Searching Off-line Arabic Documents

Jim Chan
jdchan@gmail.com

Celal Ziftci
cziftci@uiuc.edu

David Forsyth
daf@uiuc.edu

University of Illinois
201 N. Goodwin Ave, Urbana, IL 61801

Abstract

Currently an abundance of historical manuscripts, journals, and scientific notes remain largely unaccessible in library archives. Manual transcription and publication of such documents is unlikely, and automatic transcription with high enough accuracy to support a traditional text search is difficult. In this work we describe a lexicon-free system for performing text queries on off-line printed and handwritten Arabic documents. Our segmentation-based approach utilizes gHMMs with a bigram letter transition model, and KPCA/LDA for letter discrimination. The segmentation stage is integrated with inference. We show that our method is robust to varying letter forms, ligatures, and overlaps. Additionally, we find that ignoring letters beyond the adjoining neighbors has little effect on inference and localization, which leads to a significant performance increase over standard dynamic programming. Finally, we discuss an extension to perform batch searches of large word lists for indexing purposes.

1. Introduction

Offline handwriting recognition is a traditional topic in computer vision, with a revived interest as improved learning methods become available [6, 12, 3]. However, some writing systems, such as Arabic, present major challenges for handwriting recognition. There is a small body of work [1], but few involving large datasets or vocabularies. Pechwitz and Maergner [11] introduce a fairly extensive form database, but the majority of it is used as supervised training data. We describe a system to search printed and handwritten Arabic documents using relatively little training data, whose performance is comparable to modern searches of carefully handwritten Roman letters.

1.1. Why Arabic is hard

The written Arabic system used today evolved from a dialect of Aramaic, which had fewer phonemes than Arabic. The original letter forms adopted from the Aramaic language had to represent 28 different sounds using only 15 characters. Eventually, additional letters were formed by adding dots above and below existing letters to disambiguate the script; several characters thus differ by a single dot, making discrimination hard. Additionally, Arabic is a consonantal writing system where vowels are frequently omitted; the reader must have some understanding of the language to restore them. In vocalized or sacred texts, *diacritics* - small marks indicating short vowels, are added to facilitate pronunciation of the word, and can be confused with dots, posing another barrier to recognition.

Additionally, Arabic letters change form based on their position in a word. Almost all letters can appear in 4 different forms: initial, medial, final, or isolated. The differences between forms are generally quite dramatic. (Fig. 1 inset)

Writers frequently elongate characters for aesthetic reasons or to justify text. These elongations appear as extensions of the baseline between characters, or lengthening of hooks, often creating vertical overlaps with neighboring characters. Many letter combinations form ligatures, changing their appearance completely. The dots which are necessary for discrimination in transcription frequently end up above neighboring letters, an artifact of fast writing. (Fig 1)

2. Preprocessing

The manuscript pages were first converted to gray-scale and inverted so that the background value was close to 0, and thresholded to remove background noise. The images were then roughly cropped to remove margins, etc. Pages were scanned carefully enough such that a rough cropping could be achieved using the same rectangular regions for every image. The cropped images were then rectified to correct for page skew; details in Section 2.1. After de-skewing, each page was segmented into a series of lines. During this stage, overlaps and diacritical marks were removed; details in Section 2.2.

Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)
0-7695-2597-0/06 $20.00 © 2006 IEEE

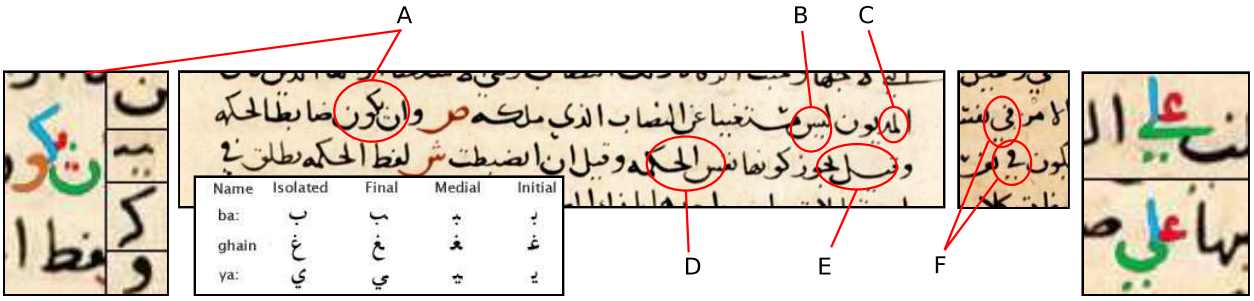

Figure 1. **Difficult features in handwritten Arabic.** A) 4 characters which have been written in such a way as to be nearly impossible to segment (see detail on far left) B) dot association is not clear; they belong to the ب but end up underneath the س. C) the small stub in front of the vertical ل is in fact the letter ه. D) ل forms a ligature with ح, 'لح'. Note also the overlaps caused by the hook of the س. E) gives an example of stylistic elongation of letters. F) two different forms of في. They are the exact same word, except written completely differently. **Table Inset:** Letters in Arabic have up to 4 different forms, depending on the position in the word. Note that the appearance of a letter can change quite drastically, and the difference between two completely different letters can be as subtle as a single dot. **Far Right:** Different forms of the word علي, note in the top version it is connected to another word.

