypoints provides the similarity measure. They keypoints can be defined by a variety of classic image recognition metrics such as gradient angles or corner positions and then matched using an elastic distance measure similar to those used in part-based deformable models [5, 6]. They can also consist of word profiles [7, 8], contours [9], or other relevant features.

For many years, the state-of-the-art algorithm in this field was Dynamic Time Warping (DTW) [7]. More recently, other algorithms have been found that outperform DTW, both in terms of classification accuracy and computational efficiency [10]. The most successful of these have used hidden Markov models (HMMs) [2, 11, 12]. Importantly, HMM-based word spotting algorithms outperform DTW and other vector distance metrics on handwritten corpuses composed of text from multiple writers, which is exactly the problem we face with clinical notes.

However, even within the set of algorithms that use HMMs, considerable variation exists. This is because the optimal choice of featurization technique and HMM design varies depending on one's particular application. For example, many keyword spotting approaches have sought to train individual models for handwritten characters, which are then concatenated into a model for a keyword of interest [13, 14, 15]. These methods have the advantage that they only need to train 26 different character models, as opposed to thousands of different word-based models. However, they often choke on cursive text, which is ubiquitous in clinical notes [10, 16]. In general, character-based HMMs are powerful in cases where one must be able to search for any keyword (such as when one wishes to index a database of handwritten historical documents and then allow users to search for words of interest) but they are less powerful than word-based models when one only wishes to search for a limited number of keywords of interest, as in the case of patient names. For that reason, I chose to use word-based HMMs in this project.

2.2. Clinical Text

Clinical text presents a particularly challenging environment for keyword spotting for three reasons.

1. Traditional keyword spotting algorithms involving DTW or HMMs usually need to be trained with a huge amount of labeled, handwritten words and sentences. No such labeled data exists for our purposes, and it is unlikely that in the health care environment, anyone would be willing to create such a resource. In addition, many patient names are unusual and would not be found in a large general corpus

of handwritten text.

2. The performance of virtually all keyword spotting algorithms decreases dramatically when documents from more than a few authors are involved. Handwritten patient names found in clinical notes come from a variety of physician practices and hospitals, and a single patient chart can include handwritten text from dozens or even hundreds of individuals.

3. Most patients do not go to the doctor very often. As a result, new faxed documents will often include names that are on the practice's patient list, but have never been seen before in handwritten form.

However, clinical documents also have certain features that make them particularly tractable for keyword spotting:

1. A clinical document is guaranteed to contain exactly one patient name. Both the first and last name are always included.

2. Clinical documents are often short, containing fewer than 100 words. Often the patient name is one of only a few handwritten words on the document (as with patient information forms or questionnaires, for example). This reduces our chances of false positive results and makes the process of word segmentation easier.

3. In contrast to scribbled doctor's notes and prescriptions, patient names are usually written on these forms by administrative staff, and tend to be relatively neat.

Finally, we do not need to reach a very high level of success for this method to prove a worthwhile addition to EMR software. Administrative staff are currently doing 100% of this work on their own, and there is no penalty if we suggest an incorrect name. Accuracy of even 40-50% or so would still save staff a considerable amount of time, effort, and typing.

2.3. A Note on Word Segmentation

The initial challenge of any keyword spotting algorithm is word segmentation. However, due to this project's limited timeframe and based on the advice of Prof. Li, I elected to work with a pre-segmented corpus of word images. I believe that the word segmentation process may be a bit easier for clinical notes than for other types of handwritten text for the reasons outlined in Section 2.2. In addition, high-quality word segmentation algorithms exist that could be applied to these documents [17, 18, 19]. Other segmentation-free, text-line-based word matching approaches might also work for these data [5, 8, 20]. But more importantly, I felt that the total lack of handwritten training data was a more serious problem that needed to be addressed first for the method to have any chance of success.

3. Approach

3.1. Training Set

The biggest challenge in this project was the complete lack of handwritten training data. Whatever training data I needed had to be generated automatically by a computer. I reasoned that the best approach to this problem would be to use a variety of different fonts to represent each name. The initial training data for each of the 32 patient names, therefore, consisted of the same name repeated in 40 different fonts (Figure 1). However, 40 repetitions of a name is not sufficient to build a hidden Markov model with more than a few states.

