Finding Words in Alphabet Soup: Inference on Freeform Character Recognition for Historical Scripts

Nicholas R. Howe [a]

[a] *Dept. of Computer Science, Smith College, Northampton, MA 01063*

Shaolei Feng [b] R. Manmatha [b]

[b] *Dept. of Computer Science, University of Massachusetts, Amherst, MA 01003*

Abstract

This paper develops word recognition methods for historical handwritten cursive and printed documents. It employs a powerful segmentation-free letter detection method based upon joint boosting with histogram-of-gradients features. Efficient inference on an ensemble of hidden Markov models can select the most probable sequence of candidate character detections to recognize complete words in ambiguous handwritten text, drawing on character n-gram and physical separation models. Experiments with two corpora of handwritten historic documents show that this approach recognizes known words more accurately than previous efforts, and can also recognize out-of-vocabulary words.

Key words: Character recognition, cursive text, historical text

1 Overview

Modern offline recognition methods transcribe most machine-printed text with ease, and also handle handwriting within restricted contexts such as postal addresses. But open-vocabulary cursive scripts remain a challenge, particularly for older documents in less than pristine condition. Handwritten documents with large vocabularies [1] and handwritten historical documents [2,3] are particularly challenging. Reliable recognition of texts from historical collections is often infeasible with current technology, and yet holds the potential to open new worlds to scholarship.

This paper introduces a handwriting recognizer with a flexible inference model that utilizes results from a character detector (rather than a segmented character recognizer). The character detector identifies putative characters and their locations along with associated confidence scores for each detection; this is the "alphabet soup". Usually there will be many more putative characters than real ones. The inference task thus is to identify the most probable sequence of correct detections. The choice must account for spacing and transition probabilities between letters, in combination with the level of confidence in each detection. In effect, selected detections are "strung together" to form a word in a manner that maximizes joint likelihood for all these considerations. (See Figure 2 below for an illustration.) For reasons detailed in Section 3 we use an ensemble of hidden Markov models (HMMs) for this purpose; simultaneous inference over the entire ensemble can be done using an efficient dynamic programming algorithm.

The character detector in this work is based on a classifier developed for object recognition and trained by a procedure called *joint boosting* [12]. Focusing

on detection allows us to entertain many overlapping hypotheses for letters and positions, and more easily handles connected text. This differs from traditional segmentation schemes which usually allow for a single hypothesis at each position. Because the system builds words out of individual characters, it can recognize novel words not seen in the training documents. Note that our approach decouples the character detector from the inference stage – one can easily replace the character detector with a different one that works better for a given purpose.

Comprehensive surveys document a wide range of methods employed for handwriting recognition, but only a minority handle unrestricted cursive text [4,5]. Prior work on cursive historic documents has favored a holistic word recognition approach [6,7,3,8], which creates an unreasonable burden in providing comprehensive training data. To address this, several works have attempted to build words out of smaller units [9–11]. However, each of these earlier works uses an inference model that limits the choices for character detection and representation.

In the next section we discuss the related work in more detail. We follow this up with a section which describes how the preprocessing and character detection are done. Section 3 describes the hidden Markov models and the inference schemes used. The next section describes the experiments performed on two different datasets while the last section concludes the article.

1.1 Related Work

Offline handwriting recognition has worked well in small-vocabulary and highly constrained domains like bank check recognition and postal address recogni-

tion. In recent years researchers have investigated large vocabulary handwritten documents using HMM's [13,1]. Marti and Bunke [13] proposed to use an HMM for handwritten material recognition. Each character is represented using an HMM 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. [1] used a similar model. Both papers test their results using the IAM data set, a large-vocabulary collection of modern multiple-writer handwriting created expressly for research in handwriting recognition.

Handwritten historical manuscripts present different challenges since they were not created with machine recognition in mind, their vocabulary may be large, and the documents themselves are often noisy. Even the 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 [14]. Rath et al [8] focus on recognizing historical handwritten manuscripts using simple HMMs with one state for each word. By adding word bigrams from similar historical corpora they show that the word recognition rate on a set of pages of George Washington's documents approached 60%. The GW experiments here are done on the same corpus. Adamek et al. [7] use novel features with nearest neighbors to obtain good performance on this dataset. Rath & Manmatha use word spotting to index the George Washington manuscripts [15]. Feng and Manmatha [16] compare a number of different kinds of models including conditional random fields and HMM's and show that smoothing is important for good performance. Edwards et al. [9] use gHMM's to recognize Latin manuscripts. Rath et al. [2] use relevance models to create a search engine for historical documents while Howe et al. [3] use boosted decision trees to recognize handwritten documents.

The approach to word recognition herein resembles recent work on breaking visual CAPTCHAs [17]. Like the present work, Mori & Malik detect potential letters and search for a likely combination, but their assembly algorithm differs from the inference used here. To date no results have appeared in the literature for general text recognition under their method and it is unclear whether such an application is feasible. Other segmentation-free approaches have also appeared recently [18,19].

While HMM models have a strong history in both print and handwritten character recognition [20], the ensemble of HMM's proposed here is new; it bears some relation to a model for aligning printed word characters to ground truth as proposed in [21].

