A Hidden Markov Model for Alphabet-Soup Word Recognition

Shaolei Feng
Dept. of Computer Science
University of Massachusetts
Amherst, MA-01003
slfeng@cs.umass.edu

Nicholas R. Howe
Dept. of Computer Science
Smith College
Northampton, MA-01063
nhowe@cs.smith.edu

R. Manmatha
Dept. of Computer Science
University of Massachusetts
Amherst, MA-01003
manmatha@cs.umass.edu

Abstract

Recent work on the "alphabet soup" paradigm has demonstrated effective segmentation-free character-based recognition of cursive handwritten historical text documents. The approach first uses a joint boosting technique to detect potential characters - the alphabet soup. A second stage uses a dynamic programming algorithm to recover the correct sequence of characters. Despite experimental success, the ad hoc dynamic programming method previously lacked theoretical justification. This paper puts the method on a sounder footing by recasting the dynamic programming as inference on an ensemble of hidden Markov models (HMMs). Although some work has questioned the use of score outputs from classifiers like boosting and support vector machines for probability estimates, experiments in this case show good results from treating shifted boosting scores as log probabilities.

Keywords: character detection, word recognition, inference models, cursive, historical manuscripts

1. Introduction

Handwritten cursive documents continue to pose challenges for text recognition methods. Handwritten documents with large vocabularies [16] and handwritten historical documents [14, 8] are particularly challenging. One recently proposed approach employs high-quality letter detection techniques in an unsegmented framework to identify the characters present in each word [7]. Called an "alphabet soup" paradigm after the jumble of candidate letter detections produced, this technique must then determine the correct sequence of characters that form the word tag. The prior work uses a dynamic programming method without a formal inference model. In this paper we present a more principled approach using using an ensemble of hidden Markov models for the letter assembly task.

To create the "alphabet soup", a joint boosting algorithm modeled on work in object recognition [15] detects individual letters by scanning across a word or document image. Performing detection rather than segmentation allows the method to easily entertain many overlapping hypotheses for letters and position. The particular method used here employs an ensemble of easily testable features, taking advantage of those that are common to multiple characters. For example **d**, **o** and **g** all share a common part, and a single test may provide evidence in favor of or against all three. The training phase of joint boosting deliberately selects features which are common to multiple character classes. This both decreases the amount of training required and increases the number of samples per feature, resulting in robust detection with relatively shorter training time.

Letter detection is applied to every position along the horizontal extent of a word image, resulting in a large number of possible character detections for every word - only some of which are correct. (The procedure may also be applied to entire lines or even page images, but we focus on words in this paper.) Valid characters may also be missed in the detection phase. Further analysis must therefore identify the correct sequence. Previous work does this using generic dynamic programming [7] but hidden Markov models (HMMs) offer a more principled way to find a sequence with maximum posterior probability. One advantage of the HMM is that it decouples the detection steps from the other steps in the process and allows one both to understand the technique better and to more easily make changes to different parts of the estimation.

The character detections occur at known positions and therefore can be placed in order, such that the detections corresponding to the correct sequence appear strictly from left to right. The model used is an ensemble of HMM's generated automatically from the detection sequence and statistics from analysis of a training corpus. One HMM is created for each possible word length, from one character up to the total number of detections. The HMM for word length m has m states (plus implicit states for start and end of word). Each regular state generates a corresponding detection, while transitions between states correspond to character transitions with probability derived from a transcribed corpus. For each length of HMM, the Viterbi algorithm determines an optimal sequence, and thus the ensemble of HMMs produces one optimal sequence for each length. The globally optimal sequence with correct length is found by dividing the probability of each sequence by its length m and selecting the maximum.

Estimating the generative probability correctly is crucial to good accuracy. Two different techniques are used to estimate the generative probability. The first approach fits Poissonians for valid and non valid detections on a character basis and from these computes a mapping from scores to probabilities. The second approach transforms the weighted score from the detection stage to a probability, treating a linear shift of the detection scores as log probabilities. Although previous work with support vector machines [2] and boosting [12] has indicated that the scores cannot reliably be used as probabilities, the experiments here show perhaps surprisingly that the latter technique works quite well. In fact it is identical in implementation to the dynamic programming technique used for assembling the characters in prior work [7].