My solution to this problem was to "morph" each of the 40 different name representations slightly several times using randomly generated sinusoidal transforms. My reasoning was that if a human being tried to represent a name shown in a particular font, the name would look similar to the font except for random sinusoidal motions caused by the wobbling of the person's hand. The results of this "morphing" operation don't look exactly like handwriting, but they serve their purpose: to artificially create more training data with more variation than the original set of typed data (Figure 2). Using this technique, I was able to

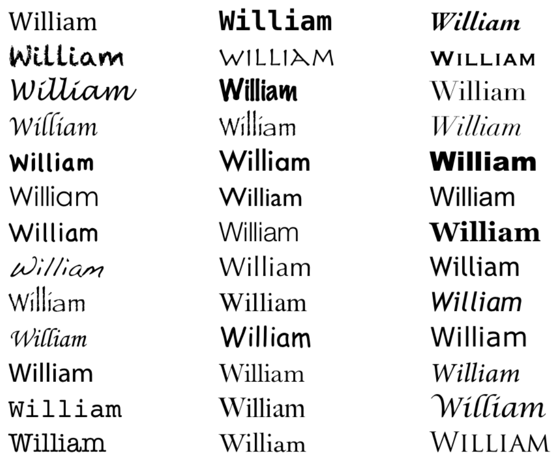

Figure 1. The 40 fonts used to train the classifier.

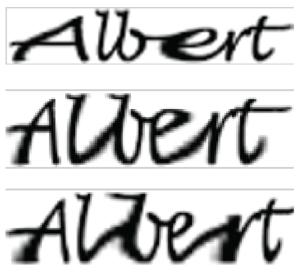

Figure 2. Three “morphed” versions of the name “Albert”, originally represented using the font Lucida Handwriting. They are similar to the original and completely recognizable to a human as “Albert”, but are variable enough to constitute unique training examples.

generate 800 training examples for each name, as opposed to the original 40.

3.2. Local Gradient Histogram (LGH) Features

After generating the training set, the next step was to extract features from each training example that would allow me to build a unique HMM for each name. When building a HMM for word recognition, one can use a potentially limitless number of different features for classification. One popular set of features frequently employed in this process was originally proposed by Marti and Bunke [21] in their 2001 paper that focused on indexing historical documents. These nine features are obtained by moving across an image with a one-pixel-wide window and calculating (1) the number of black pixels in that column, (2) the center of gravity of the column, (3) its second order moment, (4) the upper contour, (5) the lower contour, (6) the vertical gradient at the position of the upper contour, (7) the gradient at the lower contour, (8) the number of black-white transitions in the column, and (9) the number of foreground (black) pixels between the upper and lower contours. In my early experiments, I used these features, but they did not perform as well as the Local Gradient Histogram (LGH) features, which I will now describe.

The LGH features were originally proposed by Rodriguez and Perronnin [22] and were inspired by David Lowe’s SIFT features [23]. Rodriguez and Perronnin describe the process again in their 2009 article [2] and provide a useful pictorial illustration of it there. The steps are as follows:

1. Apply a Gaussian filter to the word image $I(x, y)$ to obtain a smoothed image $L(x, y)$.

2. Compute the horizontal and vertical gradient components G_x and G_y as:

$$G_x(x, y) = L(x + 1, y) - L(x - 1, y)$$

   and

$$G_y(x, y) = L(x, y + 1) - L(x, y - 1).$$

3. Compute the magnitude and direction of the gradient at each pixel as:

$$m(x, y) = \sqrt{G_x^2 + G_y^2}$$

   and
$$\theta(x, y) = atan2(G_y, G_x)$$

   where $atan2$ is a Matlab function that returns the direction of the (G_x, G_y) vector in the range $[-\pi, \pi]$.

4. Move a sliding window of fixed width across the image from left to right, obtaining a sequence of overlapping images of sub-word parts. At each position:

    - Subdivide this window into a 4×4 grid of cells.