2.1. Page Rectification

If we project ink pixels onto the y-axis, pages with no skew will yield row sums of lowest entropy. Any rotation of the lines will scatter the pixel densities into the gaps, increasing the entropy of the confi guration. Thus, we compute the entropy at various angles using a golden section search between -5 and 5 degrees; each iteration searches over the subregion with minimum entropy until the angular search range is within $\epsilon = .01$ degrees.

2.2. Diacritics, Overlaps, and Line Segmentation

Diacritical vowel marks proved to be a nuisance in initial printed experiments because they drastically altered the appearance of letters in our system. The discussion in Section 3.3 and Fig. 6 explain why this occurs. Since Arabic is a consonantal language, we simplifi ed the printed text by removing the diacritics; this was done by binarizing the image, then thresholding connected components within some size and orientation range. Orientation is measured by pixel height-to-width ratio. This technique was not used in our handwritten experiments because the document rarely used diacritics.

The handwritten data has the additional nuisance of hooks frequently overlapping underneath neighboring letters, several instances of which can be seen in Fig. 2. We correct this in most cases by ensuring that bounding boxes of connected components do not overlap. The algorithm is as follows:

Binarize the image, compute the connected components, and their bounding boxes. Then scanning from right to left, check for intersecting bounding boxes, if they overlap, shift all components after the fi rst bounding box left until the overlap is removed. Note that the binarized image is only used to fi nd the bounding box of the components. Shifts are performed in the original image.

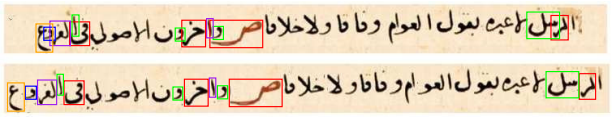

Figure 2. Hooks overlapping neighboring letters can cause dramatic changes in appearance when rectifi ed. In many instances, such overlaps can be removed quite easily by ensuring the bounding boxes of connected components do not overlap.

Example output can be seen in Fig. 2

The line fi nder and overlap removal are conveniently integrated. To segment the lines, we compute the centroid of each bounding box as (x, y), where x is the horizontal geometric center, and y is the average baseline height in the box. The centroids are projected onto the y-axis and clustered to associate the bounding boxes into line clusters. We opted to take this approach because segmentation using straight lines produced poor results due to wavy writing. This method produced satisfactory segmentations: out of the 420 lines segmented for the handwritten experiments, 9 lines exhibited small errors due to excessive descenders connecting to the line below. The errors typically only involve a single character per line, so they did not have signifi cant affect on overall search performance.

3. Modeling Ink

Several recent and earlier works choose to model ink at the word level [8, 2, 7] by directly computing $P(W|I)$, the likelihood of word W given a segment of ink I. It has been shown that word spotting techniques can do quite well at recognizing entire words even when some of the letters are ambiguous[8]. We modeled words at the letter level, where

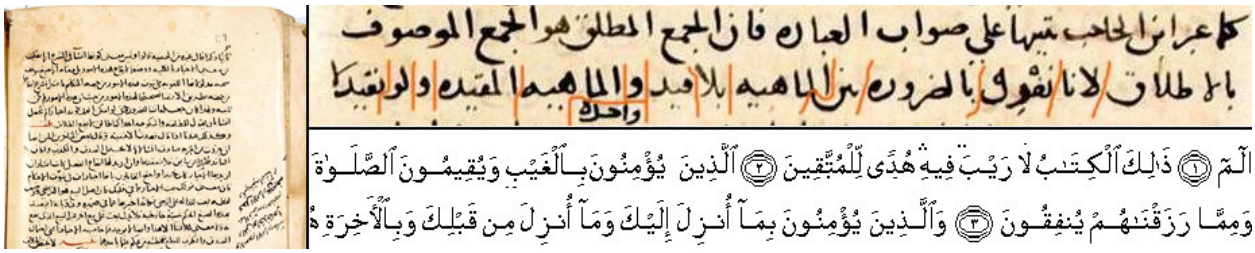

Figure 3. **Examples of the data. Left:** A page from the manuscript Kitab fi 'l-fi qh (كتاب في الفقه); note the writing in the margins. **Top:** Sample handwritten lines; note that it is not possible to fi nd word boundaries with conventional methods. Note also the words written underneath others. **Bottom:** Sample printed lines from the Qur'an; diacritics and verse markers can be seen.