The next section discusses related work. This is followed by a summary of the preprocessing and letter detection steps, as described in more detail elsewhere [7]. Section 4 presents the new HMM framework. The last two sections describe experiments with the new system and conclude the paper.

2. Related Work

Offline handwriting recognition has worked well in small-vocabulary and highly constrained domains like bank check recognition and high postal address recognition. In recent years researchers have investigated large vocabulary handwritten documents using HMM's [11, 16]. Marti and Bunke [11] proposed to use a Hidden Markov model (HMM) for handwritten material recognition. Each character is represented using a Hidden Markov model with 14 states. Words and lines are modelled as a concatenation of these Markov models. A statistical language model was used to compute word bigrams and this improved the performance by 10%. Vinciarelli et al. [16] used a similar model. Both papers used constrained modern handwriting to test their results.

Handwritten historical manuscripts are even more challenging since the vocabulary may be large, they are often noisy and there are few constraints on them. Even papers of single historical figures like George Washington consist of multi-authored multi-writer collections; George Washington had almost 30 secretaries over the years who helped him draft and write the letters. Rath et al [10] focus on recognizing historical handwritten manuscripts using simple HMMs with one state for each word. By adding word bigrams from similar historical corpora they showed that the word recognition rate on a set of pages of George Washington's documents approached 60%. The experiments here are done on the same corpus. Adamek et al. [1] use novel features with nearest neighbors to obtain good performance on this dataset. Feng and Manmatha [5] compared a number of different kinds of models including conditional random fields and HMM's and showed that smoothing was important for good performance. Edwards et al. [4] use gHMM's to recognize Latin manuscripts. Rath et al. [14] used relevance models to create a search engine for historical documents while Howe et al. [8] used boosted decision trees to recognize handwritten documents.

The alphabet soup approach to word recognition resembles recent work on breaking visual CAPTCHAs [13]. This work also detects potential letters and searches for a likely combination, but uses a different algorithm for the assembly step. Also, no results have appeared in the literature for general text recognition under this method.

While HMM models have a strong history in both print and handwritten character recognition [9], the ensemble of HMM's proposed here is new and based on a model for aligning printed word characters to ground truth as proposed in [6].

3. Character Detection

Character detection works on binarized document images. Gradients of the binarized images are categorized into eight cardinal orientations and aggregated over spatial bins at three orders of resolution to yield a histogram of gradients (HoG) descriptor of the neighborhood around each point [3]. These 2830-dimensional descriptors capture the location and direction of gradient edges in the neighborhood, and thus serve to describe any character that may be present in the area. Joint boosting [15] searches for common patterns across classes, building up sets of feature tests that can classify each character type present in the document.

The boosting training set includes some character examples identified by hand and additional examples identified automatically from a document transcript [7]. The 128 combined training examples of each character class are used to train a joint boosting classifier, which learns to identify each individual class from its descriptor. This classifier, when applied to fresh HoG descriptors of previously unseen points, functions as the desired letter detector. Any point whose classifier score for a character exceeds a threshold (set at -5) becomes a potential letter detection. In cases where a sequence of adjacent points all exceed the threshold, only the local maximum point is retained as a candidate. Nevertheless, the classifier can identify numerous false positives due to the low threshold. The word recognition step described below determines which are real by finding the sequence of detected characters with the greatest probability of correctness.

4. A Hidden Markov Model for Word Recognition

In this section, we propose a hidden Markov model (HMM) to recognize a sequence of characters of fixed length given the character detection results. Note that generation of the required HMM probabilities proceeds automatically from statistics measured on a training corpus with transcript and the specific sequence of character detections. A custom HMM provides a natural way to determine the most probable sequence of characters selected from the detection sequence, which may contain both false positives and false negatives due to ambiguities in the written characters and imperfections in the detector. The HMM explicitly combines information about character transition, character visual appearance, and the horizontal spacing.

