Combining different classification approaches to improve off-line Arabic handwritten word recognition

Ilya Zavorin, Eugene Borovikov, Ericson Davis, Anna Borovikov, Kristen Summers
CACI, Knowledge and Information Management Division
4831 Walden Lane, Lanham, MD 20706, USA
{izavorin,yborovikov,eridavis,aborovikov,ksummers}@caci.com

ABSTRACT
Machine perception and recognition of handwritten text in any language is a difficult problem. Even for Latin script most solutions are restricted to specific domains like bank checks courtesy amount recognition. Arabic script presents additional challenges for handwriting recognition systems due to its highly connected nature, numerous forms of each letter, and other factors. In this paper we address the problem of offline Arabic handwriting recognition of pre-segmented words. Rather than focusing on a single classification approach and trying to perfect it, we propose to combine heterogeneous classification methodologies. We evaluate our system on the IFN/ENIT corpus of Tunisian village and town names and demonstrate that the combined approach yields results that are better than those of the individual classifiers.

Keywords: offline handwriting recognition, classifier combination, Arabic script, word recognition

1. INTRODUCTION
Machine perception and recognition of handwritten text in any language is a difficult problem. Even for Latin script, where the problem is fairly well studied and the state of the art is more mature than for Arabic script, most solutions are restricted to specific domains (e.g., mail sorting, bank checks courtesy amount recognition, tax forms\textsuperscript{1–3}) and have significant limitations on lexicon size and recognized writing style.

1.1. Background
The difficulty of general handwriting recognition stems primarily from the large amount of variability that exists in handwritten text. Given glyphs of two characters, one can readily observe the following types of differences:

- topological and geometric differences between glyphs, which provide a basis for recognizing and distinguishing characters
- inter-writer differences in the production of the same glyphs, which are roughly analogous to font differences in printed text, but with wider variations
- intra-writer differences in productions of the same glyph, which are unique to handwriting and introduce additional difficulty in general-purpose recognition

Arabic script presents additional challenges for handwriting recognition systems due to its highly connected nature, numerous forms of each letter, presence of ligatures, and regional differences in writing styles and habits. Most cursive script recognition systems can be separated coarsely into three major categories:\textsuperscript{4,5}

- segmentation-based: take a sequence of primitive segments derived from a word image together with a group of possible words, and attempt to concatenate individual segments to best match candidate characters,
- segmentation-free: no attempt is made to separate the image into pieces that relate to characters, although splitting into smaller pieces is still possible, followed by generation of a sequence of observations,

Send correspondence to izavorin@caci.com
perception-oriented: perform a human-like reading by utilizing some stable features to reliably find characters anywhere in the image, and to use them to bootstrap a few word candidates for a final evaluation phase.

In both segmentation-based and segmentation-free cases, algorithms try to reproduce how a word was written by recognizing its components in the left-to-right (or right-to-left) fashion, thereby constituting a group of writing-oriented methods. Perception-oriented methods, however, try to emulate human-like reading techniques, but they have rarely been turned into complete recognition systems; instead, they are sometimes used to enhance more traditional writing-oriented algorithms.

1.2. Previous work

During the last several years, off-line Arabic handwriting recognition has been a popular topic in the OCR/ICR research community. According to the recent publications, the research and development efforts are growing leading to important advances in this area.

The ICDAR’05 held a competition on Arabic handwriting recognition, which provided an excellent snapshot of the state of the field. Six recognition systems were compared using the IFN/ENIT database of individual words that represent a small-size lexicon of village and town names in Tunisia. The results of this competition indicate that most existing methodologies are capable of recognizing individual words at below the 76% accuracy level.

In September 2006 the Summit on Arabic and Chinese Handwriting Recognition (SACH’06) took place at the University of Maryland. SACH’06 gave an overview of the state-of-the-art in handwriting recognition with emphasis on Chinese and Arabic. Several interesting ideas appeared in papers presented at the Summit. These include

- use of elaborate language models such as morphological rules for post-recognition uncertainty resolution,
- multi-tier approach to recognition, e.g. by recognizing Parts of Arabic Words (PAWs) first followed by recognition of complete words,
- use of features that are tied to the baseline in Arabic handwritten text.