the likelihood of a word given a segment of ink is:

$$P(W|I) = P(l_0) \cdot \max_{\alpha_0}\{P(l_0|I_{(0,\alpha_0)}) \cdot P(\alpha_0|l_0)\} \cdot \prod_{l_i \in W} \max_{\alpha_i}\{P(l_i|I_{(s_i,s_i+\alpha_i)}) \cdot P(\alpha_i|l_i)\} \cdot \prod_{l_i \in W} P(l_i|l_{i-1}) \quad (1)$$

where the l_i's are the letters and α's are the possible column widths of the letter, and s_i is defi ned as $s_i = \sum_{j=0}^{i-1} \alpha_j$.

The closed vocabulary implied by word spotting is inconvenient for highly inflected languages such as Arabic. Word frequencies obey the power law: the vast majority of words have extremely low frequencies, making it diffi cult to fi nd examples of them. In contrast, letter frequencies are far more uniform, making language modeling easier and more robust (in terms of estimation), training examples are easier to obtain, and we reduce the number of classes to discriminate among drastically. Finally, note in Fig. 3 that whitespace reveals little information about word boundaries in handwritten Arabic, which would be troublesome to segmentation stages used in works such as [8].

In our approach, the segmentation stage is completely integrated with recognition. The line is divided into small, uniform columns, which are then grouped during inference to form the correct transcription. Thin letters such as 'ا' are composed of 2-4 columns, whereas letters like 'ف' are composed of 12-17 columns. The only parameters that need specifi cation are the width of the columns in pixels (chosen such that most letters span 15 columns), and a single σ that governs the column width variance for each letter. Transcription/search performance is fairly insensitive to these parameters; they serve only to optimize the size of the search space. We will see that it is also possible to take advantage of the local independence of the ink to further reduce the search space.

3.1. HMM Structure

Inference is performed using an HMM, in which each of the hidden states can transition across multiple columns. The HMM structure is quite similar to the gHMM used in Edwards [6]. Each state is labeled as a triplet, (b, e, c), where b and e are the column indices, and c is hidden character variable. Each state thus infers a single character at a given position and width in the line. Paths through this trellis structure will yield sequences of triplets, which give possible transcriptions of the line.

The structure of the HMM can be seen in Fig. 4. An exaggerated line of ink is given at the top. Notice that the line is segmented into equal sized columns of ink. Just below, we have a condensed version of our HMM. Each circle in this condensed drawing actually represents $|\alpha| \cdot |C|$ states, where $|\alpha|$ is the number of possible widths (in columns) which can be emitted per time step, and $|C|$ is the cardinality of the alphabet. Here, $|\alpha| = 3$ - in our implementation $|\alpha|$ is roughly between 8 for letters with little horizontal variation, to 12 for letters with long hooks for example. The range of widths here is $[1, 3]$, meaning the states can transition 1, 2, or 3 states in the future. In our actual model, the ranges are typically $[5, 15]$ ($|\alpha| = 11$), meaning it is possible to move between 5 and 15 columns forward.

In the fi gure, transitions leaving each state are a different shade of grey, depending on how many columns of ink the state emits. This can be seen more clearly in the expanded version at the bottom (C). The fi rst row of states all emit a single column, the second row two columns, and the third row three. The hidden variable at any state in the expanded view is the character, C. Thus, each column of states has $|\alpha| \cdot |C|$ possible triplets.

3.2. Transitions

The letter transitions correspond to $P(l_i|l_{i-1})$ in Equation 1. The transition matrix has a banded structure, as most transitions have zero probability (cannot go from state $(1, 2)$ to $(4, 5)$). The non-zero entries in the matrix are composed of the bigram probabilities computed between each letter pair in the alphabet. Simple additive smoothing was used, since letter bigrams are relatively abundant.