As a sequence model, the HMM has the advantage of utilizing character dependence information for decoding. The proposed HMM model recovers the most probable sequence of detected characters by integrating information on the character dependence, the visual appearance and the relative positions of detected characters. Let $D =< d_1, d_2, \ldots, d_n >$ represent the sequence of candidate detections obtained in the detection step, where n indicates its length. Each element d_k in the detection sequence is denoted as a triple $d_k = (c_k, \phi_k, x_k)$, where x_k is the cartesian coordinate of the k-th detection, c_k the character and ϕ_k the detection score for detecting c_k at that position. Since false positives may exist in the candidate detection sequence, the length m of the genuine word is taken as an integer within $[0, n]$, i.e. $0 \leq m \leq n$. 0 corresponds to the extreme case where all detections are false positives. For each possible length m of a possible latent word, we build an HMM consisting of m state nodes, each of which generates the observation at a particular position of the detection sequence. We represent the state sequence of the HMM as $S =< s_1, s_2, \ldots, s_m >$, where each state s_i in the HMM is an integral index to a position in the candidate detection sequence. The observation sequence $O =< o_1, o_2, \ldots, o_m >$ denotes the feature vectors generated by each of the state nodes. For example, if $s_i = 10$ then o_i is the feature vector extracted at the 10-th detection position from the word image. The HMM estimates the joint probability of the feature vector sequence and the

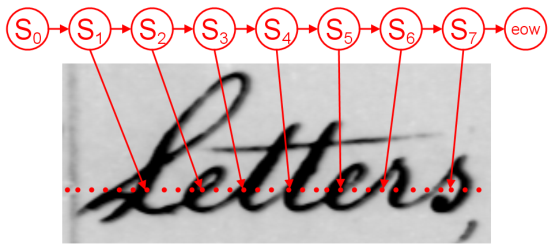

Figure 1. Diagram of the HMM with length equal to 7, showing hidden states and observation points. Note that the state at each node determines the corresponding observation point.

hidden position sequence $P(O, S)$ as:

$$P(O, S) = \prod_{i=1}^{m} P(s_i|s_{i-1})P(o_i|s_i) \qquad (1)$$

where $P(s_i|s_{i-1})$ is the transition probability which indicate the possibility of transition from one position s_{i-1} to another s_i in the detection sequence, and $P(o_i|s_i)$ the probability of generating the feature vector o_i from the s_ith possible detection. Figure 1 shows the diagram of the HMM with length equal to 7.

Inference in the HMM requires requires finding the $\tilde{S}$ maximizing $P(O, S)$, i.e.:

$$\tilde{S} = arg\max_{S} P(O, S) \qquad (2)$$

4.1 Probability Estimation

4.1.1 Generative Probabilities

The generative probability $P(o_i|s_i)$ in this model is the probability of image feature set o_i given a true detection at the s_i-th detection position. The scores from the output of the boosting detector need to be mapped to probabilities.

One approach models the distributions of positive and negative detection scores to get an estimate of $P(o_i|s_i)$. A given boosted detector comprises some number of feature tests representative of each character class on the training set. Positive examples of the class should pass most of the tests, and negative examples should fail most of them. The distribution of both positive and negative scores may therefore be modeled by a Poissonian distribution. Fitting a Poisson to the observed distribution on a training set yields a smooth curve, which allows one to map scores to probabilities. Figure 2 shows the score profiles and their fitted models for one character class.

A second alternative estimate of $P(o_i|s_i)$ comes directly from the exponential of the score ϕ_{s_i} reported by the letter detector, times a constant β small enough to ensure that $P(o_i|s_i) << 1$ (see Equation 3). The Viterbi

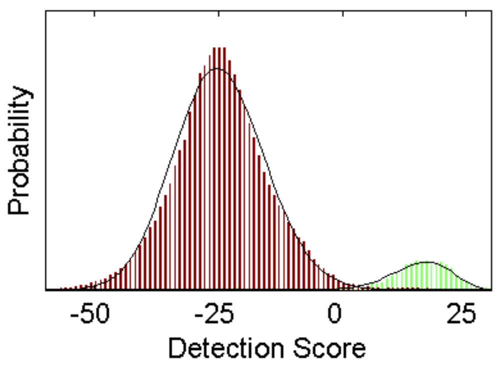

Figure 2. Histogram of detection scores for positive (green) and negative (red) examples of the letter 'h'. The positive scores have been exaggerated by a factor of 10 for visibility. Curves show fitted Poissionian curves, used to estimate the generative probability.