2 Preprocessing and Character Detection

Historic documents vary widely in quality. Although the documents tested in Section 4 have suffered some degradation, they are in reasonable condition and show manageable amounts of staining and bleed-through. No scaling or deslanting are necessary in the experiments described here. Although the GW data set includes slanted text, the amount of slant remains fairly consistent and the recognition algorithm simply learns to detect characters with the slant. On the other hand, inconsistent ink fading can cause trouble and thus the documents are binarized [22]. Space constraints preclude describing details of the binarization method employed, as it is not central to the success of the word recognition at the focus of this paper.

The character detector in this work is simply a classifier that accepts a featural description of the environment of any point in the document, and determines

whether the point qualifies for membership in any of the character classes it has been trained to recognize. More specifically, the joint boosting classifier used here computes a set of scores to indicate its confidence that the point belongs to each and every target class; these scores will be used during inference as proxies for the log generative probabilities. (Normally positive scores indicate character class membership and negative scores indicate non-membership, but in practice the threshold will be set lower to include near-misses.) The remainder of this section describes the creation of the detector/classifier training set, the features used to characterize a point of interest, the boosting process, and further details of the detector/classifier application.

2.1 Character Model Training

Training requires multiple examples of each character class, taken from a training document similar to the one to be recognized. The baseline experiment (designated *T32* below) uses hand-identified samples of each target character, extracted using an interactive tool to draw polygons around select portions of the training images. This character training set contains around sixteen examples of each character class, or fewer for cases where the training data did not contain enough examples of a character.[1] The samples of each character class are aligned by subjecting them to an entropy-minimizing warping transformation called *congealing* [23], then mapping the center point of the transformed images back to the original. This provides a consistent detection point for all training samples.

Sixteen examples per class is insufficient for optimal classification. To provide

[1] In fact 32 samples of each character were identified where available, but these are split between two folds in the experiments.

additional training data to the classifier without excessive human effort, a second, extended training set utilizes automatic techniques to supplement the original hand-segmented samples with additional examples located automatically. The resulting set contains up to 128 examples of each character where available, selected from the training corpus, and is designated *T128* in the experimental results. The automatic training set expansion relies on the original T32 training samples and a transcript of the training data from which the additional samples are to be extracted. Section 3.5 below describes the method for automatically locating the additional samples. Because they have not been hand-segmented, the extra characters cannot be aligned using the same method as the originals. However, analysis of the results suggests that the lack of precise alignment actually improves the generality of the T128 detector/classifier.

2.2 Feature Sets

The features used for detection play a major role in determining its success. They must be simple enough to be consistent across diverse examples of each character, yet discriminative enough to distinguish between two characters with similar appearance. The histogram of gradients (HoG) used here consists of features that measure the fraction of image gradients within a given area that are aligned in a given range of directions [24]. Because the images have been binarized, gradients naturally group into eight directions plus areas of zero gradient. Nine binary maps therefore represent the spatial distribution of the various gradient orientations, and the original binary image forms a tenth distribution. Each of these maps is then summed across spatial histogram bins at varying resolutions to yield a final set of 2830 HoG features. More

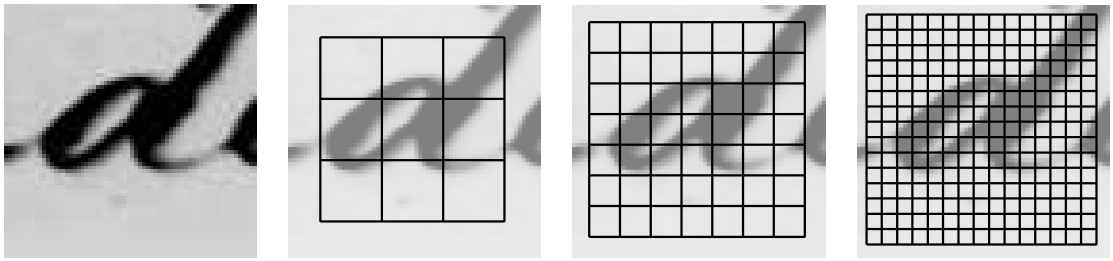

Fig. 1. A portion of a word, showing the spatial extent of the histogram of gradient features. Bins at three different resolutions are used.

specifically, the histogram bins used comprise a 15×15 array of bins each four pixels across, a 7×7 array of bins each eight pixels across, and a 3×3 array of bins each sixteen pixels across, as shown in Figure 1. At each resolution the bin arrays center on the detection point.

2.3 Boosting Implementation

Boosting classifiers learn an additive function that maps feature sets to class indicator scores, which should be either positive or negative depending on the correct class label of each sample. The implementation of joint boosting closely follows that of Torralba, et. al. [12] and is therefore not detailed here. Training proceeds in rounds, selecting one optimal feature for addition to the classifier at each round, modifying the indicator scores for each class according to the value of the selected feature. Our implementation differs in only one significant respect: Torralba et. al. assess each possible feature f and threshold θ by summing a binary threshold function $\delta(v_i^f > \theta)$ over each training instance i. Because noise can modify a HoG feature that is close to a decision threshold, features that discriminate classes by a wide margin are preferable to ones that discriminate them by a narrow margin. To bias the classifier towards more reliable features, we replace the binary threshold with the continuous

sigmoid-like function shown below.

$$\tilde{\delta}(v_i^f, \theta) = \frac{1}{2}\left(\text{erf}\left(\frac{v_i^f - \theta}{\sigma}\right) + 1\right) \tag{1}$$

Here the free parameter σ controls the width of the border margin. Values falling within the margin are considered insufficiently discriminative. (In the experiments, σ is set to 5.) The change modifies the feature selection at each boosting step, giving preference to features offering wide discrimination margins. The technique resembles the method of Vedaldi et. al. [25] in applying additional constraints during training in order to improve later accuracy. It improves upon the simpler feature knock-out approach of Wolf and Martin[26] since it does not require extra training.