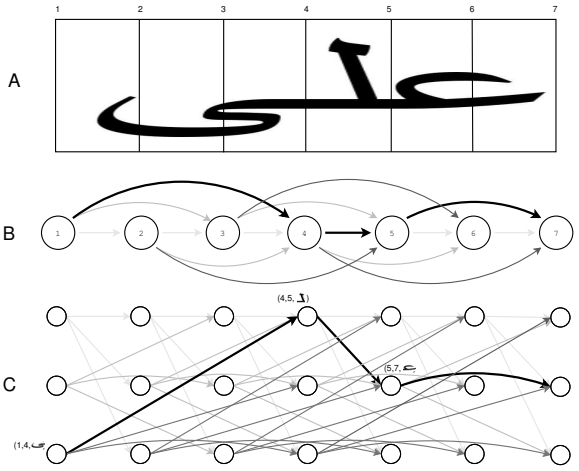

Figure 4. **A diagram of the HMM structure.** Note that transitions span multiple columns, and each small circle in the expanded view (bottom) represents $|C|$ possible letters in the alphabet. We can obtain both the segmentation and the transcription simultaneously using dynamic programming on this trellis. The bold path gives the segmentation and transcription.

3.3. Emissions

At every state in the HMM, we compute $P(l_i|I_{(s_i,s_i+\alpha_i)})$ using a nearest neighbors discriminative model. From previous experiments, it was found that simple nearest neighbors in the original image space was not good enough to discriminate between letter classes reliably. Instead, we project bits of ink into a high dimensional non-linear space using kernelized principle components analysis (KPCA) [13], followed by linear discriminant analysis (LDA) [4] to find a discriminative subspace. Auxiliary features such as height, width, and relative baseline offset are thrown in between the KPCA and LDA stages. The Mahalanobis distance metric is used to account for non-uniform scatter. Fig. 5 shows the process of converting ink into a feature vector.

Vertical bounds of the ink are first computed by finding the first and last zeros of the second derivative of the row sums. The cropped ink is then resized into a 20×20 grayscale image. Horizontal and vertical pixel differentials are computed. The resulting positive and negative pixels are separated into two channels and normalized so their values range between $[0, 255]$. The original image is then augmented with these filtered copies. Next, the input vector is transformed into feature space via KPCA using a gaussian kernel. After projection into feature space, auxiliary features such as bounding box height and width, and relative baseline height are appended. Baseline is computed similarly to [14], except regression fitted to a 4th degree polynomial to account for waviness in the scribe's writing. Finally, LDA is used to project the resulting vector to a more discriminative subspace.

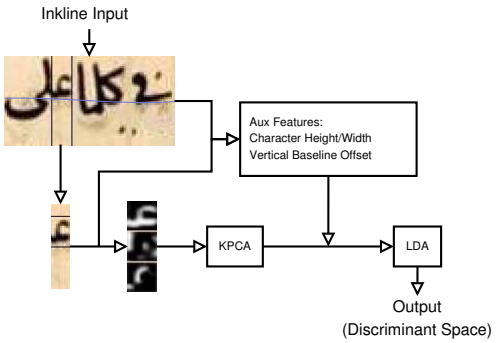

Figure 5. Given a column of ink, we compute the vertical bounds via row sum derivatives and rectify the resulting region to a 20×20 square. Directional filters are then applied to create the feature vector in image space. After projecting the vector to a lower dimensional space, extra features computed from previous stages are added in. LDA is then used to further project the vector down to the most discriminative subspace.

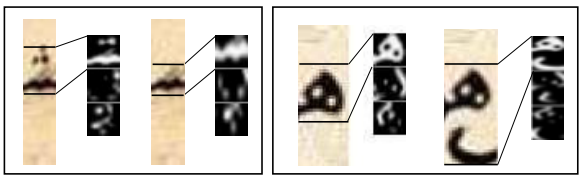

Figure 6. **Left:** The presence or absence of dots can change the appearance of the ink quite drastically after rectification. This helps when discriminating between characters with dots and those without, but can also be a drawback when the dots are missing or written in the wrong place. **Right:** As mentioned earlier, removing overlaps is quite important as they can have large effects on rectification.

The KPCA/LDA module is trained using manually extracted templates. Each letter class is estimated as a gaussian in discriminant space. Training the discriminant model is just a matter of building the kernel and linear discriminant transformations, and estimating the gaussian by computing the class means and variances after the templates have been projected. Since most classes have fewer than 10 training examples, it is not feasible to estimate the variance of each class - thus a single estimate is used for all classes.