- From all the pixels in each cell, construct a histogram of gradient orientations, considering only 8 possible angular orientations. Each pixel contributes to the closest bin with an amount $m(x, y)$.
- Concatenate the 4×4 histograms of 8 bins each into a 128-dimensional feature vector.

5. Scale each of the feature vectors to have unit norm.
6. The complete name is now represented as a set of 128-dimensional feature vectors of unit length.

When generating these features, I used a fixed window width of 20 pixels (I tried several different window widths and this performed the best). The initial Gaussian filter I applied to the images had a size of 10 pixels and used $\sigma = 2.0$. Again, I tried several filter sizes and strengths and this one led to the best overall performance.

3.3. Hidden Markov Models

Hidden Markov models (HMM) work well in handwriting recognition primarily because of their ability to handle sequences of variable length [2, 11]. HMMs can be used to represent entire words, or they can be used to represent subword entities such as characters. For the purposes of patient name spotting, I chose to build word-level HMMs, since the number of patient names in a given medical practice is limited to several hundred, and a complete list is always known in advance. My procedure for building each word HMM was as follows:

1. Obtain 800 "morphed font" training examples of the name.
2. For each "morphed font" training example:
   - Find the upper and lower baseline (described in Section 3.5).
   - Remove silence from the image as described in Section 3.8.

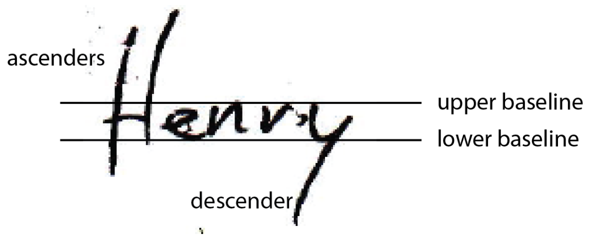

Figure 3. The anatomy of an English word. The upper and lower baselines divide the central portion of the word, which is usually of fairly uniform height, from the ascenders and descenders, which are not.

   - Resize the image. I chose a uniform bounding box for each word that put exactly 20 pixels between the upper and lower baselines (see Section 3.5 and Figure 3) and was 100 pixels wide. Normalizing the height to the inter-baseline distance instead of the overall word height allowed me to control for the highly variable size of ascenders and descenders.
   - Obtain the LGH features for the image as described in Section 3.2.
3. Use the 800 different morphed font training images to construct a word HMM for the name using Matlab's PMTK toolkit [24]. The best-performing models contained 10 states each. Each state produces emissions from a single 128-dimensional Gaussian. The Baum-Welch algorithm [25] is used to train the HMMs.

3.4. Test Set

To create a handwritten test set on which to test the word HMMs, I began by obtaining a set of 32 patient first names from original clinical documents. Due to patient privacy issues, I was not able to use any other words from the original clinical documents; in particular, I was not able to use both the patient's first and last names, as this would have been a clear violation of patient confidentiality.

I obtained 27 additional copies of each patient first name from friends and colleagues (a fairly random sample of male and female, young and old, physician and non-physician). Those 28 copies (the original plus the 27 copied versions) served as the test set for this project. The final test set therefore contained 28×32 different handwritten names.

3.5. Baseline Detection

For the purposes of both training and testing the HMMs, baseline detection was important. To detect the baselines (Figure 3), I used the method outlined by Blumenstein and colleagues [26], which was adapted from [27]:

1. The row containing the largest run of black pixels in the word image is labeled as the "peak line".
2. The average number of foreground (black) pixels in each row is calculated and a vertical histogram of foreground pixels in each row is obtained.
3. The row containing the minimum number of foreground pixels *prior* to the peak line is located and marked. Commencing from the marked row, we find the first row containing a number of foreground pixels greater than or equal to the average number. This line is marked as the **upper baseline**.
4. Repeat step 4, but start at the row with the minimum number of foreground pixels *after* the peak line and move toward the peak line. The first row containing a number of foreground pixels greater than or equal to the average number is marked as the **lower baseline**.

