Ancient Document Recognition Using Fuzzy Methods

Cláudia S. Ribeiro, João M. Gil, João R. Caldas Pinto, João M. Sousa

Technical University of Lisbon, Instituto Superior Técnico, GCAR
Av. Rovisco Pais, 1049-001 Lisboa Portugal

Abstract. This paper proposes an optical character recognition system based on fuzzy logic for 17th century printed documents. The process consists of two stages: training with collected character image examples and new character image classification. Training builds fuzzy membership functions from aligned oriented features extracted using Gabor filters. These are used in classification to select a most likely character group for new data. A post-processing stage with a proposed modified Levenshtein word distance metric further improves results. A success rate of 88% is achieved on a significant test set.

1 Introduction

Optical character recognition (OCR) is a practical application of state-of-the-art image processing and pattern recognition developments. Current communication facilities could allow broad distribution of vast libraries of books, newspapers, magazines and all kinds of printed media, if quality, cost-effective OCR procedures are available for mass digitizing. While modern printed text can be recognized very accurately with commercially available software, performing OCR on material such as gothic fonts, ancient typesets and handwriting is noticeably less successful.

This paper proposes a character recognition system specifically tailored to ancient documents (17th century) and corresponding typesets based on a handwriting OCR system using fuzzy logic [1]. The use of fuzzy classification [2] improves results by providing larger tolerance for unstable typesetting and printing technologies. The original modifications introduced to [1] are reported and explained along the article. A modified probabilistic Levenshtein word distance metric is also proposed, coupled with a post-processing system that is able to correct both character recognition and word segmentation.

Unlike the original holistic system, recognition is performed from an analytic perspective, i.e., by taking each character separately. The recognition procedure works in two steps. The first step, training, considers sets of character images, known as character groups, and combines their dominant graphical features. These are then used to build fuzzy membership functions that, in a sense, describe the visual attributes of every character group. For the second step, classification, a new character image is compared to the training results. The closest match, dictated by a fuzzy decision maker, is returned as the most likely classification. The final post-processing

S. Ribeiro C., M. Gil J., R. Caldas Pinto J. and M. Sousa J. (2004).
Ancient Document Recognition Using Fuzzy Methods.
In *Proceedings of the 4th International Workshop on Pattern Recognition in Information Systems*, pages 98-107
DOI: 10.5220/0002685600980107
Copyright © SciTePress

stage, based on spell checking, word distance metrics and probabilistic error information, is able to increase recognition success further, by selecting character corrections and improving word segmentation.

2 System Overview

This section intends to give an introduction to the general functioning of the developed application. Fig. 1 displays a diagram representing schematically the organization of the developed system and the streamlined design connecting its components. The main application relies on a commercial OCR package, ABBYY FineReader Engine [12], in order to obtain individual character images automatically. This package also provides methods to segment entire words, as well as its own OCR output. This information is used to build a manually classified character database, which is applied in the training stage to build models for every known character. The main module is then able to assign a most likely classification to other character images, thus performing the intended OCR on characters with high error probability. Word segmentation information, on the other hand, is considered and improved upon in the final post-processing step, which also takes advantage of spell checking and probabilistic error information.

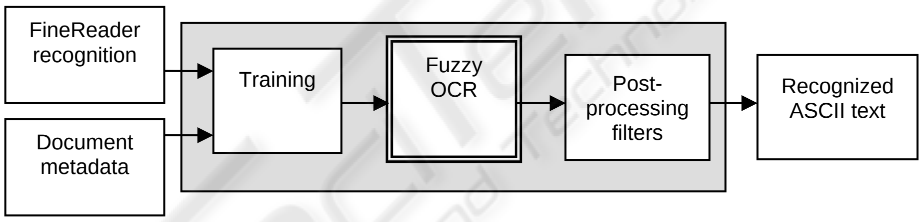


Fig. 1. System overview diagram

The underlying system for the recognition engine is based on a handwriting OCR system using fuzzy logic [1]. The following sections describe the original changes applied to this system in order to build the intended analytic recognizer.