New ink is converted into discriminant space using the previously computed transformations. We compute the unnormalized score d_i as the Maholonobis distance plus the log of the prior:

$$d_i(x) = -\frac{1}{2}(x-\mu_i)^t\Sigma^{-1}(x-\mu_i) + lnP(l_i) \qquad (2)$$

4. Transcription

Transcription is performed using the viterbi algorithm. While producing a high quality transcription is not a main goal of this work, this simple method produces a good

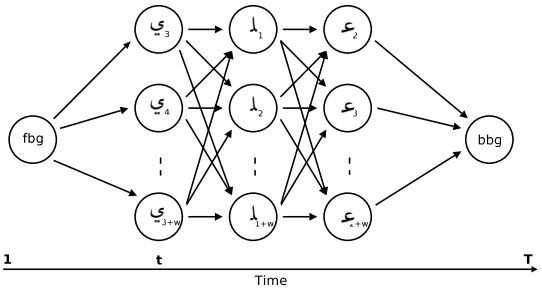

Figure 7. DP structure for searching. Many of these edges can be removed if we perform a beam search with one-character lookahead.

enough approximate transcription so that we can localize search results properly. This is discussed in Section 5.

We implemented a few optimizations: for each time step t, we prune all states that are not within some threshold of the maximum likelihood state at t. Additionally, every time we propagate partial probabilities forward, we will loop over all $|\alpha| \cdot |C|$ outgoing edges. We can prune the number of outgoing edges by limiting the range of widths for each letter as discussed earlier. The range of widths for each letter is determined from the training examples by taking all column widths within 1 stdev. of the mean.

5. Search

Search involves examining all confi gurations of the word at all positions t. The number of confi gurations is exponential with the length of the word, but we can use dynamic programming to reduce the complexity to $O(|\alpha| \cdot |W|^3)$ states, where $|W|$ is the number of characters in the search term. Fig. 7 gives an illustration of how search is performed for the word. There are a few things to note about this fi gure:

The 'fbg' token represents the sequence generated by the transcription model up to time t. If the path chosen goes through the states $\text{ي}_a, \text{ا}_b, \text{ع}_c$, the token 'bbg' represents the sequence given by the transcription model from time $t+a+b+c$ to the end of the line.

The 3 columns have different subscripts which denote the different set of width ranges. $|\alpha|$ is not necessarily the same for each letter, since some letters have more horizontal variance than others. The letter 'ا', for example, is a single vertical stroke, and thus has little variation.

The most likely position of the word in the line will be given by the path that maximizes sum of the partials at 'fbg', 'bbg', and the partial from the traversal through the intermediary states.

We should note that 'fbg' and 'bbg' are only used to localize the search term; the score used for ranking the lines will be determined purely from the local emission and bigram transitions through the intermediary states.

We can in fact do better than DP by utilizing a beam search. It is important to realize that a completely greedy search with no look-ahead will fail. Ligatures and abnormal letter forms causing non-optimal letter boundaries at any point will cause all subsequent letters to localize incorrectly. By taking maximum likelihood bigrams, however, we can always fi nd the optimal letter boundaries. The left boundary is assumed to be optimally determined by construction. Consider now the bigram formed by the current character and the one directly following. If we have a strong model for at least one of the characters, then the boundary can be established reliably. If both are poor, then even if we search exhaustively we would not be able to fi nd a better boundary between these two letters. Note that the fi rst and last boundaries are formed by pairing the forward partial at the end of 'fbg' and the fi rst character, and the backward partial at the beginning of 'bbg' with the last character. Since we have to look at all combinations of widths for each bigram, we come up with the complexity of $O(|\alpha| \cdot |W|^2)$.

We build a trie structure to represent the words, and keep the current score and position at each node. At terminal nodes, we also keep the current best score and word confi guration. Traversing the trie at each time step will search all words simultaneously. Using this extension we were able to search word lists with tens of thousands of words, which can be used to support indexing or possibly transcription by search.

6. Results

While ideally we would like to have high performance transcription, in our view it is not always as useful as a high performance search, especially if we are dealing with large quantities of low-value documents. We show in the following results that even with relatively low quality transcriptions, we can still achieve very good search results using little training data.

6.1. Printed Arabic

The fi rst dataset we tried as a printed copy of the Qur'an that was generated using computer fonts. This dataset has little to no variation between different instances of letters, so it was a reasonable starting point before trying handwritten datasets. We used simple template matching to fi nd the verse markers in the data so we could align the images with an electronic copy of the Qur'an. We ran the thresholding mechanism described in Section 2.2 to remove diacritics to reduce the number of appearances for each letter.