Below we discuss some of the most interesting papers relevant to the off-line Arabic handwriting recognition.

Lorigo summarized the current state of the Arabic handwriting recognition field and the direction that research appears to be trending. He also outlines attempts at the University of Buffalo to analyze ancient Arabic documents using existing recognition tools. After reviewing some basic image processing techniques related to Arabic handwriting recognition, the author offers a broad look at the use of Neural Networks, Hidden Markov Models and combinations of the two in handwriting recognition. The author concludes by discussing a practical application of these tools in analyzing ancient documents.

In this paper, Govindaraju outlines the two currently accepted approaches to handwriting recognition: Holistic and Analytical. Holistic approaches attempt to identify whole words at once while analytical methods tries to build words from recognized characters. In order to unify the two, Govindaraju attempts to find a middle ground between the two for Arabic handwriting recognition by proposing a recognition tool that utilizes PAWs (pieces of Arabic words) lexicons.

AbdulKader describes the ICRA system for offline Arabic handwriting recognition (AHWR). The system is a two-tier recognizer. The methodology exploits the fact that, due to the fact that six Arabic letters do not connect to other letters from the left, all Arabic words consist of PAWs (Part of Arabic Word). The number of unique PAWs grows sub-linearly with the number of words. Thus the first tier of the system consists of a PAW recognizer that processes connected components of a given word image using a Neural Network and a PAW-to-letter lexicon. The second tier uses a word-to-PAW lexicon to produce the final result using a variation of the best-first search algorithm called Beam search. This system was a participant of the competition at ICDAR2005 and was reported to have one of the best performance levels when applied to the IFN/ENIT corpus.

Belaïd and Choisy summarize Arabic handwriting recognition methods inspired by human reading. These methods work inversely to most other recognition tools. First a Part of Arabic Word (PAW) is isolated and
potentially matching PAWs are selected from the lexicon. Each PAW is segmented into characters and in some
cases characters are segmented into their constituent features. Using the letters or features the target PAW
is matched to its most closely matching PAW in the lexicon. These methods utilize a range of probabilistic
tools such as HMMs and various types of Neural Networks. Per authors, some researchers have produced PAW
recognition rates as high as 97%.

Cheriet and Beldjehem\textsuperscript{12} describe problems encountered during Arabic handwriting recognition process and show
why, where and how \textit{morphological analysis} and \textit{natural language processing} NLP of Arabic could be used to
improve the accuracy of the Arabic handwriting recognition. The authors give a summary of techniques that
could be used to generate an Arabic lexicon. They acknowledge that manual lexicon creation is impractical, and
propose to generate the dictionary automatically from a rich lexical source such as Koran. A semi-automatic
creation is considered a feasible and promising option. The authors cite a survey of Arabic morphological analysis
techniques by Al-Sughaiyer et. al\textsuperscript{13} as a comprehensive source of Arabic morphology analysis.

The HMM-based Arabic handwriting recognition system we describe in this paper is largely based on the approach
described by Khorsheed.\textsuperscript{14} An image of a word is skeletonized, the knots (end- and cross- points) are computed
in each connected component of the resulting skeleton. The links between knots are approximated by piecewise-
linear curves and the linear segments that correspond to large-scale features such as loops, T- and X-cross points
and turning points are appropriately labeled. The tracer converts the resulting piecewise-linear approximation
into a sequence of integer observation symbols according to the labels of the individual linear segments. This
sequence of observation symbols is then used for training the word based HMMs that are used for recognition of
individual words or phrases. Section 2 presents a more detailed description of our system.

1.3. Paper organization

In this paper we address the problem of pre-segmented Arabic handwritten word recognition. Rather than
focusing on a single classification approach and trying to perfect it, we propose to combine heterogeneous
classification methodologies to obtain results that are better than those of the individual classifiers. This approach
was inspired in part by the results we obtained when developing a multi-engine multi-evidence OCR system\textsuperscript{15}
that combined multiple OCR engines followed by several filter-type text processors that yielded performance on
Arabic machine-printed text that was considerably higher than that of simple majority voting. We believe that
the framework that we have developed may be used to easily incorporate other classifiers.