Training on the joint boosting algorithm proceeds for 1500 rounds on the T32 experiments and 2000 rounds on the T128 experiments. This is easily sufficient for perfect performance on the training set, yet empirically it appears short enough to avoid overfitting. Although the training takes many CPU hours, the resulting classifier runs fairly quickly: computing the detection score amounts to a weighted sum of either 1500 or 2000 feature values. The features themselves are essentially just pixels in multiresolution histogram images derived from the original word or document image, as outlined above in Section 2.2.

2.4 Character Detection

The boosting algorithm generates a detector/classifier that can evaluate each point in an image and produce a set of detection scores $B_c(x, y)$, where positive values indicate the presence of character class c. In practice, due to ambiguities in the written characters and imperfections in the detector, not all target

characters will register positive scores. Thus the detection threshold is set somewhat lower (-5 in the experiments) to avoid the problem of false negatives. Naturally this raises the rate of false positives, but these can usually be dealt with in the inference stage because the words they can form tend to be improbable in most cases. Points in the immediate vicinity of a strong detection may also exhibit scores over the detection threshold, but should not be recognized as independent detections. Thus only locally maximal scores are considered as potential detections.

If necessary, the character detector can search all points of an entire document. However, previous work on the GW test set used in the experiments assumes accurate word segmentation and baseline detection [3,8]. Under these conditions detection should be necessary only at the series of points situated along the midline of each word image. In practice subtle inconsistencies in baseline location can cause detection errors with this approach by causing the detector to look too high or too low. Experiments performed with midline-only search are designated *narrow* in Section 4 below. Additional experiments designated *broad* apply the detector over a vertical range up to two pixels above and below the nominal midline, taking the maximal score over this range to account for possible flaws in the midline location.

3 Hidden Markov Models for Word Recognition

This section describes an HMM to recognize a sequence of characters of fixed length given the character detection results. Since the actual length of any target word is unknown, as described below a set or *ensemble* of such HMMs will be used, one for each possible word length. Because the reader may not

find it obvious why a single traditional HMM will not suffice, a brief discussion of the ensemble's motivation follows.

HMMs offer a principled way to find a sequence with maximum posterior probability. However, standard HMM formulations whose states correspond to fixed or regular spatial positions have difficulty accounting for varying character separations without introducing a very large state space. The technique described here avoids this issue by using model states corresponding to word characters, which generate observations (i.e., detections) at positions that may vary according to a spatial probability distribution. HMM solutions with unusual spatial layout thus will have low probability, even if they appear likely with respect to character sequence and appearance. Using an ensemble of HMMs imposes little additional cost, since dynamic programming efficiently evaluates the maximum probability solution to all HMM models in the ensemble.

3.1 HMM Framework

The HMMs used herein explicitly combine information about character transition, character visual appearance, and horizontal spacing of characters. 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, and thus the character detector output constrains and informs the hidden state probabilities. Furthermore, the detections for any HMM sequence are constrained to appear in order from left to right. Transitions between states correspond to character transitions, with estimated probability based on spacing and character sequence statistics derived from a transcribed training corpus. For each length of HMM, the Viterbi algorithm determines an optimal sequence with

maximum posterior probability, 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 largest.

To be more specific, let $D = \langle d_1, d_2, \ldots, d_n \rangle$ represent the sequence of candidate detections obtained in the detection step, where n is the total number of detections over the threshold for a given word image. Each element d_k in the detection sequence is denoted by a triple (c_k, ϕ_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 $B_{c_k}(x_k)$ 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, the algorithm builds 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 $S = \langle s_1, s_2, \ldots, s_m \rangle$, where each state s_i in the HMM is an integral index to a position in the candidate detection sequence. The observation sequence $O = \langle o_1, o_2, \ldots, o_m \rangle$ denotes the detection triples generated by each of the state nodes. For example, if $s_i = 10$ then o_i is the triple extracted at the 10-th detection position from the word image, $(c_{10}, \phi_{10}, x_{10})$. The HMM estimates the joint probability of the observation sequence and the hidden position sequence $P(O, S)$ as:

$$P(O, S) = \prod_{i=1}^{m} P(s_i | s_{i-1}) P(o_i | s_i) \tag{2}$$

where $P(s_i|s_{i-1})$ is the transition probability indicating the possibility of transition from one position s_{i-1} to another s_i in the detection sequence, and

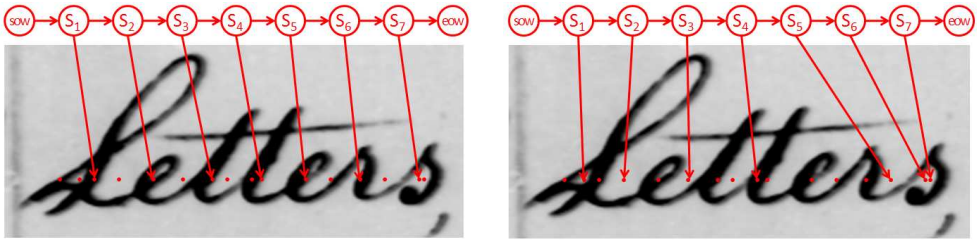

Fig. 2. Two possible configurations of an HMM with length equal to 7, showing hidden states and (illustrative) corresponding detection points, which vary according to the hidden state values. The left image depicts "letters", a high probability configuration. The right image depicts "Abutvsr", a configuration with low probability due to unusual letter transitions, irregular spacing, and use of low confidence detections.