To train our emission model, we took a few verses and segmented them by hand using a custom GUI tool designed for this purpose. There are 100 different types of letters that we had to search for, several of which are quite infrequent. In addition to the hand segmented examples, we

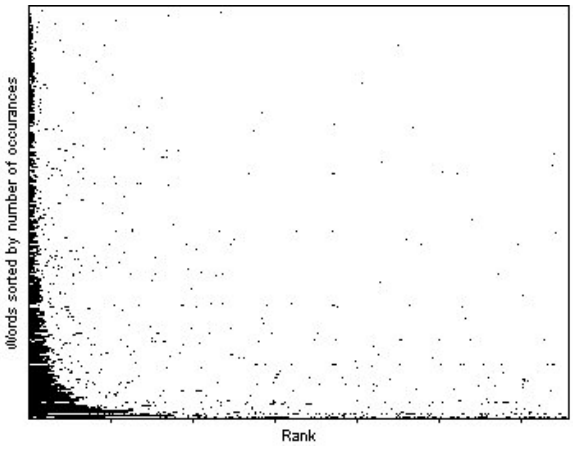

Figure 8. **Printed Arabic Search Results on the 250 most common words.** The results for ranking of 350 lines are given. The search words are sorted along the y-axis by the number of times they actually occur in the test corpus. The lines are sorted by their search score along the x axis. A black pixel indicates the given search term actually appears in the given line, white otherwise. An ideal search would result in an image where all the black pixels are packed to the left.

cut out templates from an image of the Arabic alphabet to cover the letters which we did not easily find in the text. The fonts used in the text and the alphabet image are not the same, but our hope is that they are close enough that they serve as a good enough approximation. This also adds some more variance to our samples. It seems especially reasonable since the letters which we could not find occur so infrequently that they would rarely affect our search results.

Letter classes with less than 6 examples were augmented with a set of artificially jittered copies. Jittered examples were created by performing small rotations and shifts, and adding gaussian noise to the copies of the original examples. The size of the final kernel matrix was 1423×1434.

The bigram probabilities were computed from a text with around $400,000$ characters. The model was then adjusted to retain zero probability transitions between impossible bigrams, and to accommodate transitions to a special token which denoted character elongations. Essentially, the transition to this special token had a probability of 1 for medial and final characters and 0 for everything else, so we relied completely on image cues to transition to this state. The corresponding unicode characters which represent these elongations in the ground truth were deleted before evaluating transcription performance.

We arbitrarily chose 350 verses from the Qur'an to perform our experiments on. All diacritic symbols were deleted from the ground truth. The Unicode standard does not encode the form of the characters (ie, medial, isolated, etc), thus a finite state machine was used to reconstruct this information in the ground truth.

The initial results for printed Arabic were quite strong. Transcription accuracy was 88.1%, which was surprising given the simplicity of our language model. Studies on current Arabic OCR [10] has shown that leading commercial products achieve a performance of about $87-90\%$ for most printed documents. Unfortunately, we didn't have access to such systems to make a true comparison.

There are a few things which could be done to improve this score. As mentioned in Section 2.2, the removal of diacritics make discrimination easier. The diacritical form of ا was removed during this preprocessing, however they were represented using the standard ا in the downloaded text. The notation difference accounts for up to 2% of edit errors.

The biggest failure during transcription was the inability to deal with the compulsory ligature 'لا'. These ligatures account for nearly 3.4% of the characters, and since in this initial experiment we had no templates for them, we were pretty much guaranteed to make mistakes on them. In our handwritten experiments, we built models for common ligatures as well because they were even more pervasive.

Fig. 8 shows search results for the 250 most common words in the 350 lines used to test transcription and search. Precision and recall plots can be found in Fig. 9 for selected words. In this experiment we had no model for ligatures, even obligatory ones like 'لا', which affects results for both search and transcription (see Fig. 9 for more detail). Another factor which affected our search was that, unlike in transcription, we did not implement anything to take stylistic elogations into consideration. In Fig. 10, we give the top 4 results returned, along with the last 4 true positives, which reveals that this phenomenon caused the low rankings.

6.2. Handwritten Arabic

We were not able to find handwritten copies of the Qu'ran that were not either written so carefully as to look printed, or written in an overly decorative style. Instead, we chose 20 pages from *Kitab fi 'l-fi qh* (كتاب في الفقه), a 12th century document on Islamic Jurisprudence, found at Jafet [9]. It should be noted that our document, as typical of ancient manuscripts, was still fairly carefully written. However, as one can see in figures 1 and 3, the data is certainly not without irregularities or variation. Templates were extracted from various regions of the 420 segmented lines to train the emission model. In all, 878 letter samples were used for training. The 420 transcribed lines contain 21,776 characters. Jittered templates were not used in this experiment because the method we used in the printed experiments seemed to be too mechanical to model the variation in the hand of a scribe. A more sophisticated jittering model may be worth investigating, however, because from the previous experiment it proved to be a cheap method of extending the training set.

