Arabic Font Recognition using Decision Trees Built from Common Words

Ibrahim S. I. Abuhaiba
Department of Electrical and Computer Engineering, Islamic University of Gaza, Gaza, Palestine

We present an algorithm for a priori Arabic optical Font Recognition (AFR). The basic idea is to recognize fonts of some common Arabic words. Once these fonts are known, they can be generalized to lines, paragraphs, or neighbor non-common words since these components of a textual material almost have the same font. A decision tree is our approach to recognize Arabic fonts. A set of 48 features is used to learn the tree. These features include horizontal projections, Walsh coefficients, invariant moments, and geometrical attributes. A set of 36 fonts is investigated. The overall success rate is 90.8%. Some fonts show 100% success rate. The average time required to recognize the word font is approximately 0.30 seconds.

Keywords: Arabic fonts, optical font recognition, vertical normalization, projections, Walsh coefficients, invariant moments, geometrical features, decision tree learning.

1. 1. Introduction

Optical font recognition can be addressed through two complementary approaches: the a priori approach, in which characters of the analyzed text are not yet known, and the a posteriori approach, where the content of the given text is used to recognize the font. To our knowledge, there has been nearly no studies of the Arabic Font Recognition (AFR) problem. Available studies deal with Latin fonts, which have different characteristics than Arabic fonts. Therefore, in this paper, we present a novel solution to the a priori AFR problem.

Often, the font style is not the same for a whole document; it is a word feature, rather than a document feature, and its detection can be used to discriminate between different regions of the document, such as title, figure caption, or normal text. The detection of the font style of a word can also be used to improve character recognition: we know that Mono-font OCR systems achieve better results than Multi-font ones, so the recognition of a document can be done using first an OFR, and then a Mono-font OCR.

To our knowledge, the only work that addresses the AFR problem is that of reference [1], where an algorithm for AFR is presented. First, words in the training set of documents for each font are segmented into symbols that are rescaled. Next, templates are constructed, where every new training symbol that is not similar to existing templates is considered as a new template. Templates are sharable between fonts. To classify the font of a word, its symbols are matched to the templates and the fonts of the best matching templates are retained. The most frequent font is the word font. The overall error, rejection and success rates were 15.0%, 7.6%, and 77.4%, respectively. The high error rate is mainly due to errors in size recognition.

In [2], font recognition is developed to enhance the recognition accuracy of a text recognition system. Font information is extracted from two sources: one is the global page properties and the other is the graph matching result of recognized short words such as a, it, and of.

In [3], a multi-font OCR system to be used for document processing is presented. The system performs, at the same time, both character recognition and font-style detection of the digits belonging to a subset of the existing fonts. The detection of the font-style of the document words can guide a rough automatic classification of documents, and can also be used to improve character recognition. The system uses the tangent distance as a classification function in a nearest neighbour approach. The nearest neighbour approach is always able to recognize the digit, but the performance in font detection is not optimal. To improve the performance, they used a discriminant model, the TD-Neuron, which is used to discriminate between two similar classes.

In [4], a texture analysis-based approach is used for font recognition. Existing methods are typically based on local features that often require connected components analysis. In this work, the document is taken as an image containing some special textures, and font recognition as texture identification. The method is content independent and involves no local feature analysis. The well-established 2-D Gabor filtering technique is applied to extract such features and a weighted Euclidean distance classifier is used in the recognition task. The reported average recognition accuracy of 24 fonts over 6,000 samples is 98.6%.

In [5], a statistical approach based on global typographical features is proposed for font recognition. It aims at the identification of the typeface, weight, slope, and size of the text from an image block without any knowledge of the content of that text. The recognition is based on a multivariate Bayesian classifier and operates on a given set of known fonts. The effectiveness of the adopted approach has been experimented on a set of 280 fonts. Font recognition accuracies of about 97% are reached on high-quality images. Rates higher than 99.9% were obtained for weight and slope detection. Experiments have also shown the system robustness to document language and text content and its sensitivity to text length.

In [6], a study of image degradations effects on the performance of a font recognition system is presented. The evaluation that has been carried out shows that the system is robust against natural degradations such as those introduced by scanning and photocopying, but its performance decreases with very degraded document images. In order to avoid this weakness, a degradation modeling strategy has been adapted, allowing an automatic adaptation of the system to these degradations. The adaptation is derived from statistical analysis of features behavior against degradations and is performed by specific transformations applied to the system knowledge base.

All previous OFR studies, except the first, deal with Latin fonts and there have been no similar studies on Arabic fonts, which have different characteristics than Latin fonts. The most impeding characteristic of Arabic OCR systems is the cursive nature of Arabic script, which makes characters not ready for direct OCR or OFR. In this paper, we present a novel contribution to the a priori AFR problem.

Due to difficulties in segmenting cursive Arabic text, the basic idea in our method is to recognize fonts of some common Arabic words. Once these fonts are known, they can be generalized to lines, paragraphs, or neighbor non-common words since these components of a textual material almost have a single font. Figure 1 shows one line of Arabic text written in three different fonts, from top to bottom: Simplified Arabic, Traditional Arabic, and Tahoma. This line contains the words "بعض", "على", and "و", which are all common Arabic words used in all contexts irrespective of the subject of material presented. If we could recognize the font of these

يطلق بعض المفكرين على عقد الخمسينيات بفترة ولادة الصحافة الأردنية الحديثة والتي تأثرت

a)

يطلق بعض المفكرين على عقد الخمسينيات بفترة ولادة الصحافة الأردنية الحديثة والتي تأثرت بالصحافة الفلسطينية المهاجرة

b)

يطلق بعض المفكرين على عقد الخمسينيات بفترة ولادة الصحافة الأردنية

c)

Fig. 1. One line of Arabic text written in three different fonts, all Roman and Regular, from top to bottom: (a) Simplified Arabic, (b) Traditional Arabic, and (c) Tahoma.

words only, it can be assigned to the rest of words in the line.