3.6. Skew Correction

Figure 4 shows an example of a word that suffers from baseline skew, where the word is written at an angle relative to the bottom edge of the image box. To fix this problem, I used the method outlined in [26]:

1. Detect the upper and lower baselines and remove the ascenders and descenders.

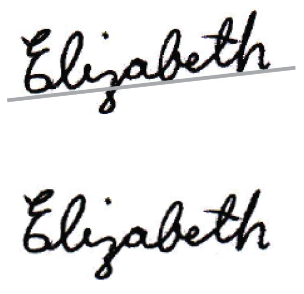

Figure 4. A name that suffers from baseline skew. The adjusted version is beneath the original.

2. Locate the center of the word image and divide the image into left and right halves.
3. Calculate the center of mass of each component, (x_L, y_L) and (x_R, y_R).
4. Calculate the word's slope as

$$\text{slope} = \frac{y_R - y_L}{x_R - x_L}.$$

5. Find angle of skew as

$$\theta = \tan^{-1}(\text{slope}).$$

6. Rotate the image about the image center by θ to correct the skew.

3.7. Slant Correction

Handwritten words also tend to be slanted relative to the vertical, which is a particular problem for our application because the training data are generated using fonts which are mainly not slanted (though a few are). I used a modified version of the method from Vinciarelli *et al* [28] to correct the slant of each word in the test set. This method assumes that a word is deslanted when most of the vertical lines in the word are contained in single columns. It works as follows:

1. For each angle α in an interval $[-\pi/4, \pi/4]$, apply a shear transform to the image.
2. Calculate the average vertical stroke thickness using the following technique [26]:

- Starting at the top of each column in the word image and proceeding to the bottom, sum the number of foreground pixels contained in the last continuous run.
- Sum all of the runs from the previous step and calculate the average.

3. The angle, α, yielding the highest value for the average stroke thickness is taken as the slant estimate.

3.8. Silence Removal

Since applying the sinusoidal transformations to typed words often leads to large regions of white pixels in between different letters, and since handwritten words often contain regions of white space of variable width in between the letters, I applied a common technique from speech recognition (in both the training and testing phases) and removed these "silent" portions. Silence removal simply means progressing horizontally through the image with a box one pixel wide and removing those columns that contain only background (white) pixels.

3.9. Resizing

As in the training procedure described in Section 3.3, we resize the test images to a uniform width and uniform inter-baseline height. The uniform bounding box puts exactly 20 pixels between the upper and lower baselines and is 100 pixels wide.

3.10. Performance Evaluation

I evaluated the performance of the final model in two ways:

1. **Training error.** I used each word HMM to reclassify each of the original typed words. This was mostly just a sanity check to make sure I hadn't morphed the words so far with the sinusoidal transforms and Gaussian blurring that the original training examples were unrecognizable.
2. **Test error.** I tested each model on the test set of 28 handwritten examples $\times$ 32 different patient names. I calculated the posterior probability of each handwritten name given each word model and ranked the models based on how likely they were to have produced the handwritten name.

If this method were really applied to classifying handwritten clinical notes, we would likely present the user with a list of the "top hits" (probably about 10) ranked by their posterior probabilities. In addition, we would have trained separate HMMs for both the patient's first name and last name, which would help us narrow down the pool of names dramatically since a document would have to have a high likelihood of containing *both* the first and last name for that patient to be listed as a top hit. However, we would also be dealing with many dozens more word models when making these predictions, increasing our chances of a false positive. Because of all these considerations, I considered a test classification "accurate" if the correct patient name showed up within the top 5 hits.

However, there is a nontrivial chance that a name will end up within the top 5 names entirely by chance, since there are only 32 possible patient names. In fact, that probability is given exactly by the hypergeometric distribution. If I just pull five names randomly from the list of 32 names, I should observe that I obtain my name of interest about 15.6% of the time. If we then treat this probability as the parameter of a binomial distribution which describes the random variable "success" (obtaining the correct name within the top 5 names), we see that we need to achieve "success" on at least 11 tests out of 40 (on the training set) or 8 out of 28 (on the test set) to be doing better than chance with $p < 0.05$. So this is our benchmark for hope, using this method.

4. Experiment

4.1. Parameter Adjustment

The construction of these HMMs is, by its very nature, somewhat heuristic. In addition, it takes an hour or so just to extract all of the relevant features from each set of 800 training examples and build the 10-state HMMs, so a full exploration of the entire parameter space for these models was impossible. However, I did notice several important patterns:

- Test set classification accuracy displayed a weak positive correlation with word length, although this association was not quite statistically significant ($p = 0.06$, Figure 5).