The rest of the paper is organized as follows. In Section 2 we describe the individual components of our recognition
system and ways of combining them. Then, in Section 3 we describe experiments performed on the system. Our
current work is summarized and future work is proposed in Section 4.

2. SYSTEM OVERVIEW

In this section we first describe the individual components of our recognition system. Specifically, in Sections
2.1 and 2.4, we present two types of HMM-based word recognizers that differ in the codebooks that they use.
Section 2.2 is devoted to a PAW segmenter designed to recognize several classes of simple isolated Parts of Arabic
Words. In Section 2.3 we describe a word lexicon reducer that is built using the results of the PAW segmenter.
Then, in Section 2.5 we show how the individual components were connected together.

2.1. HMM recognizer based on large-scale features

This HMM-based recognizer is a modified version of the word recognizer developed earlier.\textsuperscript{16} It is designed
for pre-segmented word recognition. The recognizer consists of two major components: a \textit{(glyph) tracer} and a
classifier. A tracer performs feature extraction and creates from an image (of a word) a sequence of integer
observation symbols corresponding to various large-scale features found in the image. It was developed following
closely the methodology described by Khorsheed.\textsuperscript{14} Feature extraction is performed in several steps. The
image is first thinned. Then, knots (end and cross points) are computed in each connected component of the
resulting skeleton. The next step is to determine links that are sequences of skeleton points that connect pairs
of knots. The links are approximated by piecewise-linear curves and the linear segments that correspond to
large-scale structural features, such as loops, dots, T- and X-cross points and turning points, are appropriately
labeled. We use a codebook of size 14 shown in Table 1, which is larger than the original one and has shown better discrimination capabilities. The tracer then converts the resulting piecewise-linear approximation into a sequence of integer observation symbols according to the labels of the individual linear segments. This sequence is passed to the classifier for either training or recognition.

We currently make an assumption that test data consists only of words seen during training. Therefore, for the classifier, we build a word lexicon that consists of discrete Hidden Markov Models (DHMM) corresponding to unique words that appear in training data. For each lexicon entry, we train a unique model using data generated from all sample images corresponding to that word. A word HMM model has a left-to-right topology and is a connection of individual character models. Each character model also has a left-to-right topology and the number of its states depends on the complexity of the corresponding character. For instance, the model of the letter alif has fewer states than that of the letter sheen. We use standard Baum-Welch training to compute parameters of the model.

During recognition, the observation sequence produced by the tracer from a word image is evaluated against all word models. For a given sequence and a given model, the most likely path is found through the model and the corresponding Viterbi probability is computed. Viterbi probabilities collected from all lexicon models are first normalized and then are used to rank the lexicon entries and top \( N \geq 1 \) candidates are reported, where \( N \) is specified by the user. We denote this recognizer producing top-\( N \) candidates for each test word by \( DHMM_T(N) \).

### Table 1. Expanded codebook for the tracer

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOT_ABOVE</td>
<td>Dot above baseline</td>
<td>LP_CX2</td>
<td>2-Link Complex Loop</td>
</tr>
<tr>
<td>DOT_BELOW</td>
<td>Dot below baseline</td>
<td>LP_CX3</td>
<td>3-Link Complex Loop</td>
</tr>
<tr>
<td>END</td>
<td>End point</td>
<td>TP_Left_R</td>
<td>Regular left Turning Point</td>
</tr>
<tr>
<td>T</td>
<td>Branch point</td>
<td>TP_Left_S</td>
<td>Sharp left Turning Point</td>
</tr>
<tr>
<td>X</td>
<td>Cross point</td>
<td>TP_Right_R</td>
<td>Regular Right Turning Point</td>
</tr>
<tr>
<td>LP_SMPL</td>
<td>Simple Loop</td>
<td>TP_Right_S</td>
<td>Sharp Right Turning Point</td>
</tr>
<tr>
<td>LP_DBL</td>
<td>Double Loop</td>
<td>OTHER</td>
<td>None of the above</td>
</tr>
</tbody>
</table>