3 Training

In this section, the fuzzy recognition training process is described, highlighting particularly the several modifications performed on the original recognizer.

The dominant features of a character consist of what is more common not to change between typing styles, such as the long vertical stroke in the b's and t's, for example. In this paper their extraction is performed through Gabor filter banks, which allow oriented feature extraction. The Gabor filter is a typical wavelet that offers localized operations [3]. Part of its parameters specify the Gabor wavelet direction φ

[3], as shown in Fig. 2, and the image region from which we intend to extract character features. Other parameters are character dependent, which means that the estimation has to take into account the image and the thickness of the writing [3]. For this reason, usually, their values are selected on a trial-and-error basis. For this paper, 12 wavelet directions were considered, from 0º to 180º.

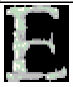
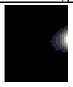
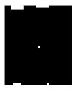
Original image	Gabor filter (φ = 0º)	After applying the filter
		

Fig. 2. Example of oriented feature extraction using Gabor filters

The filtering process is virtually identical to that in [1]. However, the original system required a time-consuming alignment algorithm, needed to compensate for variations in character spacing and shape and normalize word bounds. This procedure was disabled because it can produce heavy distortion when feature match is not effective. A single character has a smaller number of dominant features; printing flaws, common in this context, complicate feature matching even further. Character segmentation is also tight and accurate from the beginning.

As each training sample of the same character contains essentially the same extracted features structure, the major structural components can be established by adding the standardized images together, which forms the composite image for each trained character. Character images are classified manually; ancient characters are labeled as their present-day equivalent, therefore solving the problem of generating standard ASCII text from these occurrences.

Unlike the original system as presented in [1], information regarding image aspect ratios is also stored. The average aspect ratio ar_j for each group j is defined as:

$$ar_j = \frac{\sum_{k=1}^{N_j} ar(I_{j,k})}{N_j} \; . \tag{1}$$

where $I_{j,k}$ is the k-th sample image for character group j, N_j is the total number of images for group j and $ar(I)$ is the quotient between the width and the height of image I. The ar_j values are used as further assistance in the classification of new characters.

Before the classification process can take place, a set of fuzzy membership functions is generated for each character group and each orientation based on the extracted features of the training images. These intend to provide a description of the image features for use within the recognition algorithm.

Membership function generation begins by thresholding the feature image. Upper and lower thresholds H_u and H_l respectively are used to binarize the image [4], finding the upper and lower boundaries for each feature. These thresholds depend on the maximum intensity of the image and on constants c_u and c_l, set at 0.4 and 0.25, respectively. These values are determined empirically [1], but a standard binarization method [5] may also be effective. Two bounding rectangles are found around each extracted feature [6], corresponding to each intensity threshold, as shown in Fig. 3(a).

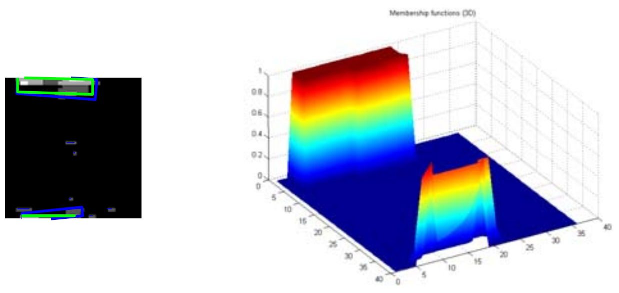


Fig. 3. (a) Bounding rectangles around extracted features; (b) Generated membership functions

Each rectangle pair is used to build a partial membership function. Its value $\mu(x, y)$ is 1 in the inner rectangle area and zero outside the outer rectangle. The vertices of a rectangle pair are linked based on Euclidean distance minimization. The intermediate function values are interpolated [7], forming a twisted trapezoidal shape. To do so, the domain is divided into 13 regions; the equations used to compute the function values can be found in [1]. An example is graphically represented in Fig. 3(b).