algorithm computes probabilities using Equation 1. By taking the logarithm of both sides in Equation 1 it can be shown that a constant $m\beta$ is added to all character chains of the same length and hence this does not affect the output of the Viterbi algorithm (which maximizes likelihood). In the final step when chains of different lengths are compared, the scores are divided by the length m and hence the constant is again the same for all chains. That is, the choice of β does not change the result. Thus previous work using dynamic programming on the raw detection scores [7] turns out to be equivalent in implementation to inference using Equation 3. Experiments show that this approach works well. Effectively, the boosted scores are treated as logarithms of the generative probabilities, up to a constant. This is somewhat surprising since the literature indicates that the output scores of classifiers such as support vector machines [2] and AdaBoost [12] are not necessarily good probability measures.

$$P(o_i|s_i) = \beta exp(\phi_{s_i}) \tag{3}$$

4.1.2 Transition Probabilities

The transition probability $P(s_i|s_{i-1})$ is essentially the transition from the s_{i-1}-th detection $d_{s_{i-1}}$ to the s_i-th detection d_{s_i}, which measures the possibility that two consecutive characters in the real word correspond to the s_{i-1}-th and the s_i-th candidate detections respectively. (Note that s_i and s_{i-1} are consecutive Markov states, but do not necessarily refer to consecutive detections.) This probability is determined by two different components of the detections: the candidate characters and the cartesian coordinates/positions of the detection. The candidate character transition models the statistical dependency of characters, i.e. the conditional probability of one character occurring given the previous character. The position transition models the relative horizontal separation of different characters in word images. This probability penalizes unusual (too large or too small) separations of the two candidate detections. Formally, the character transition $P(c_{s_i}|c_{s_{i-1}})$ is estimated from the smoothed bigrams of characters in the training set (or from an external corpus). The position transition is estimated as a Gaussian function of the separations:

$$\begin{aligned} &P(x_{s_i}|x_{s_{i-1}}) = \\ &\exp(-\frac{((x_{s_i} - x_{s_{i-1}}) - \mu_{s_i s_{i-1}})^2}{2\sigma^2_{s_i s_{i-1}}}) \end{aligned} \tag{4}$$

where $\mu_{s_i s_{i-1}}$ is the mean separation of the characters c_{s_i} and $c_{s_{i-1}}$ estimated from training set, and $\sigma_{s_i s_{i-1}}$ the corresponding standard deviation. The transition probability $P(s_i|s_{i-1})$ is estimated as a weighted combination of the character and position transitions:

$$P(s_i|s_{i-1}) = \lambda P(c_{s_i}|c_{s_{i-1}}) + (1-\lambda)P(x_{s_i}|x_{s_{i-1}}) \tag{5}$$

where λ determines the weights for the two components. The value of λ may be estimated from a validation set. For simplicity, we have used a predefined value in our experiments.

The character separation $\mu_{s_i s_{i-1}}$ and deviation $\sigma_{s_i s_{i-1}}$ numbers are estimated from a model that assigns a width and deviation to each character type, and averages them to find the corresponding value for any two characters. The character widths in turn are measured from estimated positions of the characters in training data with supplied transcript. Values for common characters are used directly, while uncommon characters are smoothed. The treatment of this problem is reported elsewhere [7].