### 2.2. PAW segmenter

Both machine-printed and handwritten Arabic is written cursively with most letters connected on both sides with their immediate neighbors. Therefore an image of an Arabic word may constitute a single connected component. However, since six Arabic letters do not connect to other letters from the left, all Arabic words can be represented by one or more PAWs (Parts of Arabic Word). Each PAW is a set of consecutive connected letters in a word that ends with either the last letter of the word or a letter that does not connect on the left to the next letter of the word. Thus each PAW contains a single primary connected component corresponding to the connected letters. In addition, since many letters also contain dots and various diacritics, a PAW may also contain one or more secondary connected components corresponding to the dots and diacritics. For instance, the word on the left of Figure 1 consists of a single PAW that contains six secondary connected components (dots). The primary connected component represents 5 letters connected together. The word on the right of Figure 1 consists of three PAWs. The rightmost PAW consists of only the primary component that represents two connected letters. The middle PAW does not have any secondary components either and represents a single letter. This type of PAW is of particular interest to us. The leftmost PAW also consists of a single letter with one primary and three secondary components. We note that, rather than using a relatively small alphabet of letters to represent all words in the Arabic language, one may use a much larger, but more sparse, alphabet of unique PAWs, with its size growing sublinearly with the number of unique words.

The PAW Segmenter that we developed attempts to group individual connected components into PAWs. This is done using proximity of connected components to one another and partial recognition results from Isolated Character Detection.
Before any actual PAW segmentation is done, we first attempt to recognize isolated character PAWs. These are characters such as й that commonly occur in isolation and constitute their own free-standing PAW. Since we are using connected component analysis to detect these characters, it is necessary to group similar characters into character classes based on the primary character component. For example, ﬂ and ﬂ differ only by a single dot appearing above the main body of ﬂ. Though training for each character is done separately, there is no distinction between the two in the detection results. Further, to avoid complications created by local machine settings, a character detection is indicated in the output by the transliteration of its class representative rather than the actual representative. So in the case where ﬀ is detected, the output will show the transliteration of ﬀ, or "waw." See Table 2 for current detected characters and their corresponding classes.

<table>
<thead>
<tr>
<th>Characters</th>
<th>Class</th>
<th>Characters</th>
<th>Class</th>
<th>Characters</th>
<th>Class</th>
<th>Characters</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ﬀ</td>
<td>0</td>
<td>ﬀ</td>
<td>2</td>
<td>ﬀ</td>
<td>3</td>
<td>ﬀ</td>
<td>6</td>
</tr>
<tr>
<td>2 ﬀ</td>
<td>2</td>
<td>ﬀ</td>
<td>3</td>
<td>ﬀ</td>
<td>3</td>
<td>ﬀ</td>
<td>6</td>
</tr>
<tr>
<td>8 ﬀ</td>
<td>8</td>
<td>ﬀ</td>
<td>9</td>
<td>ﬀ</td>
<td>9</td>
<td>ﬀ</td>
<td>9</td>
</tr>
<tr>
<td>ﬀ</td>
<td>ﬀ</td>
<td>ﬀ</td>
<td>ﬀ</td>
<td>ﬀ</td>
<td>ﬀ</td>
<td>ﬀ</td>
<td>ﬀ</td>
</tr>
</tbody>
</table>

Isolated Character Detection uses a series of “sieves” through which each connected component candidate is passed. Each sieve either “catches” or “permits” a connected component based on one of the following features:

1. Aspect Ratio
2. Centroid Location Relative to Width
3. Centroid Location Relative to Height
4. Density
5. Edge to Mass Ratio
6. Height to Line Height Ratio
7. Width to Line Height Ratio
8. Mass to Stroke Width Ratio
9. Loop Count
10. Maximum Transitions in the Horizontal Direction
11. Maximum Transitions in the Vertical Direction
12. Top Edge to Bottom Edge Ratio
13. Right Edge to Left Edge Ratio
14. Template Match score

A sieve will catch a component candidate if the component statistic in question falls outside of a fixed number (as determined by training) of standard deviations from the training set mean. If a component candidate is caught by any sieve, then the component is not recorded as a match. As each sieve is trained using the entire match vs. non-match corpus, there is no correlation between sieve ordering and precision or recall.
Segmentation is performed by attaching dots to non-dot components and by grouping non-dot connected components pairwise according to their proximity from one another. Dots are connected components whose pixel mass falls below $b\rho^2$ where $\rho$ is the average stroke width of the word image and $b$ is a scalar determined during training. Dots are deterministically attached to the non-dot component with whom they overlap the most. In the case where the dot overlaps multiple characters (or no characters at all) the dot is attached to the component with the bounding box center located closest to the dot.