As mentioned earlier, the previous experiment made it

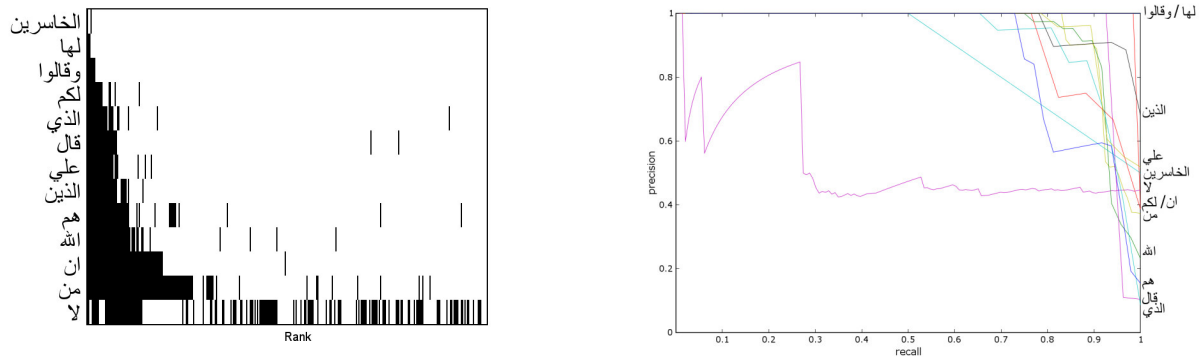

Figure 9. **: Precision Recall for selected Arabic words (printed).** On the left are the search results for selected Arabic words (see Fig. 8 for explanation). On the right are the corresponding precision recall curves. The last search term,'لا', is a ligature for which we had no template in this experiment. Using standard templates it appears as 'ل', which explains why we will not be able to find it in the actual document. However, since it occurs in nearly every other line, recall is not zero, which is somewhat misleading. For the other cases though, we obtain quite good results.

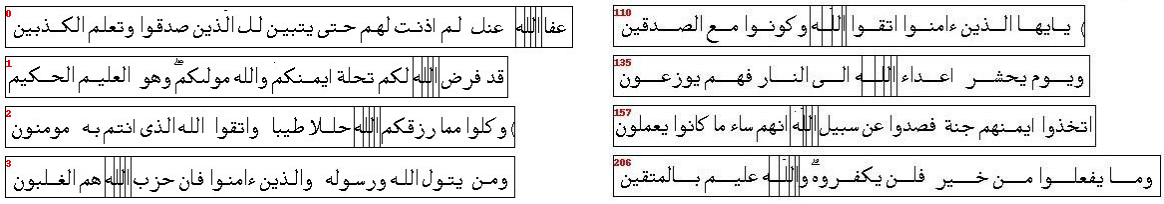

Figure 10. **Search results for the Arabic word, 'ألله'.** On the left are the top four lines returned. On the right we have the last four true positives. Their corresponding rankings can be seen in the upper left corner. The low ranking examples were caused by noise and stylistic elongation to justify the text.

readily apparent that we would need models for ligatures in addition to the stand alone letters. We made models for 85 of the 100 Arabic letter forms, and added models for 12 common ligatures. It turned out that ligatures were also easy to discriminate due to their complex appearances, further increasing their importance. The same data as before was used to train the language model, except converted so that ligature transitions could be modeled.

Transcription performance was quite low at only 24.7%. Since our language model has zero probability transitions for impossible bigrams, the mistakes in discrimination tended to snowball: many times the transition back to the correct sequence was impossible. A stronger language model is likely needed; however just giving the impossible transitions some non-zero likelihood may have improved the transcription rate signifi cantly.

Fortunately, in search the term itself imposes the correct letter ordering, so performance is much better. We built a word list containing all words in the manual transcription; with a total of 2133 unique words. Only words with 4 or more occurrences are reported here for brevity. A summary of our search results can be found in Fig. 11. Fig. 12 shows some of the words we searched for. Using the trie extension we described earlier, we were able to search a wordlist composed of the 20k most frequent words from modern news texts. This shotgun approach allowed us to build a reasonable index of the document without actually knowing its contents; an online demo can be found at **`http://handwriting.qwef.org`**.

6.3. Discussion

Our method compares well to modern searches of carefully handwritten roman letters; this is most likely because we have a carefully engineered discriminative character model. Future work will include determining which groups of stacked characters need individual templates; which variant letters need individual templates; automated searches for useful character template training information (after [5]); methods to transcribe a document using search against large external word list; and studies on larger datasets.

References

[1] A. Amin. Off line arabic character recognition - a survey. In *ICDAR*, pages 596–599, Washington, DC, USA, 1997.

[2] A. Amin. Structural description to recognising arabic characters using decision tree learning techniques. 2002.