Decision tree learning is one of the most widely used and practical methods for inductive inference. It is a method for approximating functions that is robust to noisy data. So, a decision tree is our approach to recognize Arabic fonts. In our AFR system, a font is identified by four attributes: typeface (Simplified Arabic, Traditional Arabic, etc.), size expressed in typographic points, slant (Roman, Italic), and weight (Regular, Bold).

The rest of the paper is organized as follows. Common Arabic words are selected in Section 2. The feature set to be used in the font classification decision tree is presented in Section 3. The learning algorithm of the font decision tree is described in Section 4. The recognition process is explained in Section 5. In Section 6, experimental results are reported. Finally, the paper is concluded in Section 7.

2. Finding Common Arabic Words

Font recognition of every character is a very difficult and time-consuming problem, especially in cursive scripts such as Arabic. The reason of this difficulty is that words have to be segmented first into characters, which is also another difficult problem and no efficient solution to the segmentation problem is available. Actually, the details of character segmentation depend on the word font which, in current OCR systems, remains unknown until the characters are recognized. Therefore, we chose not to detect the font per character.

In document analysis systems, the stage of layout analysis results in segmenting the page into logical components. Examples of such components are non-textual components such as graphs, photos, line arts, etc., and textual components such as titles, section headings, and paragraphs. When we talk about font recognition, we are interested in textual components. Assuming that a textual component is written in almost one font, it is not necessary to detect the font of every word in a textual component. It is sufficient to detect the font of some representative words and generalize that font to the whole textual component, which can be faster.

The set of representative words are the most common words used in Arabic texts irrespective of the subject. For easier segmentation, we selected words that do not contain a separation character in the middle, see Figure 2 for a list of these characters. These are characters which,

ا أ إ آ ر ز د ذ و

Fig. 2. Arabic separation characters.

في من على عن لا ما عليه صلى له بن ثم كما هو لم به فيه صحيح

مع بين عند كل بعد بعض فلا حتى غير مسلم فقد قد عنه فيها بها و

عبد هي قبل منها عليها يا منه لها لما ليس حيث مثل عنها فما فهو

بل هل محمد بما يجب كثير سنة حق يعلم يصلي لهم تلك فقط لك

عنهما بد مما لو نفسه فيما فمن لمن فهل يمكن لله جميع يتم بسبب

كنت عليهم عمل منهم عنهم فلم منع معه فهي عشر شك شخص

ضعيف خير هم فلما كلمة لنا فعل تحت كيف عليك سبيل يعتبر

Fig. 3. The 100 most common Arabic words not having a separation character in the middle.

when exist in the middle of the word, separate it into more than one non-connected sub-words. Such discontinuity causes difficulties when segmenting lines into complete words. A word without separation characters is connected and it is almost very easy to segment it successfully. So, we compiled very long files of Arabic texts from many disciplines such as Islamic publications, arts, engineering, science, journalism, etc. The frequency of every such word is computed. The top 100 words having the highest frequencies were selected, see Figure 3. These words were printed in different fonts to constitute the learning dataset and one testing dataset. Details of the learning and testing datasets can be found Section 6.

3. Feature Set

Some features to be described in this section need words to be normalized first in order to get a fixed number of features, which facilitates comparison processes. In the context of document understanding, the normalization operation almost means to normalize in two directions: x and y. Actually, this can be problematic since information is lost due to this kind of two-dimensional normalization. In our system, we perform a one-dimensional normalization, in the vertical direction, such that the height/width ratio is preserved, which results in normalized words that retain the relative geometrical attributes of the original un-normalized words. Assume that the image of a word is enclosed by a minimum bounding rectangle and is defined by the function:

$$I(x,y)=\begin{cases}1 & \textit{for foreground pixels}\\ 0 & \textit{for background pixels}\end{cases} \quad (1)$$

In the following, there is a complete description of our set of features.

3.1. Projections

After vertically normalizing a word Image, $I(x, y)$, its horizontal projection, $h(y)$, is calculated using the equation:

$$h(y)=\sum_x I(x,y), \quad y=0,1,...,N-1 \quad (2)$$

where N is the word height after normalization. This yields N features, f_i, $i = 0, 1, \dots, N-1$. We don't use the vertical projection since the number of columns varies from word to word, which produces different number of features for different words.