4.2 Decoding the Most Likely Word

The Viterbi algorithm is used to determine the most likely state sequence $\tilde{S}$ of an HMM; for the HMM of any given length m denote this as $\tilde{S}_m$. The algorithm keeps track of both the most likely state at each position of the word and the likelihood associated with it. Specifically, the log likelihood of decoding the i-th state as the k-th candidate detection is calculated as:

$$\gamma_i(k) = \phi_k + \max_{j=0}^{k}[\gamma_{i-1}(j) + \log(P(k|j))] \tag{6}$$

where the latent constraint $j \leq k$ ensures that the decoding never traverses the detection sequence backwards. During the decoding, the Viterbi algorithm keeps track of the path leading to the current state k by recording its prior state $\psi_i(k)$ at each step i:

$$\psi_i(k) = \arg\max_{1\leq j\leq k-1}[\gamma_{i-1}(j) + \log(P(k|j))] \tag{7}$$

Since we build a separate HMM for each possible length ($0 \leq m \leq n$) of the real word, after the Viterbi decoding we get the n most likely word labels of different lengthes. We denote these most likely words as W_m and the corresponding likelihoods as γ_m, with $0 \leq m \leq n$. Note that although we define a separate HMM for each possible word length, the Viterbi scores calculated for a shorter sequence can be reused with a longer sequence for significant computational savings. Thus the Viterbi scores $\gamma_{m+1}(k)$ at the $m+1$ step can be calculated using the scores $\gamma_m(k)$ calculated at the last step of the sequence of length m according to Equation 7.

The character sequencer just described identifies the best character sequence for each possible length up to the total number of detections n. Comparing γ_m between sequences of different length may be misleading since longer sequences include more terms and hence may potentially have a bias toward lower score. In general, any specific word containing more letters may be expected to have lower likelihood than a shorter word since more letters offer more possible combinations overall. The normalized likelihood therefore picks the correct sequence more often when comparing possibilities of different lengths.

$$\hat{\gamma}_m = \frac{\gamma_m}{m} \tag{8}$$

Then the most likely word is the $W_{\tilde{m}}$ with:

$$\tilde{m} = \arg\max_m \hat{\gamma}_m \tag{9}$$

5. Experiments

We test the new inference method under the same experimental conditions used to evaluate the original alphabet soup algorithm. Twenty pages of correspondence from the letters of George Washington make up the test set. These are written in longhand script by several of Washington's secretaries, so they represent multiple styles. We use the same word image segmentations as previous work [10]. The distribution of word lengths appears in Figure 3.

The experiments follow a 20-fold cross-validation methodology, with nineteen pages used for training and one page for testing, alternated until all pages have been tested. Only half the pages (either the even or odd pages not including the test page) are used to build the letter detector, since this is the most time consuming step and ten pages easily provide sufficient training samples of most characters. All nineteen training pages contribute to the estimation of character bigrams and separations.

The joint boosting process builds a letter detector based on 2000 feature tests, trained on 128 examples of each character class. There are sixty character classes total, including all lowercase letters, numerals, most uppercase letters, and one instance of the British pound symbol £. Candidate detections include all points with $\phi_k > -5$ that are local maxima of ϕ_k with respect to their neighbors.

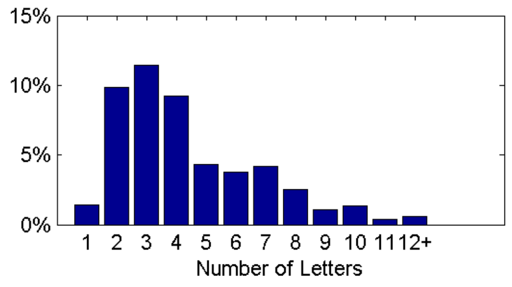

Figure 3. Distribution of word lengths in the GW20 corpus.

Table 1. Results of inference on the ensemble of HMMs using two different generative probability estimates. Character error rates are computed as total string edit distance between the ensemble prediction and ground truth, divided by the number of ground truth characters. Allowable edits include elisions, insertions, and one-for-one or two-for-one substitutions.

Probability Estimate:	Score Profile Models	Raw Boosting Score
Character error rate (%):	39 ± 4	19 ± 4

Table 1 presents the results of inference on the ensemble of HMMs using the two different estimates of the generative probability described in Section 4.1.1. We choose to cite character error rates here to emphasize the fact that our method performs character-by-character inference. The first result given, based upon score profiling as illustrated in Figure 2, represents a new experiment. The second, based upon Equation 3, represents results equivalent to prior work [7]. As is evident from the table, the second method gives much better results.[1] This indicates that treating boosting scores as log probabilities may be more reasonable than could be expected expected based upon existing research [12].