$P(o_i|s_i)$ the probability of generating the feature vector o_i from the s_i-th possible detection. Figure 2 shows diagrams of an HMM with length equal to 7.

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

$$\tilde{S} = arg \max_{S} P(O, S) \tag{3}$$

3.2 Probability Estimation: 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 $B_c(x, y)$ from the output of the boosting detector need to be mapped to probabilities.

Empirically, direct conversion of the score ϕ_{s_i} reported by the letter detector yields an effective estimates of $P(o_i|s_i)$. The probability is taken as the exponential of the score, times a constant β small enough to ensure that $P(o_i|s_i) << 1$ (see Equation 4). The Viterbi algorithm computes probabilities using Equation 2. By taking the logarithm of both sides in Equation 2 it can

be seen 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 additional term is the same for all chains. That is, the choice of β does not change the result. 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 [27] and AdaBoost [28] are not necessarily good probability measures.

$$P(o_i|s_i) = \beta \exp \phi_{s_i} \tag{4}$$

3.3 Probability Estimation: Transition Probabilities

The transition probability $P(s_i|s_{i-1})$ measures the possibility that the character detected at d_{s_i} follows the character at $d_{s_{i-1}}$ consecutively in the word image under consideration. (Note that s_i and s_{i-1} are consecutive Markov states, but do not necessarily refer to consecutive detections since false detections may occur between them.) 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 expected 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:

$$P(x_{s_i}|x_{s_{i-1}}) = \frac{1}{\sqrt{2\pi}\sigma_{s_i s_{i-1}}} \exp\left(-\frac{((x_{s_i} - x_{s_{i-1}}) - \mu_{s_i s_{i-1}})^2}{2\sigma^2_{s_i s_{i-1}}}\right)$$

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 $\lambda = 2/3$ in our experiments, which seems to work well across many data sets. Details on the gathering of other statistics required for the model above appear in Section 3.5, following the discussion of the dynamic programming below.

3.4 Decoding the Most Likely Word

The Viterbi algorithm is used to determine the most likely state sequence $\tilde{S}$ of an HMM. The log likelihood of decoding the i-th state as the k-th candidate detection is denoted γ_i^k and computed in the standard manner.

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

where the latent constraint $j \leq k$ ensures that the decoding never traverses the detection sequence backwards. 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 most likely word labels of n different lengths. We denote these most likely words as W_m and the corresponding likelihoods as γ_m. Note that although we define a separate HMM for each possible word length, the Viterbi scores γ_m^k calculated for the length m sequence can be reused to compute γ_{m+1}^k for the length $m+1$ sequence, achieving significant computational savings. The entire computation corresponds to filling in the table shown in Figure 3.

The inference complexity scales as the cube of the number of detections. This has proved manageable in practice, with most words producing on the order of 100 or fewer detections, sometimes far less. However, the computation can be made quadratic if necessary with little change in the result by computing the maximum in Equation 6 over the most recent h states only, where h is large enough that only long-distance, low probability transitions are ignored.

3.4.1 Choosing a Word Length

Viterbi identifies the best character sequence for each possible length up to 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. All other things being equal, a word containing more letters may be expected to have lower likelihood than a shorter word since more letters offer more possible combinations overall. The best pick $W_{\tilde{m}}$ therefore normalizes the likelihood by word length.

$$\tilde{m} = \arg\max_m \frac{\gamma_m}{m} \tag{7}$$

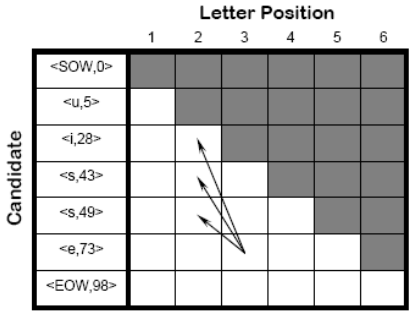

Fig. 3. Dynamic programming table. Rows correspond to candidate detections, sorted by their x coordinates. Columns correspond to possible placements in the output word label. Shaded boxes represent impossible configurations (i.e., the first/leftmost detection cannot be the second character in the word). Values are filled in by columns from left to right. The partial score γ_j^k entered in the table for a particular detection j in a particular word position k is the maximal value computed via Equation 6 over all possible sequences that could precede j at k. Arrows show the three possible immediate predecessors for one such computation.

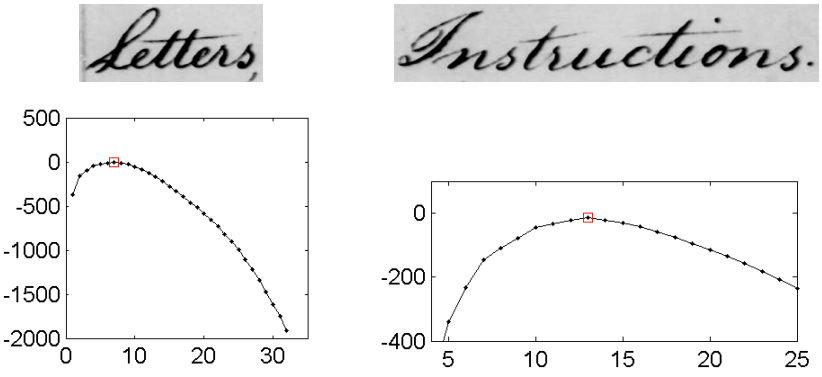

Fig. 4. Mean score per character transition of the best label at various lengths for two sample words. The prediction for the first word is "Letters" and for the second is "Instreictions". In the latter case, the incorrect 13-letter prediction has higher $\hat{\gamma}_m$ than the correct 12-letter prediction.