3.2. Walsh Coefficients

Since Walsh transform proved to be powerful in shape discrimination, we used the 1-D Walsh discrete transform of the horizontal projections, $h(y)$, described above, to find N Walsh coefficients. This provides N other features, f_i, $i = N, N+1, \dots, 2N-1$. Walsh coefficients are found using the equation:

$$W(u)=\frac{1}{N}\sum_{y=0}^{N-1} h(y)\prod_{i=0}^{n-1}(-1)^{b_i(y)b_{n-1-i}(u)} \quad (3)$$

where $N = 2^n$, and $b_k(z)$ is the kth bit in the binary representation of z. We don't use Walsh coefficients for the vertical projections since we perform vertical normalization resulting in a varying number of columns from word to word. In practice, Equation (3) is not used directly. Instead, the fast Walsh transform is found using the successive doubling method in a way similar to the FFT [7]. Thus, Equation (3) becomes:

$$W(u)=\frac{1}{2}[W_{even}(u)+W_{odd}(u)] \quad (4)$$

and

$$W(u+M)=\frac{1}{2}[W_{even}(u)-W_{odd}(u)] \quad (5)$$

where $M = N/2$, $u = 0, 1, \dots, M-1$.

3.3. Invariant Moments

For a 2-D discrete word image, $I(x, y)$, the moment of order $(p+q)$ is defined as

$$m_{pq}=\sum_x\sum_y x^p y^q I(x,y) \quad (6)$$

The central moments can be expressed as

$$\mu_{pq}=\sum_x\sum_y (x-\bar{x})^p(y-\bar{y})^q I(x,y) \quad (7)$$

where

$$\bar{x} = \frac{m_{10}}{m_{00}} \tag{8}$$

and

$$\bar{y} = \frac{m_{01}}{m_{00}} \tag{9}$$

The central moments of order up to 3 are:

$$\begin{aligned}
&\mu_{00} = m_{00}, \quad \mu_{10} = 0, \quad \mu_{01} = 0,\\
&\mu_{20} = m_{20} - \bar{x}m_{10}, \quad \mu_{02} = m_{02} - \bar{y}m_{01},\\
&\mu_{11} = m_{11} - \bar{y}m_{10},\\
&\mu_{30} = m_{30} - 3\bar{x}m_{20} + 2\bar{x}^2 m_{10},\\
&\mu_{12} = m_{12} - 2\bar{y}m_{11} - \bar{x}m_{02} + 2\bar{y}^2 m_{10},\\
&\mu_{21} = m_{21} - 2\bar{x}m_{11} - \bar{y}m_{20} + 2\bar{x}^2 m_{01},\\
&\mu_{03} = m_{03} - 3\bar{y}m_{02} + 2\bar{y}^2 m_{01}
\end{aligned}$$

The normalized central moments, denoted, η_{pq}, are defined as

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{10}$$

where

$$\gamma = \frac{p+q}{2} + 1 \quad \text{for} \quad p+q = 2, 3, \ldots \tag{11}$$

In our font recognition system, we used the following seven invariant moments which are derived from the second and third moments:

$$\phi_1 = \eta_{20} + \eta_{02} \tag{12}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{13}$$

$$\phi_3 = (\eta_{30} - \eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{14}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{15}$$

$$\begin{aligned}
\phi_5 = {} & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\\
& [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2]\\
& + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\\
& [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned} \tag{16}$$

$$\begin{aligned}
\phi_6 = {} & (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]\\
& + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})
\end{aligned} \tag{17}$$

$$\begin{aligned}
\phi_7 = {} & (3\eta_{21} - \eta_{30})(\eta_{30} + \eta_{12})\\
& [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2]\\
& + (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03})\\
& [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned} \tag{18}$$

This set of moments, which will be denoted by f_i, $i = 2N, 2N+1, \ldots, 2N+6$, is invariant to translation, rotation, and scale change [7].

3.4. Geometrical Features

These include: width, height, aspect ratio, area, x-coordinate of center of area, y-coordinate of center of area, perimeter, thinness ratio, and direction of axis of least second moment. After word segmentation, the limits, $x_{\min}$, $x_{\max}$, $y_{\min}$, and $y_{\max}$, of the bounding rectangle become known. The width, D, and height, H, of a word are defined as:

$$D = x_{\max} - x_{\min} + 1 \tag{19}$$

and

$$H = y_{\max} - y_{\min} + 1 \tag{20}$$

Width and height features provide information about the size of the bounding rectangle of the word. The aspect ratio, α, is defined by

$$\alpha = \frac{D}{H} \tag{21}$$

The area, A, which equals the number of 1 pixels in the image, indicates the relative size of the word, and is defined by:

$$A = \sum_x \sum_y I(x, y) \tag{22}$$

Note that the area equals m_{00} defined earlier. The x and y coordinates of the center of area are the parameters $\bar{x}$ and $\bar{y}$ defined previously. These two features locate where in the bounding rectangle the center of area lies. To find the perimeter, P, for every 1 pixel we find the number of 0 pixels in its 4-neighbors. The sum of these numbers for all 1 pixels in the word yields the perimeter, P. The thinness ratio, T, is calculated by

$$T = 4\pi\left(\frac{A}{P^2}\right) \tag{23}$$

As the perimeter becomes larger relative to the area, this ratio decreases and the word gets thinner. This can be useful in discriminating between Regular and Bold fonts and between thin and thick fonts. The axis of least second moment feature provides information about the word's orientation, which can be helpful in discriminating between Roman and Italic fonts. This axis corresponds to the line about which it takes the least amount of energy to spin the word or the axis of least inertia. If we move our origin to the center of area $(\bar{x}, \bar{y})$, the angle, θ, of axis of least second moment is defined as follows:

$$\theta = \frac{1}{2} \tan^{-1} \frac{2 \times \sum_x \sum_y xyI(x, y)}{\sum_x \sum_y y^2 I(x, y) - \sum_x \sum_y x^2 I(x, y)} \quad (24)$$

where θ is measured in counterclockwise direction from the vertical axis.