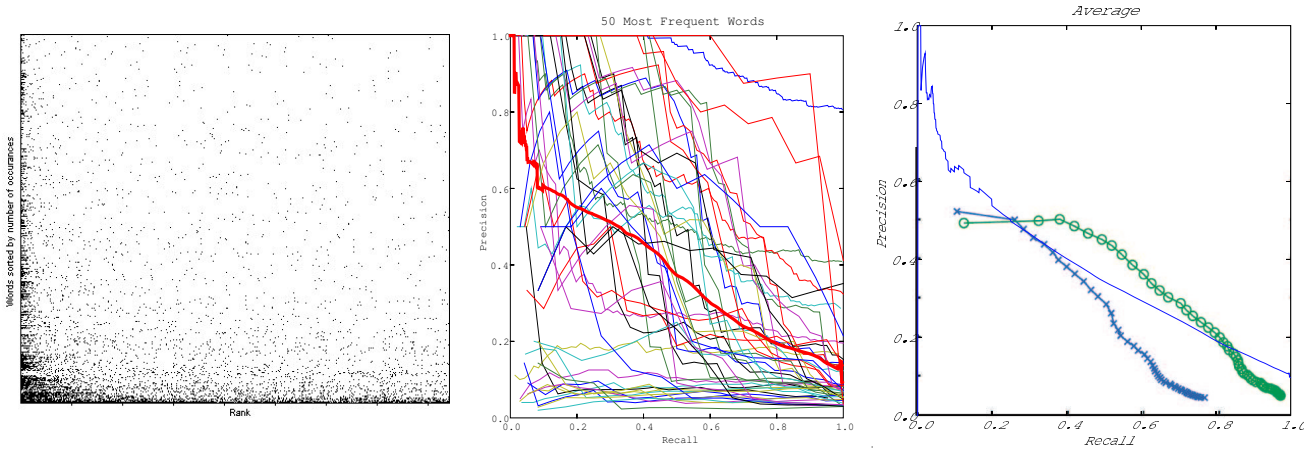

Figure 11. **Left:** Search results for handwritten Arabic words with 4 or more occurrences (see Fig. 8 for explanation). This represents 360 words. **Center:** Precision recall curves for the 50 most frequent words in the document. The thick red line represents the average curve. **Right:** The thin line represents our average precision recall for words with 4 or more occurrences. Compare this to the average precision recall curves given in Edwards [5] (marked with x's and o's) which have been manually overlaid. Our performance on handwritten Arabic is quite comparable to their performance on handwritten Latin.

[3] C. Burges, J. Ben, J. Denker, Y. LeCun, and C. Nohl. Off-line recognition of handwritten postal words using neural networks. In *IJPRAI*, volume 7, page 689, 1993.

[4] R. O. Duda and P. E. Hart. *Pattern Classifi cation and Scene Analysis*. Wiley, New York, 1972.

[5] J. Edwards and D. Forsyth. Searching for character models. 2005.

[6] J. Edwards, Y. W. Teh, D. Forsyth, R. Bock, M. Maire, and G. Vesom. Making latin manuscripts searchable using ghmm's. *NIPS*, 2005.

[7] T. K. Ho, J. J. Hull, and S. N. Srihari. A word shape analysis approach to lexicon based word recognition. In *Pattern Recognition Letters*, volume 13, pages 821–826, 1992.

[8] R. Manmatha, T. M. Rath, and V. Lavrenko. Holistic word recognition for handwritten historical documents, Apr. 29 2004.

[9] A. A. Manuscripts. Kitab fi 'l-fi qh.*American University of Beirut*, `http://ddc.aub.edu.lb/projects/jafet/`.

[10] G. A. Marton, O. Bulbul, and T. Kanungo. Performance evaluation of two arabic OCR products, Dec. 08 1998.

[11] M. Pechwitz and V. Maergner. HMM based approach for handwritten arabic word recognition using the ifn/enit-database, Aug. 13 2003.

[12] R. Plamondon and S. N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *TPAMI*, 2000.

[13] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical Report 44, Dec. 1996.

[14] A. W. Senior and A. J. Robinson. An off-line cursive handwriting recognition system. *TPAMI*, 1998.

a)

b)

c)

Figure 12. **Results returned for three handwritten words.** a) The fi rst 5 lines returned for 'الاجماع'. Note that ه is barely visible. b) the fi rst 4 lines for 'الاحكام'. Notice in the last line even though the 'لا' has a different form and is split apart from the overlap removal process, the strong models for the other letters provide a strong enough cue to retrieve the words. c) 'دليل'. Note the connected ا in 1, and the elongated spacing in 3.