- Fixing the width of the bounding box at 100 pixels during training and testing helped tremendously with respect to accuracy on the handwritten test set. It seems that having a roughly uniform number of features to build each HMM helps normalize the posterior probabilities, even though one of the advantages of using an HMM is that it can handle sequences of variable length.

- The Gaussian filter parameters used to blur the images prior to extracting the LGH features were important in determining test set accuracy. Insufficient blurring led to a high degree of noise in the data and lowered the classification accuracy.

- There is a tradeoff between using a large window width (e.g. 30 pixels or more) vs. a small window width (e.g. 12-16 pixels) when extracting the SIFT features. Larger windows reduce noise in the feature space because more pixels are incorporated into each of the $4 \times 4 \times 8$ bins, so fewer of them are empty. However, using a larger window size also means that we assign less importance to fine-grained local changes in the gradient and we generate fewer features overall, which decreases the number of states we can use in our final HMMs.

4.2. Evaluation of Final Model

The word HMMs performed excellently on the training data, as expected. Out of 40 repetitions of each of 32 names, only 6 were classified incorrectly (Table 1). If we consider the few names

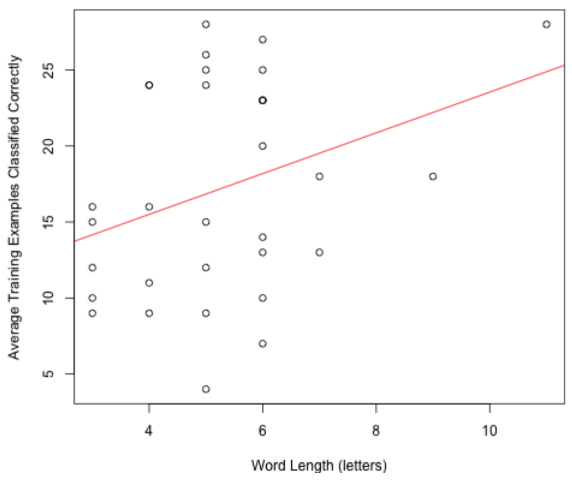

Figure 5. Test set classification accuracy displayed a weak positive correlation with word length. Longer words tended to be classified accurately on more occasions than shorter words.

that did not reach first place on the posterior-probability ranking, we see some obvious patterns, which can help alert us to likely future issues when we attempt to recognize handwritten names (Figure 6).

Results on the test set were, as expected, more mixed. Some names were classified correctly nearly 100% of the time - “Christopher” and “Myrna” were both classified perfectly (Figure 8) - while others barely beat a random assignment model. Two names, “Min” and “Brian”, were not classified any better than they would have been by chance.

Figure 7 shows a set of names that were classified correctly and another set that were not. On the surface, it is very difficult to see a difference between the correctly and incorrectly-classified names, though I think I notice a few patterns. For example, if the lower baseline of the word is unable to be straightened - if it is wavy, for example - the word tends to be classified incorrectly. In addition, if an important letter in the word, such as the first letter, can be represented multiple ways in handwriting - the capital letter “A” can be written as “A” or as a bigger version of the lower-case “a”, for example - this obviously causes problems. And finally, as expected, printed names are easier to recognize than cursive names, though I

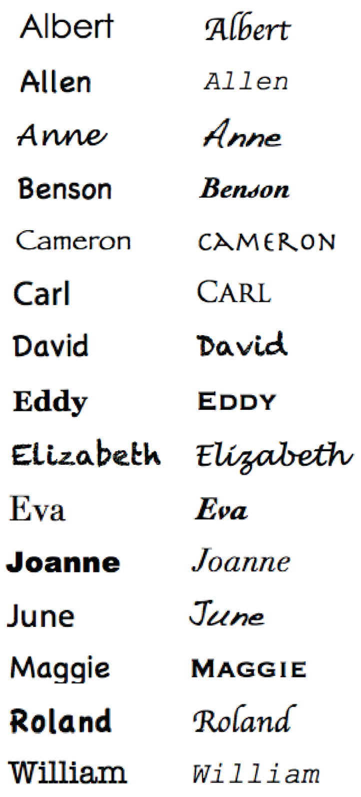

Figure 6. Some typed names that were consistently ranked first during prediction on the training set (left), next to others that were often second or third (right). We can see that the HMMs have difficulty interpreting block capitals or fonts with ornamentation, as well as highly slanted fonts.

was surprised at the number of cursive names that were classified correctly.