Geometrical features provide 9 more features and will be denoted by f_i, $i = 2N + 7, 2N + 8, \ldots, 2N + 15$. Therefore, the total number of features becomes N horizontal projections + N Walsh coefficients + 7 invariant moments + 9 geometrical features = $2N + 16$ features. In our system, N was set to 16 yielding 48 total feature $f_i = 0, 1, \ldots, 47$.

4. Decision Tree Learning

Decision tree learning is one of the most widely used and practical methods for inductive inference. It is a method for approximating functions that is robust to noisy data and capable of learning disjunctive expressions [8].

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some feature of the instance, and each branch descending from that node corresponds to one of the possible values for this feature.

Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees [8-10]. We will use an algorithm that learns the decision tree by constructing it top-down beginning with the question "which feature should be tested at the root of the tree?" To answer this question, each instance feature is evaluated using a statistical test to determine how well it alone classifies the training examples. The best feature is selected and used as the test at the root node of the tree. A descendent of the root node is then created for each possible suitable range of this feature, and the training examples are sorted to the appropriate descendent node (i.e., down the branch corresponding to the example's range for this feature). The entire process is then repeated using the training examples associated with each descendent node to select the best feature to test at that point in the tree. This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices.

The central choice in the algorithm is selecting which feature to test at each node in the tree. We would like to select the feature that is most useful for classifying examples. There are many several quantitative measures of the worth of a feature [8]. We will use a statistical property, called information gain that measures how well a given feature separates the learning examples according to their target font classification. The algorithm uses this measure to select among the candidate features at each step while growing the tree.

In order to define information gain precisely, we begin by defining a measure commonly used in information theory, called entropy, which characterizes the impurity of an arbitrary collection of examples. Given a collection S of font examples where the target font classification can take on c different values, the entropy of S relative to this c-wise classification is defined as:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \quad (25)$$

where p_i is the proportion of S belonging to font class i.

Given the entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of a feature in classifying the training examples. The measure we will use, called information gain, is simply the expected reduction in entropy caused by partitioning the examples according to this feature. More precisely, the information gain, $Gain(S, f)$ of a feature f relative to a collection of examples S, is defined as

$$Gain(S, f) \equiv Entropy(S) - \sum_{v \in Values(f)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{26}$$

where $values(f)$ is the set of all possible values for feature f, and S_v is the subset of S for which feature f has value v (i.e., $S_v = \{s \in S \,|\, f(s) = v\}$). Note that the first term in Equation (26) is just the entropy of the original collection, S, and the second term is the expected value of the entropy after S is partitioned using feature f. The expected entropy described by this second term is simply the sum of the entropies of each subset, S_v, weighted by the fraction of examples $\frac{|S_v|}{|S|}$ that belong to S_v. $Gain(S, f)$ is therefore the expected reduction in entropy caused by knowing the value of feature f.

To give a formal description of our font decision tree learning algorithm, let $S = \{s_i,\ i = 0, 1, \ldots, n-1\}$ be a set of training examples, i.e., instance word images, where n is the total number of examples and each instance, s_i, is a 2-tuple (V, τ), $V = (v_0, v_1, \ldots, v_{47})$, where v_j is the value of feature $f_j, j = 0, 1, \ldots, 47$, and τ is the value of an additional Font feature, i.e., the font classification of this example. Notice that the Font feature is the target function value to be learned and then predicted by the tree. Initially, the set S contains the whole training examples. A node in the tree to be created contains the following fields:

1. *feature*: which is used by interior nodes to store the number of the feature to be tested at that node,
2. *threshold*: which is used by interior nodes to decide which branch to follow. If the value of the feature to be tested at that node is less than threshold, then the left branch is followed, otherwise, the right branch is followed.
3. *font*: which is used by leaf nodes to store the target font classification.
4. *Examples*: which is used by a leaf node to store the examples sorted to that node. Here, we store only the V vector, i.e., the 48 feature vector values are stored. The font classification per example is not stored since it is common to all examples sorted to this node and it is retained in the *font* field.
5. *left* and *right*: which are pointers used by an interior node to point to left and right child nodes of that node, respectively.

A formal description of our font decision tree learning algorithm follows.

```
Algorithm I.  Font Decision Tree Learning (FDTL)

DecisionTreeNode FDTL(S)

Step 1.  New Node Creation
Create a Root node for the tree.

Step 2.  Leaf Node Check
If all Examples have the same font classification, $\tau$, then
{
    Mark Root as a leaf node
    Let $Root \to font = \tau$
    Store the V vector, i.e., the 48 feature values of S in $Root \to S$
    Return Root
}

Step 3.  Best Classifying Feature Selection
Find the feature, $f_{best}$, of the features $f_0$ to $f_{47}$ which best classifies the set S and the corresponding dividing threshold, $t_{best}$.

Step 4. Interior Node and Descendent Nodes Creation
Mark Root as an interior node
Set $Root \to feature = f_{best}$ and
$Root \to threshold = t_{best}$
Let $S_{<t_{best}} \subset S$ be the set of examples for which $v_{f_{best}} < t_{best}$
Let $Root \to left = \mathbf{FDTL}(S_{<t_{best}})$
Let $S_{\geq t_{best}} \subset S$ be the set of examples for which $v_{f_{best}} \geq t_{best}$
Let $Root \to right = \mathbf{FDTL}(S_{\geq t_{best}})$
Return Root

End of Algorithm I.
```