Figure 4 shows how $\hat{\gamma}_m = \gamma_m/m$ varies with word length m for several sample words. Word length errors remain a challenge: the experiments show that label accuracy could improve by up to ten percentage points if the length were always predicted correctly.

3.5 Estimating Character Positions and Separation Statistics

Evaluating the transition probabilities described in Section 3.3 requires character bigram and separation statistics. A training set with ground-truth transcription provides the bigrams, using standard backoff estimation with 10% holdout data [29]. 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 process begins by estimating the letter positions for all words in the transcribed training set. Access to the correct transcription simplifies the inference and makes accurate location tractable. $P(c_{s_i}|c_{s_{i-1}})$ is zero for all transitions not conforming to the transcript, and one for the correct transition. A heuristic estimate suffices for character separations $P(x_{s_i}|x_{s_{i-1}})$: separations are a normal distribution with mean equal to the width of the word divided by the number of characters; standard deviation is set *ad hoc* to 30% of this value. The computation of the best sequence considers all detection peaks for the known characters (without thresholding), and finds an optimum detetion sequence that represents the joint most likely character positions. Although there is no ground truth available for this task to give a numeric assessment, visual inspection of the results suggests a low error rate. Furthermore, errors are typically associated with low posterior probability, allowing for easy detection (see Figure 5).

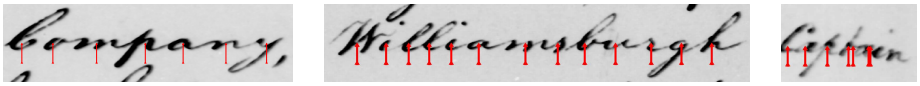

Fig. 5. Letter locations inferred from transcripts for several words. Dynamic programming scores are 20.2, 13.6 and -14.8 respectively. The low score of the third word reflects a location failure, visible in the second half of the word. Note that 'Williamsburgh' still has positive score despite the large separation between the sixth and seventh letters.

Estimated character positions still do not directly provide the expected separation μ_{ij} between two arbitrary characters c_i and c_j, because not all sequences will be observed in the training data. As with the bigram statistics, filling the gaps in the observations requires some sort of smoothing. The following model provides the necessary mechanism.

Suppose that each character c_i has an intrinsic width w_i independent of its neighbors, and that the separation between two neighboring characters c_i and c_j is thus $\mu_{ij} = \mu_{ji} = (w_i + w_j)/2$. With $n+2$ characters (including SOW and EOW) there are $n+2$ widths to estimate, but typically many more observed mean values, denoted μ^*_{ij}. This gives rise to an overconstrained linear system, with a least-squares solution to w_i. Because the character position estimates do contain occasional mistakes, μ^*_{ij} conservatively uses the trim mean of the separations of all observed instances of c_ic_j or c_jc_i, i.e., throwing out the highest and lowest 10% of the data and computing the mean on the middle 80%. [2]

The character widths w_i gives estimates for all mean separations μ_{ij}. A heuristic threshold then smooths the data: For sequences observed more than 5 times, the observed mean separation μ^*_{ij} is used directly; otherwise an interpolation with the character width estimates is used instead. Let $\mathcal{N}_{ij}$ represent

[2] Although logically implausible, with extremely sparse and corrupt data the least-squares solution can give a negative result for some w_i. In these rare cases the value is set arbitrarily to 0.

the number of observations of $c_i c_j$ or $c_j c_i$ in the training transcript.

$$\mu_{ij} = \begin{cases} \mathcal{N}_{ij} \geq 5 & : \quad \mu^*_{ij} \\ \mathcal{N}_{ij} < 5 & : \quad \frac{\mathcal{N}_{ij}}{5}\mu^*_{ij} + \left(1 - \frac{\mathcal{N}_{ij}}{5}\right)\frac{w_i + w_j}{2} \end{cases} \tag{8}$$

The deviation also derives from a heuristic mixture of interpolation and direct measurement, except that more observations are required before the direct measurement is trusted.

$$\sigma_{ij} = \begin{cases} \mathcal{N}_{ij} \geq 10 & : \quad \sigma^*_{ij} \\ 5 \leq \mathcal{N}_{ij} < 10 & : \quad \frac{\mathcal{N}_{ij}-5}{5}\sigma^*_{ij} + \frac{10-\mathcal{N}_{ij}}{5}\bar{\sigma}^* \\ \mathcal{N}_{ij} < 5 & : \quad \bar{\sigma}^* \end{cases} \tag{9}$$

Here $\bar{\sigma}^*$ refers to the deviation between the model μ and all the observed character separations regardless of class, computed robustly by dividing the interquartile separation by 1.35. σ^*_{ij} derives similarly from the interquartile separation observed for each particular transition.

4 Experiments

The experiments presented below employ handwritten corpora that have been studied by other researchers. Initial testing of the system was carried out using the George Washington corpus. The identical system was then applied to excerpts from Terence's *Comedies* as a test of generality and for purposes of comparison with additional published research.