4.3. Summary and Future Work

The model's overall accuracy on the test set was 64%, which is - frankly - much better than I was expecting. For all names except two, the classification accuracy is significantly higher than we would expect by chance (more than 8 assignments correct; $p < 0.05$). I also think that some modifications could be made that would improve these models' future performance. For example, the size of ascenders and descenders varied dramatically throughout the test set. One could imagine normalizing not only the inter-baseline height of the word, but also the heights of the ascenders and descenders. Doing so would put the center of the LGH window at roughly the same place in the word for each training example, which should improve accuracy. Unfortunately I did not think of this strategy until it was too late to implement it, but I plan to try it in the near future.

Table 1. Results of best model on test set, ranked in order of test set accuracy.

Name	Training Accuracy (40)	Test Accuracy (28)
Christopher	40	28
Myrna	40	28
Maggie	40	27
Roland	40	27
Philip	40	25
Henry	40	24
Kathy	40	24
Rosa	40	24
William	40	24
David	40	23
Eddy	40	23
Thomas	40	23
Carl	40	21
Gerald	40	20
Anne	40	19
Cameron	40	19
Robert	40	19
Benson	40	17
Elisa	40	17
Elizabeth	40	17
Dennis	40	16
Lee	39	14
Kevin	40	12
Albert	40	11
Allen	40	11
Eva	39	11
Sid	40	10
Dan	38	9
Joanne	40	9
June	39	9
Min	39	7
Brian	40	5

One limitation of the current training/test set was the low resolution of the images involved. I obtained the test images by scanning pages of handwritten samples and segmenting out the words by taking screenshots (and I did the same thing for the typed fonts, only using screenshots from Word), which meant that each training image was only a few dozen pixels wide.