Steps 3 of Algorithm I needs explanation. Notice that the feature set, f_0 to f_{47}, in our problem, is continuous-valued. The decision tree creation can be made easier if these features are converted into discrete-valued features. Actually, this is accomplished by dynamically defining new discrete-valued features that partition the continuous-valued features into a discrete set of intervals. In particular, for a feature f_j that is continuous-valued, the algorithm creates a new Boolean feature b that is true if the value of f_j is less than a certain threshold t and false otherwise. We pick a threshold, t, which produces the greatest information gain of feature b. This can be achieved by sorting the examples according to the continuous feature, f_j, then identifying adjacent examples that differ in their target font classification, we can generate a set of candidate thresholds midway between the corresponding values of f_j. The value of t that maximizes information gain of feature b must lie at such boundary [11]. These candidate thresholds are evaluated by computing the information gain associated with each and the best feature can be selected. This is repeated for every feature, f_j, and the feature which yields the maximum information gain is selected.

Therefore, in Step 3 of Algorithm I, the best feature, f_{best}, which best classifies the set S and the corresponding dividing threshold, t_{best}, are found as follows. For every feature, f_j, in the 48 features:

1. Let $Y = \{y_i,\ i = 0, 1, \ldots, n-1\}$, where $y_i = (v_{i,j},\ \tau_i)$ and $v_{i,j}$ is the jth feature value in s_i.
2. Sort Y in ascending order according to feature f_j
3. Let $gainMax = 0$
4. In the following For loop, instead of comparing the font classifications of adjacent examples, we compare font classifications of y_i and $y_{i+\delta}$ to speed up the learning phase, since otherwise a great number of candidate thresholds arises, which highly increases the learning time. This is achieved by defining $\delta = \lfloor \log_2 n \rfloor$ instead of $\delta = 1$, where n is the number of examples in S:

```
For ($i = 0;\ i < n - \delta;\ i = i + \delta$)
{
    Let $\tau_i$ and $\tau_{i+\delta}$ be the font classes of $y_i$ and $y_{i+\delta}$, respectively.
    If $\tau_i \neq \tau_{i+\delta}$ then
    {
        $t = (v_{i,j} + v_{i+\delta,j})/2$
        Define the set of examples $Z = \{z_i = (b_i,\ \tau_i)\}$, where $\tau_i$ is the font class in $y_i = (v_{i,j},\ \tau_i)$ and $b_i$ equals 1 if $v_{i,j} < t$ and 0 otherwise.
        Let $g$ be equal to the information gain of feature $b$ relative to the set $Z$
        If $g > gainMax$ then set $gainMax = g$ and $t_{best} = t$
    }
}
```

The feature, f_{best}, which achieves the maximum information gain and the corresponding threshold, t_{best} are retained.

Pages of common Arabic words, defined in Section 2, are printed and scanned to constitute the learning dataset. Some image preprocessing is required in both the learning and testing stages. First, an image is skew-corrected using the algorithm of [12]. Next, horizontal and vertical solid lines are removed. Third, pepper noise is removed. The image is segmented into lines using horizontal white cuts, where each line consists of a sequence of words. Then, every line is segmented into words using vertical white cuts. Words which don't satisfy certain size constraints are filtered out. For example, if the height or width of a word is less than some specified thresholds, or is greater than some other thresholds, then the word is discarded, see Section 6 for values of these thresholds. This process is repeated for every image of the training dataset. Features of extracted words are calculated to obtain the complete set of training examples. The range of each feature, to be used in the recognition stage, is found. A formal description of the complete learning algorithm follows.

Algorithm II. Arabic Font Learning (AFL)

DecisionTreeNode AFL()

Step 1. Preprocessing

a. Correct the skew of the input image
b. Remove horizontal and vertical solid lines
c. Remove pepper noise
d. Segment the image into lines using horizontal white cuts

e. Every line is segmented into words using vertical white cuts

f. Repeat steps (a-e) for all images of the training dataset

Step 2. Feature Calculation

a. Find the 48 feature values of every extracted word passing some tests to be mentioned in Section 6. At the end of this step, a set of training example words, $S = \{s_i,\ i = 0, 1, \ldots, n-1\}$ is obtained, where n is the total number of examples and each instance, s_i, is a 2-tuple (V, τ), $V = (v_0, v_2, \ldots, v_{47})$, where v_j is the value of feature f_j, $j = 0, 1, \ldots, 47$, and τ is the value of the additional Font feature or the font classification of this example.

b. Find the full range, r_j, of every feature, f_j, $j = 0, 1, \ldots, 47$, where $r_j = \max_{i=0}^{n-1}(v_{i,j}) - \min_{i=0}^{n-1}(v_{i,j})$, where $v_{i,j}$ is the value of feature f_j in training example s_i.

Step 3. Decision Tree Learning

Apply the decision tree learning algorithm, Algorithm I, to the set of examples S to obtain the target font recognition decision tree. Let *Root* point to the root of this tree.

Return *Root*

End of Algorithm II.