The probability for grouping two non-dot components is determined using a “Gaussian Ellipse” with parameters $\sigma$, $\psi$, and $\theta$. This Gaussian Ellipse is nothing more than a standard 2-D Gaussian of variance $\sigma^2$ stretched horizontally by a factor of $\psi^2$ and rotated counterclockwise by $\theta$ (in radians). For connected components $c_1$, $c_2$, the probability of grouping denoted $c_1 \diamond c_2$ is given by:

$$P(c_1 \diamond c_2) = \max_{p_1 \in c_1, p_2 \in c_2} (G(p_1, p_2))^*$$

where

$$G(p, q) = \frac{1}{\sigma \sqrt{2\pi}} e^{\left(\frac{D_x(p, q, \theta)}{\psi^2} + (\psi D_y(p, q, \theta))^2\right) / 2\sigma^2}$$

and

$$D_x(p, q, \theta) = (p_x - q_x) \cos \theta - (p_y - q_y) \sin \theta$$
$$D_y(p, q, \theta) = (p_x - q_x) \sin \theta + (p_y - q_y) \cos \theta$$

Grouping is transitive in that if $c_1 \diamond c_2$ and $c_2 \diamond c_3$ then $c_1 \diamond c_3$ regardless of $P(c_1 \diamond c_3)$. However, in that case, the $c_1 \diamond c_3$ probability will not be considered when ranking grouping schema. After the pairwise grouping is complete and all likely (above a fixed minimum) schema have been constructed, the schema are ranked according the product of the probabilities of all pairwise groupings ($p$) and nongroupings ($1 - p$). Once this is complete, PAW Segmenter converts the top schema to a string array representation of the grouping. Detected isolated characters are represented by their character class, dots are subdivided by their position relative to the baseline into “dot_A” or “dot_B”, and all remaining connected components remain as “unknown”.

### 2.3. Ranking lexicon reducer

The ranking lexicon reducer performs the two functions that its name implies: it reduces the full list to a set of candidates and then ranks these candidates with scores in the range $[0, 1]$. Thus, it can act as a full recognizer, but its primary purpose is to perform simple processing that supplements another recognizer, by reducing the number of candidate terms to consider and also providing initial scores that can act as “hints” to a fuller recognition filter.

This filter builds on the PAW Segmenter’s partial recognition of isolated Arabic characters. The resulting sequence of hypothesized PAWs expressed as a collection of component labels, e.g., “[unknown, dot_A, dot_A, X]” is called a shape chain. Note that without PAW grouping, each element of a shape chain has at most one symbol other than dots; with grouping, a shape chain element may contain many symbols, each corresponding to a connected component. During training, the Ranking Lexicon Reducer records, for each observed shape chain element, the hypothesized PAW positions in which it has occurred (first PAW, second PAW, etc.), and the words in which it has occurred in each position, with counts. During recognition, it uses the shape chain of the input word and this recorded training information to first reduce the lexicon and then rank the remaining candidates.

The filter maintains as candidates only those words for which all shape chain elements in the maximal prefix of the input shape chain have occurred in training at least once in the same position for that word. For example,

---

*Probabilities of grouping for components recognized as isolated characters are reduced by half.*
consider the shape chain above: “|unknown, dot_A, dot_A|.” If there are any words for which \( \downarrow \) occurred at least once as PAW 0, “unknown, dot_A, dot_A” occurred as PAW 1, and \( \uparrow \) occurred as PAW 2, then these words become the candidate set. If no such words exist in the lexicon, but there are words for which \( \downarrow \) occurred at least once as PAW 0 and “unknown, dot_A, dot_A” occurred at least once as PAW 1, then these words become the candidate set, and so on. In the filter’s strictest configuration, the shape chain elements must match precisely; in an alternate configuration that emphasizes maintaining candidates, only the elements consisting of single recognized characters are used for reduction.