6. Conclusion

Establishing an HMM model for character assembly in alphabet soups improves our understanding of that word recognition process. Simultaneously, it highlights areas that could benefit from further investigation, in particular

[1]With postprocessing, the word-recognition rate using this implementation averages 72% on the George Washington letters for unrestricted vocabulary [7].

the generative probability estimates and the comparison of word predictions of different lengths.

The noticeable difference in the error rate between our two estimates of the generative probabilities highlights the importance of this crucial measurement, and serves as motivation to find estimates that will serve better still. The utility of the raw boosting scores in this capacity comes as a surprise in light of prior results to the contrary. This may indicate a superior property of joint boosting as compared with other boosting variants. In any case, the discovery would not have been possible without the theoretical framework provided by the ensemble-of-HMMs model.

Acknowledgment

Shaolei Feng and R. Manmatha were supported in part by the Center for Intelligent Information Retrieval and in part by a grant from Google, and all the authors were supported in part by grant #NSF CNS-0619337. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsor.

References

[1] T. Adamek, N. E. O'Connor and A. F. Smeaton, "Word Matching Using Single Closed Contours for Indexing Handwritten Historical Documents", *International Journal of Document Analysis and Recognition*, 9:153–165, 2007.

[2] C. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[3] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection", *IEEE Conference on Computer Vision and Pattern Recognition*, 2005, pp 886–893.

[4] J. Edwards, Y. W. Teh, D. Forsyth, R. Bock, M. Maire and G. Vesom, "Making Latin Manuscripts Searchable using gHMM's", *Advances in Neural Information Processing Systems 17*, 2005, pp 385–392.

[5] S. Feng and R. Manmatha, "Exploring the Use of Conditional Random Field Models and HMMs for Historical Handwritten Document Recognition", *the Proceedings of the 2nd IEEE International Conference on Document Image Analysis for Libraries (DIAL 06)*, 2006, pp 30–37.

[6] S. Feng and R. Manmatha, "A Hierarchical, HMM-based Automatic Evaluation of OCR Accuracy for a Digital Library of Books", *ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'06)*, 2006, pp 109–118.

[7] N. Howe, "Finding Meaning in Alphabet Soup: Segmentation-Free Character Recognition for Longhand Script", Technical report, Smith College, 2008.

[8] N. Howe, T. Rath and R. Manmatha, "Boosted Decision Trees for Word Recognition in Handwritten Document Retrieval", *28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2005.

[9] S. Impedovo, "Hidden Markov Models in Handwriting Recognition", S. Impedovo, editor, *Fundamentals in Handwriting Recognition*, 1994.

[10] V. Lavrenko, T. Rath and R. Manmatha, "Holistic Word Recognition for Handwritten Historical Documents", *Proceedings of the IEEE Workshop on Document and Image Analysis for Libraries*, 2004, pp 278–287.

[11] U.-V. Marti and H. Bunke, "Using a Statistical Language Model to Improve the Performance of an HMM-Based Cursive Handwriting Recognition System", *Int'l Journal of Pattern Recognition and Artifical Intelligence*, 15(1):65–90, 2001.

[12] D. Mease, A. J. Wyner and A. Buja, "Boosted Classification Trees and Class Probability/Quantile Estimation", *Journal of Machine Learning Research*, 8:409–439, October 2007.

[13] G. Mori and J. Malik, "Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA", *Computer Vision and Pattern Recognition*, 2003, volume 1, pp 134–141.

[14] T. M. Rath, R. Manmatha and V. Lavrenko, "A Search Engine for Historical Manuscript Images", *Proc. ACM SIGIR*, 2004, pp 369–376.

[15] A. Torralba, K. P. Murphy and W. T. Freeman, "Sharing Visual Features for Multiclass and Multiview Object Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(5):854–869, May 2007.

[16] A. Vinciarelli, S. Bengio and H. Bunke, "Offline Recognition of Unconstrained Handwritten Texts Using HMMs and Statistical Language Models", *IEEE transactions on PAMI*, 26(6):709–720, 2004.