5. Recognition

The same preprocessing operations used in the learning stage are also used in the recognition stage. However, it is not necessary that all examples extracted in the recognition stage belong to the set of common Arabic words defined in Section 2. To recognize the font of a word, its features are calculated. Feature testing starts at the root of the tree. Let the feature which must be tested at the root be $feature_j$ and the corresponding threshold be $threshold_j$. The value v_j of feature f_j of the word is compared against $threshold_j$. If $v_j < threshold_j$ then the left branch is followed, otherwise, the right branch is followed and the testing process is repeated at new interior nodes we arrive at. If a leaf node is reached, a check is made to make sure that the word being tested belongs to the set of common Arabic words defined in Section 2. This is achieved by finding whether the distance between the word being tested and any training example, in the set of training examples found at that leaf node, is not greater than a previously defined distance threshold. If so, the font saved at the leaf node is returned as the recognized font. Otherwise, the word is rejected. We define our distance threshold as:

$$DistThreshold = \alpha \times N_f \tag{27}$$

where α is some constant that is empirically determined and N_f is the number of features, which is 48 in our case. The distance measure we incorporated in our test is the range-normalized city block distance metric defined as

$$D(V, U) \equiv \sum_{j=0}^{47} \left| \frac{v_j - u_j}{r_j} \right| \tag{28}$$

where v_j is the value of feature f_j in a training example found in the leaf node and u_j is the corresponding feature value of the word being tested. This metric is computationally faster than the Euclidean distance, but gives similar results. The following is a formal description of the recognition algorithm per word. Before we call the algorithm, the set of feature values, $U = (u_0, u_1, \ldots, u_{47})$, of the word to be tested is calculated.

Algorithm III. Arabic Font Recognition (AFR)

```
font AFR(DecisionTreeNode, U)
{
    If DecisionTreeNode is an interior node, then
    {
        Let $f = DecisionTreeNode \rightarrow feature$
        If $(u_f < DecisionTreeNode \rightarrow threshold)$, then
            Return AFR($DecisionTreeNode \rightarrow left$)
        Else
            Return AFR($DecisionTreeNode \rightarrow right$)
    }
    Else
    {
        If the range-normalized city block distance between $U$ and any of the examples in the set $DecisionTreeNode \rightarrow Examples$ is not greater than $DistThreshold$, then
            Return $DecisionTreeNode \rightarrow font$
        Else
            Return $rejected$
            Here $rejected$ is a dedicated font value to indicate an unknown font value.
    }
}
```

End of Algorithm III.

Words whose fonts are recognized using the above algorithm can be used to predict the fonts of constituent headlines, text lines, and paragraphs of a page in many ways. For example, following a maximum voting strategy, a line/paragraph can be assigned the most frequent font in the set of recognized words in that line/paragraph. Or, a rejected word can be assigned the font of the nearest recognized word.

6. Results

In our system, a font, τ, is characterized by the 4-tuple (p, z, s, w), where p, z, s, w are the typeface, size, slant, and weight, respectively. In the fonts used in experimentation, we have three typefaces: Simplified Arabic, Traditional Arabic, and Tahoma. The slant is either Roman or Italic. The weight is either Regular or Bold. The size is 12, 13, or 14 points. Thus, a total of 36 fonts were investigated. Table I summarizes the fonts used in our AFR system.

Learning Dataset

The most frequent 100 Arabic words found in Section 2, see Figure 3, were printed once for every font in Table I, i.e., they were printed 3 typefaces $\times$ 3 point sizes $\times$ 2 slants $\times$ 2 weights $=$ 36 times. In every font, 20 samples were printed for each word. Thus, the total number of learning examples is 36 fonts $\times$ 20 examples per word $\times$ 100 words per font $=$ 72,000 examples.

Testing Datasets

A dataset, dataset no. 1, similar to that of the learning dataset was created, but this time 10 samples per word were created. This provided 36,000 samples for testing purposes. Another dataset, dataset no. 2, was created by compiling a file of normal Arabic text, i.e., it is not constrained to only common Arabic words of Figure 3. This file contained 15 paragraphs and, when printed in the fonts of Table I, yields from two to three A4 pages per font.

The learning and testing datasets were printed using a laser jet printer. Then, they were scanned as binary images with resolution set to 300 dpi in both directions, using an HP ScanJet 3400C scanner. Figure 4 shows one sample of a scanned paragraph from testing dataset no. 2.

In learning and testing stages, a word that does not satisfy any of the following constraints is filtered out: the word width and height are at least 12 pixels each, the maximum width and maximum height are 600 and 200 pixels, respectively. These values were empirically determined and proved adequate to eliminate flecks and some non-textual content.

It is worth mentioning that our font recognition decision tree is not designed to recognize every Arabic word. It only recognizes instances of words which belong to the set of common Arabic words defined in Section 2. If a word is

Typeface	Size	Slant	Weight
Simplified Arabic	12, 13, 14	Roman, Italic	Regular, Bold
Traditional Arabic	12, 13, 14	Roman, Italic	Regular, Bold
Tahoma	12, 13, 14	Roman, Italic	Regular, Bold

Table I. Arabic fonts used in our AFR system.