The words are then ranked according to the frequencies collected during training. The filter combines the frequencies for the matched shape chain elements, and it normalizes them in one of three ways, according to its configuration: by candidate set, distributing a total score of 1 proportionally across the candidates; by “position bin”, normalizing the scores for all words with a given element in a given PAW position; or by term, normalizing the scores for a given lexicon term.

### Table 3. Sample training set with shape chains

<table>
<thead>
<tr>
<th>Sample ID</th>
<th>Term ID</th>
<th>Grouped Shape Chain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>( \downarrow \text{dot}_A \mid \text{unknown} \mid \uparrow \text{dot}_A )</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>( \downarrow \mid \uparrow \text{unknown} \mid \text{dot}_B )</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>( \downarrow \text{dot}_A \mid \text{unknown} \mid \text{dot}_B )</td>
</tr>
</tbody>
</table>

### Table 4. Training record for sample set in Table 3

<table>
<thead>
<tr>
<th>Shape Chain Element</th>
<th>Position</th>
<th>Term ID</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \downarrow \text{dot}_A )</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \downarrow \text{dot}_A )</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>( \downarrow \text{unknown} )</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \uparrow \text{unknown} )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \uparrow \text{unknown} )</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>unknown</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>unknown dot_B</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>unknown dot_B</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

As a small example, consider a training set consisting of three word samples, where the lexicon consists of two terms. Suppose that the training samples have the characteristics shown in Table 3. In this case, the training process will record the information shown in Table 4. Now suppose that, during testing, the system sees an image that it characterizes with the shape chain “\( \downarrow \text{dot}_A \mid \uparrow \text{unknown} \).” The maximal prefix with elements that it can match consists of \( \downarrow \text{dot}_A \mid \uparrow \text{unknown} \), so the candidate list consists of the lexicon entries for which \( \downarrow \text{dot}_A \) has occurred in position 0 and \( \uparrow \text{unknown} \) has occurred in position 1, i.e., lexicon entry 1.

We denote the configuration of the RLR filter as \( RLR_c(e, n) \), where \( c \) indicates grouping (“g”) or ungrouped components (“u”), \( e \) indicates the choice of significant elements, and \( n \) indicates the normalization choice. The \( e \) value may be “all” or “char,” indicating use of all element types or only single character element types. The
value may be “candidate,” “bin,” or “term,” indicating the three normalization options described above, respectively. For example, $RLR_g(char, bin)$ denotes the RLR filter running with grouping, reducing only by single recognized character elements, and normalizing by “position bin.”

2.4. PAW-segmenter-based HMM recognizer

This recognizer is similar to the one described in Section 2.1 in that it uses discrete Hidden Markov Models for classification. The main difference is the observation codebook it uses. Its feature extraction (tracing) is performed by the PAW segmenter described in detail Section 2.2. Since we use the segmentation-free approach during training, sequences of symbols generated by the segmenter for each PAW or connected component of a given word image are concatenated into a single observation sequence. The same concatenation is used during recognition. Character classes listed in Table 2 are converted into discrete observation symbols similarly to how structural feature labels generated listed in Table 1 are converted into symbols for $DHMM_T(N)$. We denote the HMM-based classifier combined with the PAW-segmenter-based tracer and producing top-$N$ candidates for each test word by $DHMM_T(N)$.

2.5. Combining classifiers

The system is implemented in a filter-based fashion and employs a transparent page document architecture that preserves all processing history and makes it available for any filter (processing component) in the system. It includes several trainable filters (e.g. PAW Segmenter, Ranking Lexicon Reducer, HMM Classifier) that represent and implement various stages of the handwriting recognition process. Refer to figure 2 for a typical system configuration. Note that we split the tracer and classifier components of the HMM recognizer into separate filters.

![Figure 2. A typical system configuration](image)

Such a configuration would input a word/phrase image, send it (in parallel) to the [PAW segmenter - Ranking Lexicon Reducer] and to the Tracer branches, and feed the observation sequence along with the reduced lexicon to the HMM based recognizer, producing the final text output candidates and optional confidence values.

Other configurations are possible as well, for instance, when $DHMM_T(N)$ is invoked first and used essentially as an alternative lexicon reducer follower by a $RLR_c(e, n)$ filter that yields the final list of candidates for a given test image.