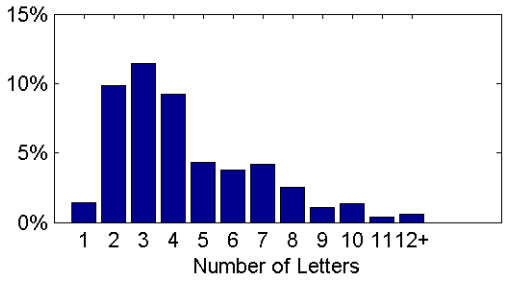

Fig. 6. Distribution of word lengths in the GW20 corpus.

4.1 George Washington's Letters

The George Washington corpus (GW20) comprises twenty pages of correspondence from the letters of George Washington. These are written in longhand script by several of Washington's secretaries, so they represent multiple handwriting styles. These experiments use the same word image segmentations as previous work [8]. The distribution of word lengths appears in Figure 6.

Previous experiments with the GW20 corpus [7,3,8,30] have employed a 20-fold cross-validation framework, with each page serving as a fold and the remaining 19 pages providing training word labels. Most prior work used holistic word recognition, and thus focused on the recognition accuracy for known words, since their *out-of-vocabulary* (OOV) recognition rate is zero. Adamek et al. report a top recognition rate of 83% for known words, but this represents only 69% of the entire sample including OOV words [7]. Since character-based recognition can identify both known and unknown words, the latter number makes the best figure for comparison.

The joint boosting process builds a letter detector as described in Section 2. Only two detectors are trained: one from the even pages and one from the odd pages. For testing any given page, the detector built without seeing that page is employed. There are sixty character classes total, including all lowercase

letters, numerals, most uppercase letters, and one instance of the British pound symbol £.

Tables 1 and 2 summarize the results under several experimental conditions. The first column of numbers in Table 1 shows the percentage of test examples for which the inference model's top character sequence matches the actual word tag. The second column of numbers shows the percentage of exclusively OOV words labeled correctly. The OOV words are more difficult to label because they include fewer short easy words. The third column shows the percentage of examples for which the most likely character sequence *of the correct length* matches the actual word tag. This number is a "cheating" experiment since it relies on knowing the actual word length, but the better performance here indicates at least that it may be worthwhile developing alternate methods to determine the correct word length. The fourth column shows the same figure for OOV words only. The last column shows the character error rate, computed from the edit distance between the prediction and the correct word divided by the total correct number of characters. In general, the accuracy for all categories improves with the number of training samples and the sophistication of the language model.

Table 2 offers numbers more directly comparable to previous work in holistic word techniques. Many of the character sequencer's erroneous predictions are not words at all, but are non-words similar to the correct label. The results in this table show the recognition rate with predictions constrained to the lexicon seen in the training set. If the unconstrained prediction from the character sequencer is not in the lexicon, then a post-analysis constructs a list of lexicon words closest in edit distance to the predicted word. The word from this list with the highest $\hat{\gamma}_m$ becomes the new prediction.

Table 1
Accuracy of predictions from the character sequencer: percentage of words recognized correctly. First column of numbers is the overall match rate; second considers only words not seen during training (out-of-vocabulary words). Third and fourth columns give equivalent results if true word lengths were somehow known *a priori*. Final column gives character accuracy rate. Deviations are computed over the 20 folds of the test set.

Experiment	Std.		+Len.		Char.
	[All]	[OOV]	[All]	[OOV]	Acc.
T32 bigram narrow	30 ± 7	12 ± 7	36 ± 7	17 ± 10	71 ± 4
T32 bigram broad	40 ± 8	18 ± 9	48 ± 8	26 ± 13	77 ± 4
T128 bigram narrow	53 ± 7	27 ± 10	63 ± 6	39 ± 10	81 ± 3
T128 bigram broad	54 ± 7	29 ± 10	64 ± 6	40 ± 9	82 ± 3
T128 trigram broad	62 ± 7	34 ± 11	70 ± 6	43 ± 11	86 ± 3

The list of candidates to check is generated via character insertion, deletion, and substitution operations, as well as swapping two characters for one or one for two. The latter two transformations are included due to the frequency of mistakes such as 'ii' in place of 'u' and vice versa. Restricting the word predictions to the training lexicon increases the percentage of correct labels significantly. On the other hand, as with prior work the chance of correctly labeling an OOV word goes to zero. A hybrid method described in Section 4.2 below overcomes this by choosing either the original prediction or the lexicon-constrained word depending on their relative score. This maintains the accuracy boost on known words afforded by the lexicon constraints while still allowing recognition of unfamiliar words when they are unambiguous enough. Results for this methods appear in the bottom two rows of Table 2.

Table 2
Accuracy with word label predictions constrained to vocabulary in the training lexicon. Results appear for in-lexicon words alone, for the entire word set, and (on the hybrid algorithm) for OOV words. Deviations are computed over the 20 folds of the test set.