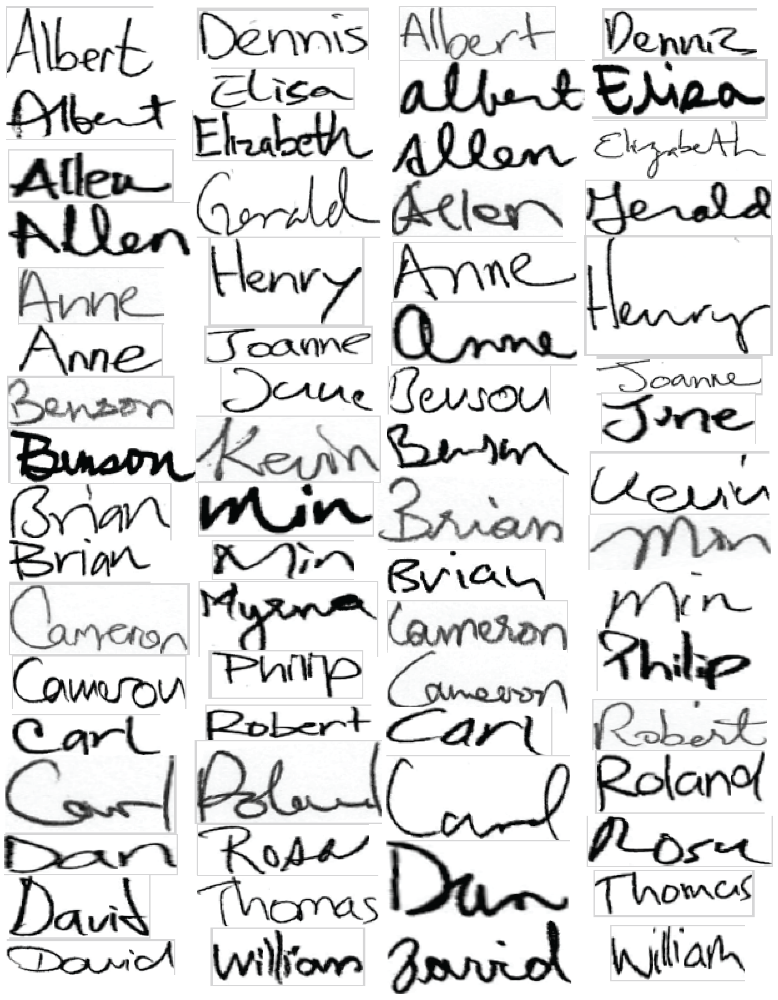

Figure 7. The names on the left half of this figure were classified correctly, and those on the right were not. It is very difficult to see why the classifier could correctly assign some of these names and not others!

I could definitely improve this method by cropping and saving high-resolution images from the scanned documents using Photoshop, but this process would be much more time-consuming than taking screenshots (which already took a long time). This would increase the number of pixels in the final images, which could help smooth the LGH features.

Finally, I will eventually need to perform one other test of this method, which will be to see if the posterior probability for a real patient name - say "Christopher" - is significantly higher than the posterior probabilities obtained by applying the "Christopher" HMM to other random words in the document. In some sense, comparing the name models to each other does the same thing, because if the "Christopher" model achieves high posterior probabilities on all words, we would expect overall test set accuracy to decrease. But it would still be good to test the models on a uniform set of "filler" words to see how much of a signal we see when examining the correct name vs. other words. Some models will perform better than others on random words just by chance [2], and there is a whole literature on score normalization to deal with this problem, but it was beyond the scope of this project.

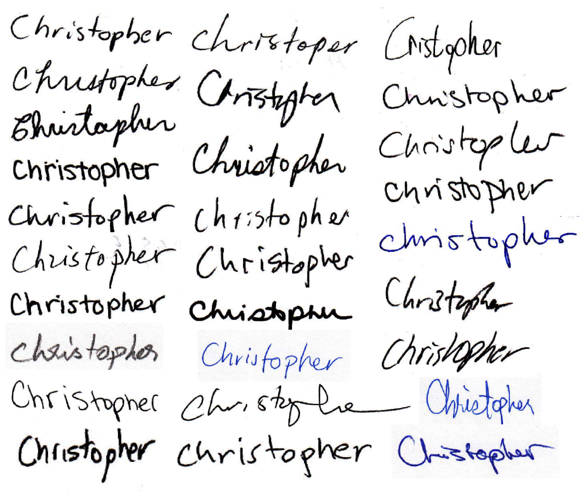

Figure 8. One of the best-recognized handwritten names in every experiment, the name "Christopher" is long, has many distinctive character patterns, and the individual letters are quite well segmented in most cases.

5. Conclusion

Using artificially-generated training data to train patient name HMMs leads to a set of models that can be used to find patient names in handwritten clinical notes. Although they are built using "fake" handwriting, these HMM classifiers perform surprisingly well, and there are several possible routes to improvement. In addition, if we were able to search for both the patient first and last name in clinical documents, we would expect accuracy to increase further, since last names are generally much more unique than first names.

References

[1] Manmatha R, Han C, Riseman EM, 1996. Word spotting: a new approach to indexing handwriting, in: Proc. Of the 1996 IEEE Conf. on Computer Vision and Pattern Recognition, p. 631.

[2] Rodriguez J, Perronnin F, 2009. Handwritten word-spotting using Hidden Markov models and universal vocabularies. Pattern Recognition 42(9), 2106-2116.

[3] Zhang B, Srihari SN, 2003. Binary vector dissimilarity measures for handwriting identification, in: Document Recognition and Retrieval X, pp. 28-38.

[4] Zhang B, Srihari SN, Huang C, 2004. Word image retrieval using binary features, in: Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 5296.

[5] Leydier Y, Bourgeois FL, Emptoz H, 2005. Omnilingual segmentation-free word spotting for ancient manuscripts indexation, in: Proc. Of the 8th Int. Conf. on Document Analysis and Recognition, pp. 533-537.