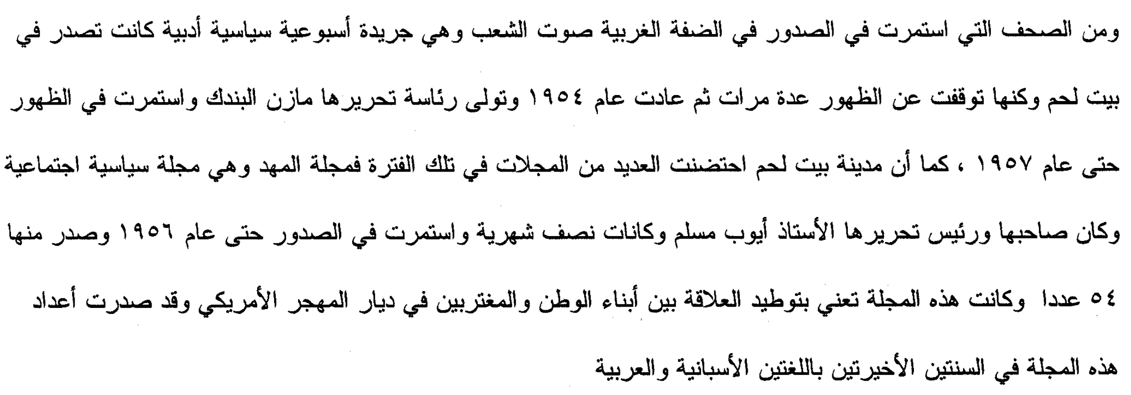
ومن الصحف التي استمرت في الصدور في الضفة الغربية صوت الشعب وهي جريدة أسبوعية سياسية أدبية كانت تصدر في
بيت لحم وكنها توقفت عن الظهور عدة مرات ثم عادت عام ١٩٥٤ وتولى رئاسة تحريرها مازن البندك واستمرت في الظهور
حتى عام ١٩٥٧ ، كما أن مدينة بيت لحم احتضنت العديد من المجلات في تلك الفترة فمجلة المهد وهي مجلة سياسية اجتماعية
وكان صاحبها ورئيس تحريرها الأستاذ أيوب مسلم وكانات نصف شهرية واستمرت في الصدور حتى عام ١٩٥٦ وصدر منها
٥٤ عددا وكانت هذه المجلة تعني بتوطيد العلاقة بين أبناء الوطن والمغتربين في ديار المهجر الأمريكي وقد صدرت أعداد
هذه المجلة في السنتين الأخيرتين باللغتين الأسبانية والعربية

Fig. 4. Sample scanned paragraph of the testing dataset no. 2, printed in (Simplified Arabic, Roman, Regular).

outside that set, then it is rejected by the tree. Therefore, in defining the recognition rate, R, we don't depend on the usual definition of this rate which is:

$$R = \frac{No.\ of\ words\ correctly\ recognized}{Total\ no.\ of\ words} \times 100 \quad (29)$$

This is due to the fact that in usual Arabic text most words don't belong to our set of common words, hence, they are rejected by the tree which gives the incorrect illusion that our algorithm is not practical. Instead, we use the following definition:

$$R' = \frac{No.\ of\ words\ correctly\ recognized}{Total\ no.\ of\ words - No.\ of\ rejected\ words} \times 100 \quad (30)$$

The recognition algorithm was run with the threshold *DistThreshold* set to $0.01 \times N_f = 0.01 \times 48 = 0.48$, where N_f is the number of features. For this threshold value, 46.6% of the total words in dataset no.1 were rejected by the tree although the training and testing datasets look similar. However, in font recognition, very small degradations in intensity values of image pixels produce new variants of word images that no longer pass the threshold test of the decision tree. The rejection rate can be decreased by increasing *DistThreshold*. However, this increases, as empirically verified, the error rate. Actually, there is very little harm of the rejection rate being high as long there is an adequate number of common words successfully recognized. Also, it is not necessary to recognize every common Arabic word in a paragraph. For the set of words accepted by the decision tree, the overall success rate of testing dataset no. 1 is 93.9%. Error is mainly due to errors in size and slant recognition. Errors in typeface are almost absent, while very few errors in weight are noticed. Simplified Arabic and Traditional Arabic fonts show high size, slant, and overall error rates, high rejection rate, and low recognition success rate compared to Tahoma fonts.

For testing dataset no. 2, 89.5% of input words were rejected by the decision tree, which is higher than the rejection rate of dataset no. 1. The reason is that dataset no. 2 contains too many other classes of words other than the common words on which the decision tree is trained. Of accepted words, the overall success rate of testing dataset no. 2 is 90.8%. Some fonts show 100% success rate. Error is mainly due to errors in size in contrast to results of dataset no. 1, in which error was mainly due to errors in size and slant recognition. Also, errors in typeface and weight are almost absent. Some Traditional Arabic fonts show high size and overall error rates and low recognition success rate compared to Simplified Arabic and Tahoma fonts. The error rate can be decreased by increasing the size of the training dataset.

The algorithm was implemented and run on a Pentium III 866MHz PC with 128 MB RAM. The average time required to recognize the word font is approximately 0.31 and 0.29 seconds for testing datasets nos. 1 and 2, respectively. This time doesn't include disk overhead and preprocessing times as these times are counted towards the overall document understanding software and not towards every constituent task in that software. The recognition time can be reduced by using some programming optimization techniques and more powerful computers.

The size of the learned decision tree is 15633 nodes, 7816 of which are interior nodes and the rest are leaves. In each interior node, one feature is tested and the same feature may be used more than once in a path from the root to a leaf node. Here, we have the following important observations about the usage of features in interior nodes of the decision tree:

1. All defined features were used in the tree. The mostly used feature is the horizontal projection of row no. 0, $h(0)$, of the vertically-normalized word image, which was used in 657 nodes out of 7816 interior nodes. The least used feature is Walsh coefficient no. 9, $W(9)$, which was used in 30 interior nodes.
2. Most of the heavily used features are very simple features which are not computationally expensive such as horizontal projections, width, area, and height.
3. Also, most of the heavily used features are the invariant moments. However, these are computationally expensive.