Experiment	Lexicon	All	OOV Only
T32 bigram narrow	63 ± 7	53 ± 8	N/A
T32 bigram broad	74 ± 7	62 ± 7	N/A
T128 bigram narrow	77 ± 5	69 ± 5	N/A
T128 bigram broad	83 ± 4	70 ± 6	N/A
T128 trigram broad	84 ± 4	71 ± 6	N/A
Feng [30]	72.3	61.1	N/A
Adamek, et. al. [7]	83	69	N/A
Hybrid bigram	82 ± 4	72 ± 5	17 ± 8
Hybrid trigram	84 ± 4	76 ± 6	32 ± 10

4.2 Hybrid Open/Constrained Recognition

The word recognition rate using pure letter detection consistently lags behind the score with a constrained lexicon. Unfortunately, using only a constrained lexicon precludes correct labeling of any OOV terms encountered. For certain applications, such as document retrieval, these OOV words may be particularly interesting precisely due to their novelty and rarity.

Figure 7 displays the differences between raw recognition and the lexicon-constrained approach. Each point represents a word, with the position taken from the length-normalized likelihood $\hat{\gamma}_m$ of the top prediction of each method. The shaded area, found by Gaussian mixture modeling on a holdout set, denotes a region wherein the unconstrained prediction performs better. The characteristics of the identified area support the intuition that one should prefer the unconstrained result precisely when it has sufficiently high score.

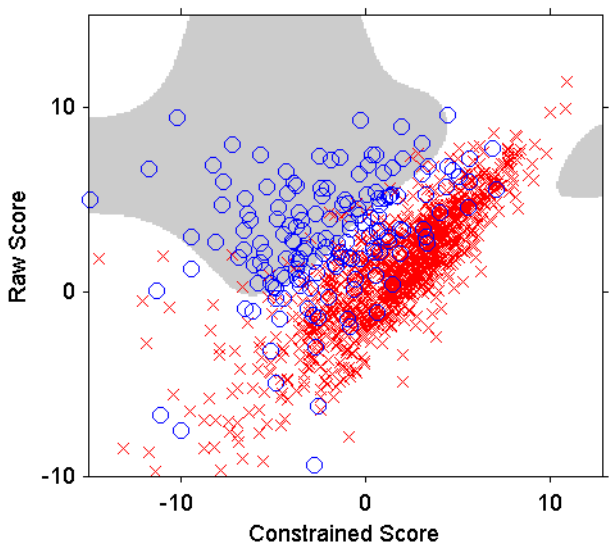

Fig. 7. Differences in scores between the raw and lexicon-constrained approaches. Circles show words only tagged correctly by the raw method, and X's show words only tagged correctly by the lexicon-constrained method. Light gray indicates areas where the former predominates, according to a Gaussian mixture model. For clarity, the figure omits words where both methods agree.

Using the holdout-trained mixture model to determine whether the raw or lexicon-constrained label should be chosen results in a highly successful hybrid algorithm. It correctly recognizes 32% of OOV words, 84% of lexicon words, and 76% of words over all, better than the previous best of 69% on this task [7]. The ability to recognize some OOV words while maintaining a high overall recognition rate distinguishes the character-based approach presented here. Note that the 32% rate for the hybrid algorithm only slightly lags the 34% rate for unconstrained recognition. As expected the OOV words recognized tend to carry content, with a median length of six characters.

4.3 Latin Results

Edwards et. al.[9,10] present recognition results for a handwritten Latin manuscript, Terence's *Comedies*. This document contrasts sharply with GW20 in style and

Table 3
Accuracy of predictions on the Latin manuscript. First two columns show percent of words correct by the basic system; third and fourth columns show equivalent results if true word lengths were somehow known *a priori*. Fifth and sixth columns show percent of words correct using constrained vocabulary. The final column gives character error rate over both folds. Results for both even and odd pages are shown.

Experiment	Std.		+Len.		Cnstr.		Char.
	Rec.	Ver.	Rec.	Ver.	Rec.	Ver.	Acc.
T128 bigram narrow	60	62	71	71	50	53	88.7
T128 trigram narrow	63	63	73	72	51	53	89.4
Edwards et. al. [9]	N/A	N/A	N/A	N/A	N/A	N/A	75

language. As a test of generality, the recognition system described above is retrained for the Latin text, without changing parameters. The original high-resolution document images are subsampled to approximate the letter size in the GW20 writing, but otherwise the processing is identical. These experiments are performed with only two folds, using the alternating *recto* and *verso* pages respectively.

The results appear in Table 3, for pages 5-47 of the *Comedies*. The basic system generalizes well to the new form of handwriting, achieving higher accuracy than prior work using similar data.[3] Only the constrained-vocabulary approaches do not generalize well. Because Latin includes multiple variant forms of the same word depending on case and gender, a dictionary based on simple word matching would require a much larger sample of training text. This could be addressed through language-aware matching, but the point of this exercise was to run the system with no changes.

[3] Because the ground truth used by Edwards et. al. is not available, a new transcript was prepared for the experiments performed here. Their result evaluates only 25 pages. Also, they constrain their method to employ just one example per character as initial training data.

5 Conclusion

This paper has developed a new approach to word recognition based upon unrestricted character detection followed by efficient inference on an ensemble of HMMs that vary in length. The character detection represents a new application of the joint boosting technique originally developed by Torralba et al. for finding objects in photographs [12]. To support word recognition in the context of multiple unsegmented and overlapping character detections, the paper also develops a novel inference framework applicable to noisy segmentation-free approaches. Inputs to the framework include a model of mean character separations estimated from sparse data, taken from inferred letter positions in a training corpus with human-provided transcription. When applied to offline historic document images of cursive script, the method described here improves on the best previously reported word recognition rates for the GW20 and Latin manuscripts, and demonstrates the ability to recognize words never seen during training.