[6] Rothfeder J, Feng S, Rath T, 2003. Using corner feature correspondences to rank word images by similarity, in: Workshop on Document Image Analysis and Retrieval.

[7] Rath TM, Manmatha R, 2003. Word image matching using dynamic time warping, in: Proc. Of the 2003 IEEE Conf. on Computer Vision and Pattern Recognition, pp. 521-527.

[8] Kolcz A, Alspector J, Augusteijn M, Carlson R, Popescu GV, 2000. A line-oriented approach to word spotting in handwritten documents. Pattern Analysis and Applications 3(2): 153-168.

[9] Adamek T, Connor NE, Smeaton AF, 2007. Word matching using single closed contours for indexing handwritten historical documents, Int. J. Doc. Anal. Recognit. 9(2):153-165.

[10] Plamondon R, Srihari SN, 2000. On-line and off-line handwriting recognition: a comprehensive survey. IEEE Trans Pattern Anal Mach Intell 22: 63-82.

[11] Rabiner LR, 1989. A tutorial on hidden Markov models and selected applications in speech recognition, Proc. of the IEEE 77, pp. 257-286.

[12] Vinciarelli A, Bengio S, Bunke H, 2004. Offline recognition of unconstrained handwritten texts using HMMs and statistical language models. IEEE Trans. Pattern Anal. Mach. Intell. 26(6): 709-720.

[13] Edwards J, The YW, Forsyth DA, Bock R, Maire M, Vesom G, 2004. Making Latin manuscripts searchable using gHMMs, in: Neural Information Processing Systems.

[14] Chan J, Ziftci C, Forsyth D, 2006. Searching off-line Arabic documents, in: Proc. Of the 2006 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, pp. 1455-1462.

[15] Cao H, Govindaraju V, 2007. Template-free word spotting in low-quality manuscripts, in: Sixth Int. Conf. on Advances in Pattern Recognition.

[16] Bunke H, 2003. Recognition of cursive Roman handwriting past, present and future, in: Proc. Of the 7th Int. Conf. on Document Analysis and Recognition, Vol. 1, pp. 448-459.

[17] Huang C, Srihari SN, 2008. Word Segmentation of Off-line Handwritten Documents. in: Proc. Document Recognition and Retrieval XV, San Jose, CA, SPIE Vol. 6815, January 2008, pp. 68150E-1-6.

[18] Kim S, Jeong S, Lee G-S, Suen C, 2001. Word segmentation in handwritten Korean text lines based on gap clustering techniques, in: Proc. Of the Sixth Int. Conf. on Document Analysis and Recognition, p. 189.

[19] Mahadevan U, Nagabushnam RC, 1995. Gap metrics for word separation in handwritten lines, in: Proc of the 3rd Int. Conf. on Document Analysis and Recognition, Vol. 01, p. 124.

[20] Fischer A, Keller A, Frinken V, Bunke H, 2011. Lexicon-free handwritten word spotting using character HMMs. Pattern Recognition Letters Epub ahead of print.

[21] Marti U-V, Bunke H, 2001. Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system. Int. J. Patt. Recog. Art. Intell. 15, 65-90.

[22] Rodriguez JA, Perronnin F, 2008. Local gradient histogram features for word spotting in unconstrained handwritten documents, in: Int. Conf. on Frontiers in Handwriting Recognition.

[23] Lowe DG, 2004. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision 60(2): 91-110.

[24] Murphy, K., Dunham, M., 2008. PMTK: Probabilistic modeling toolkit. Neural Information Processing Systems (NIPS) Workshop on Probabilistic Programming.

[25] Baum LE, Petrie T, Soules G, Weiss N, 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. Annals of Mathematical Statistics 41(1): 164-171.

[26] Blumenstein M, Cheng CK, Liu XY, 2002. New Preprocessing techniques for Handwritten Word Recognition, Proc. of the 2nd IASTED conference on visualization, Imaging and Image Processing, pp. 480-484.

[27] Bozinovic RM, Srihari SN, 1989. Off-line cursive script word recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(1): 68-83.

[28] Vinciarelli A, Luettin J, 2001. A new normalization technique for cursive handwritten words. Pattern Recognition Letters 22: 1043-1050.