The underlying framework is very flexible and can easily be extended by custom filters. It allows building and configuring stand-alone and distributed systems with filters running on different machines in a true multi-process and multi-threaded environment.
3. EXPERIMENTS

We performed numerous experiments using different filter configurations and different datasets selected from training and testing from the IFN/ENIT corpus, and here we present results of some of these experiments. For each filter configuration and each dataset, we report four values. As measures of performance, we compute the top-$N$ statistics, which is the fraction of test samples for which the correct word is found among the top $N$ candidates, for $N = 1, 5, 10$. In addition we compute the hit rate, which is the fraction of correct words among all candidates generated by a given filter configuration for a given input image. This implies that for filters such as $DHMM_T(N)$ that cannot produce more than $N$ candidates, the hit rate is the same as the top-$N$ value. For the filters that can produce any number of candidates between 0 and the size of the trained lexicon such as $RLR_e(e, n)$, these values are usually different. The main reason for selecting these performance metrics is that they are well-defined for the segmented word recognition problem that is being addressed when using the IFN/ENIT corpus.

We selected the following datasets for training and testing:

- **ABCD-E**, i.e. training on IFN subsets A through D and testing on E. We selected this data configuration for two reasons. First, it allowed us to test our system on unknown data. Second, it allowed us to emulate the ICDAR2005 competition thus giving us a performance baseline.
- **ABCE-D**, i.e. testing on D what was trained on the other four subsets. The main reason for choosing this data configuration was that it has been reported that subset E was a noticeable outlier relative to subsets A through D. This configuration therefore allowed us to test how our system behaved when additional ambiguity was introduced during training.
- **ABCDE-D**, i.e. training on the entire IFN corpus and testing on D. This data configuration allowed us to evaluate system performance on known data.

To save space, in this paper we only present test results for the **ABCE-D** configuration (see Table 5, where entries corresponding to the winning filter configuration(s) for each category are shown in bold). Tests on the other two configurations yielded comparable results. Due to a large number of filter combinations that can be created from the individual classifiers, we only show the two that produced the best results. These combined classifiers are

- $RLR_g(char, bin) + DHMM_T(10)$, which is the ranked lexicon reducer (RLR) with grouped components using single character element types with bin normalization (see Section 2.3 for details) followed by the discrete HMM recognizer based on large-scale features (see Section 2.1 for details). The RLR classifier produces a ranked subset of the lexicon for each word being recognized which is passed to the HMM classifier that narrows down this list of candidates to top 10.
- $DHMM_T(100) + RLR_u(all, term)$, which is the discrete HMM recognizer based on large-scale features (Section 2.1) producing top 100 candidates for every word being recognized, followed by the ranked lexicon reducer with ungrouped components using all element types with term normalization (Section 2.3) that ranks the 100 candidates, possibly eliminating some of them completely.

As Table 5 shows, the two outperformed all individual classifiers in terms of top-$N$ performance.

4. SUMMARY

In this paper we presented a highly configurable system for isolated Arabic handwritten word recognition. When tested on the IFN/ENIT corpus, it has achieved a 73% top-1 word recognition rate on seen test data and 52% on unseen data. While there is certainly room for improvement, what is encouraging is that combining different classification approaches produced better results than using the same approaches individually, which suggests that this methodology should be developed further. In addition we plan to improve performance of the individual

---

1. This clearly implies that for $N_1 > N_2$, $top-N_1 \geq top-N_2$, and that if $N_L$ is the size of the lexicon then $top-N_L \equiv 1$. 

components, test the system on a larger corpus of data (being currently developed) and investigate incorporation of other classification techniques into the framework.

Over the last several years, there has been a significant body of work generated in the maturing field of Arabic handwriting recognition. A lot of different approaches have been proposed each of which has its own strengths and weaknesses. Often these approaches were evaluated on test corpora that were small and/or highly specialized. Clearly, one way to improve accuracy is to fine-tune the individual classification methodologies. At the same time, based on our experience with recognition of Arabic machine-printed\(^{15}\) and handwritten text, we believe that another approach to developing an Arabic handwriting recognition system that is truly robust is to combine individual classifiers in an intelligent way. This paper demonstrates one approach to such a classifier combination. The software framework is flexible and can easily integrate other classifiers in the form of software libraries or stand-alone executables.

REFERENCES