The results presented here show great promise, with the possibility for additional gains from the application of well-established techniques not yet attempted. For example, incorporation of word-level bigram statistics in other work improved recognition rates by up to 10% [1], and similar measures could be applied here. Further effort in assembling a comprehensive and complete character training set might also yield significant improvement, as would more accurate choice of the correct word length in the inference model. Finally, experimenting with other sorts of features besides the histogram of gradients may turn up feature sets even better suited to character recognition. Exploration of the possibilities has only begun.

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

[2] T. M. Rath, R. Manmatha, V. Lavrenko, A search engine for historical manuscript images, in: 27th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 369–376.

[3] N. Howe, T. Rath, R. Manmatha, Boosted decision trees for word recognition in handwritten document retrieval, in: 28th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 377–383.

[4] R. Plamondon, S. N. Srihari, On-line and off-line handwriting recognition: A comprehensive survey, IEEE Trans. Pattern Anal. Mach. Intell. 22 (1) (2000) 63–84.

[5] A. Vinciarelli, A survey on off-line cursive script recognition, Pattern Recognition 35 (7) (2002) 1433–1446.

[6] S. Madhvanath, V. Govindaraju, The role of holistic paradigms in handwritten

word recognition, IEEE Trans. Pattern Anal. Mach. Intell. 23 (2) (2001) 149–164.

[7] T. Adamek, N. E. O'Connor, A. F. Smeaton, Word matching using single closed contours for indexing handwritten historical documents, International Journal of Document Analysis and Recognition 9 (2007) 153–165.

[8] V. Lavrenko, T. Rath, R. Manmatha, Holistic word recognition for handwritten historical documents, in: Proc. IEEE Workshop on Document and Image Analysis for Libraries, 2004, pp. 278–287.

[9] J. Edwards, Y. W. Teh, D. Forsyth, R. Bock, M. Maire, G. Vesom, Making latin manuscripts searchable using gHMM's, in: Advances in Neural Information Processing Systems 17, 2005, pp. 385–392.

[10] J. Edwards, D. Forsyth, Searching for character models, in: Advances in Neural Information Processing Systems 18, 2006, pp. 331–338.

[11] M. Decerbo, P. Natarajan, R. Prasad, E. MacRostie, A. Ravindran, Performance improvements to the bbn byblos ocr system, in: Proc. 8th Int. Conf. on Document Analysis and Recognition, 2005, pp. 411–415.

[12] A. Torralba, K. P. Murphy, W. T. Freeman, Sharing visual features for multiclass and multiview object detection, IEEE Trans. Pattern Anal. Mach. Intell. 29 (5) (2007) 854–869.

[13] U.-V. Marti, H. Bunke, Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition system, Int. J. Pattern Recognit. Artif. Intell. 15 (1) (2001) 65–90.

[14] T. Crackel, Private communication.

[15] T. Rath, R. Manmatha, Word spotting for historical documents, International Journal of Document Analysis and Recognition 9 (2-4) (2007) 139–152.

[16] S. Feng, R. Manmatha, Exploring the use of conditional random field models and hmms for historical handwritten document recognition, in: Proc. 2nd IEEE Int. Conf. on Document Image Analysis for Libraries, 2006, pp. 30–37.

[17] G. Mori, J. Malik, Recognizing objects in adversarial clutter: Breaking a visual captcha, in: Computer Vision and Pattern Recognition, Vol. 1, 2003, pp. 134–141.

[18] I. Bar-Yosef, I. Beckman, K. Kedem, Binarization, character extraction, and writer identification of historical Hebrew calligraphy documents, International Journal of Document Analysis and Recognition 9 (2007) 89–99.

[19] K. Ntzios, B. Gatos, I. Pratikakis, T. Konidaris, An old greek handwritten OCR system based on an efficient segmentation-free approach, International Journal of Document Analysis and Recognition 9 (2007) 179–192.

[20] S. Impedovo, Hidden markov models in handwriting recognition, in: S. Impedovo (Ed.), Fundamentals in Handwriting Recognition, 1994, pp. 7–39.

[21] S. Feng, R. Manmatha, A hierarchical, hmm-based automatic evaluation of ocr accuracy for a digital library of books, in: ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'06), 2006, pp. 109–118.

[22] J. He, D. M. Q. Do, A. C. Downton, A comparison of binarization methods for historical archive documents, in: 8th Int. Conf. on Document Analysis and Recognition, Seoul, Korea, 2005, pp. 538–542.

[23] E. Learned-Miller, Data-driven image models through continuous joint alignment, IEEE Trans. Pattern Anal. Mach. Intell. 28 (2) (2006) 236–250.

[24] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: Computer Vision and Pattern Recognition, 2005, pp. 886–893.

[25] A. Vedaldi, P. Favaro, E. Grisan, Boosting invariance and efficiency in supervised learning, in: International Conference on Computer Vision, 2007,

pp. 1–8.

[26] L. Wolf, I. Martin, Regularization through feature knock out, Tech. Rep. AIM-2004-025, MIT (2004).

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

[28] D. Mease, A. J. Wyner, A. Buja, Boosted classification trees and class probability/quantile estimation, Journal of Machine Learning Research 8 (2007) 409–439.

[29] F. Jelinek, Statistical Methods for Speech Recognition, MIT Press, Cambridge, MA, 1997.

[30] S. Feng, Statistical models for text query-based image retrieval, Ph.D. thesis, Uinversity of Massachusetts, Amherst (May 2008).