Table II summarizes feature usage which shows that horizontal projections occupy 47.1% of the interior nodes. This is a useful result since it indicates that we can depend on simple features to classify fonts. Next come invariant moments and Walsh coefficients which constitute 19.3% and 12.7%, respectively, of the features tested at interior nodes.

7. Conclusion

In this paper, we presented a novel contribution to the a priori AFR problem. A decision tree was our approach to recognize Arabic fonts. A set of 48 features was used to learn the tree. These features include horizontal projections, Walsh coefficients, invariant moments, and geometrical attributes.

A set of 36 fonts was investigated. The total number of learning examples was 72,000. Two datasets were used to test the method. The overall success rate is 90.8%. Some fonts show 100% success rate. The error rate can be decreased by increasing the size of the training dataset. Recognized fonts of common words can be generalized to their containing paragraphs, assuming that each paragraph contains only a single font, which can be practical in printed materials such as newspapers and old books that were not previously stored in electronic format.

Feature Name	**Symbol**	**Nodes**	**Percent**
Horizontal projections	$h(0)$ to $h(15)$	3685	47.1
Invariant moments	ϕ_1 to ϕ_7	1509	19.3
Walsh coefficients	$W(0)$ to $W(15)$	996	12.7
Width	D	354	4.5
Area	A	265	3.4
Height	H	233	3.0
Center of area x coordinate	$\bar{x}$	220	2.8
Center of area y coordinate	$\bar{y}$	129	1.7
Perimeter	P	117	1.5
Angle of axis of least second moment	θ	109	1.4
Aspect ratio	α	103	1.3
Thinness ratio	T	96	1.2
All features		**7816**	**100**

Table II. Summary of feature usage in interior nodes of learned font decision tree. Nodes: no. of interior nodes using the corresponding feature, Percent: percentage of interior nodes using the feature.

The algorithm was implemented and run on a Pentium III 866MHz PC with 128 MB RAM. The average time required to recognize the word font is approximately 0.30 seconds. The time can be reduced by using some programming optimization techniques and more powerful computers.

Regarding feature usage in the decision tree, horizontal projections occupy 47.1% of the interior nodes. This is a useful result since it indicates that we can depend on simple features to classify fonts. Next come invariant moments and Walsh coefficients which constitute 19.3% and 12.7%, respectively, of the features tested at interior nodes.

Acknowledgement

The author thanks the referees for their valuable comments which helped to improve this paper.

References

[1] I.S.I. Abuhaiba, Arabic Font Recognition Based on Templates, *The International Arab Journal of Information Technology*, 1 (2003), pp. 33–39.

[2] H. Shi and T. Pavlidis, Font Recognition and Contextual Processing for More Accurate Text Recognition, *Forth International Conference on Document Analysis and Recognition (ICDAR'97)*, (1997) Ulm, Germany.

[3] S.L. Manna, A.M. Colla, and A. Sperduti, Optical Font Recognition for Multi-Font OCR and Document Processing, *10th International Workshop on Database & Expert Systems Applications*, (1999) Florence, Italy.

[4] Y. Zhu, T. Tan, and Y. Wang, Font Recognition Based on Global Texture Analysis, *Fifth International Conference on Document Analysis and Recognition (ICDAR'99)*, (1999) Bangalore, India.

[5] A. Zramdini and R. Ingold, Optical Font Recognition Using Typographical Features, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20 (1998), pp. 877–882.

[6] A. Zramdini and R. Ingold, A Study of Document Image Degradation Effects on Font Recognition, *Third International Conference on Document Analysis and Recognition (ICDAR'95)*, (1995) Montreal, Canada.

[7] R.C. Gonzalez and R.E. Woods, *Digital Image Processing*, Prentice Hall, 2002.

[8] T.M. Mitchell, *Machine learning*, The McGraw-Hill Companies, Inc., Singapore, 1997.

[9] J.R. Quinlan, Induction of Decision Trees, *Machine Learning*, 1 (1986), pp. 81–106.

[10] J.R. Quinlan,*C4.5: Programs for machine learning*, San Mateo, CA: Morgan Kaufmann, 1993.

[11] U.M. Fayyad and K.B. Irani, On the handling of continuous-valued attributes in decision tree generation, *Machine Learning*, 8 (1992), pp. 87–102.

[12] I.S.I. Abuhaiba, Skew Correction of Textual Documents, *King Saud University Journal: Computer and Information Sciences Division*, 15 (2003), pp. 67–86.

Received: September, 2004
Revised: December, 2004
Accepted: February, 2005

Contact address:
Ibrahim S. I. Abuhaiba
P. O. Box 1276
Department of Electrical and Computer Engineering
Islamic University of Gaza
Gaza
Palestine
isabuhaiba@yahoo.com

Ibrahim S. I. Abuhaiba is an associate professor at the Islamic University of Gaza, Department of Electrical and Computer Engineering, Gaza, Palestine. He obtained his Master of Philosophy and Doctorate of Philosophy from Britain, in the field of document understanding and pattern recognition. His research interests include computer vision, image processing, document analysis and understanding, pattern recognition, artificial intelligence, and other fields. Dr. Abuhaiba presented important theorems and more than 30 algorithms in document understanding. He published many original contributions in the field of document understanding in well-reputed international journals and conferences.