The global membership function for a given orientation i, denoted as $a_i(x, y)$, is defined as the maximum of the partial membership functions at each point, to deal with cases in which features overlap, thus taking into account only the most relevant feature. These maximum values are found continuously during the membership function generation process, in order to minimize resource usage.

4 Classification

The next step in the OCR process is the classification of new characters. The input for this stage is a character image V and the training structures. First, the image is run through Gabor filter banks, a process already applied in the training phase. The intensity of point (x, y) of the resulting image is denoted as $V'_i(x, y)$ for orientation i.

The objective is to compare the image features with the membership function values. Weights $w_{ij}(x, y)$ were assigned to each image point (x, y), for each orientation i and each character group j, to measure its influence, related mostly to membership function values. Similar points should be assigned a positive weight, and those that prove dissimilarity should penalize the rating. Weights are calculated according to:

$$w_{ij}(x, y) = V'_i(x, y), \text{ if } a_{ij}(x, y) = 0 \,. \quad \textbf{(2)}$$

$$w_{ij}(x, y) = w'_{ij}(x, y), \text{ if } a_{ij}(x, y) \neq 0 \,. \quad \textbf{(3)}$$

Equation (2) assures that points where V'_i is not zero, but the membership function is, will be penalized. This happens when a feature does not match any of orientation i for word group j. In this case, the value lowers the computed similarity rating. In [1], a somewhat different formal notation is used, in part to account for the distinction between the various partial membership functions. In this paper, the partial functions, each corresponding to one feature, are iteratively combined, simplifying notation and improving computational resource usage.

The rate of significance of point (x, y) in (3) is denoted by $w'_{ij}(x, y)$. Being N_c the total number of character groups, $w'_{ij}(x, y)$ is given by:

$$w'_{ij}(x, y) = 0, \text{ if } N_+ = 0 \,. \tag{4}$$

$$w'_{ij}(x,y) = \frac{N_c + N_+}{(N_c - 1) \times N_+} \sum_{j=1}^{N_c} a_{ij}(x,y), \text{ if } N_+ \neq 0 \,. \tag{5}$$

where N_+ is defined as the number of word groups, for each orientation and each point, with a positive membership grade. It attempts to formalize the intuitive concept that point (x, y) is a distinguishing factor among character groups, when only a restricted set of groups has membership values for that point.

Character classification is based on calculating a similarity matrix S. Each row refers to a feature orientation i and each column to a character group j. The matrix entries S_{ij} are calculated as in [1]. The final classification step modifies the simple additive weighted (SAW) method [8] used in the holistic recognizer [1]. The test character is classified as belonging to the character group identified by the j^* index, which is defined as:

$$j^* = \arg\max_j \frac{\sum_{i=1}^{N_w} v_{ij} \times S'_{ij}}{\sum_{i=1}^{N_w} v_{ij}} \times r_j(V) \,. \tag{6}$$

where v_i, S'_{ij} and r_j are determined by:

$$v_{ij} = \frac{\sum_{x,y} a_{ij}(x,y)}{\sum_{i,x,y} a_{ij}(x,y)}, \quad S'_{ij} = \frac{S_{ij}}{\max_i S_{ij}}, \quad r_j(V) = \min\left(\frac{ar(V)}{ar_j}, \frac{ar_j}{ar(V)}\right) \,. \tag{7}$$

The new $r_j(V)$ factor compares the character image V aspect ratio with the average aspect ratio ar_j of class j. The value of $r_j(V)$ is never greater than 1 and decreases as the correspondence between the two aspect ratios decreases. Therefore, matching an image with a given class is more likely when the aspect ratios are more similar. This change from [1] increases recognition success.

5 Post-processing

After the core OCR stage of the recognition has been performed, a post-processing stage ensures that the final results are meaningful and as fine-tuned as possible. This stage consists in using a dictionary and a word distance metric to improve the incorrect recognition results which were caused by faded and misprinted characters, irregular character and word spacing, speckles and smudges.

The dictionary used was GNU ASpell, a free open-source spell checker [9], which can suggest a rich set of words as possible corrections. It is still necessary to select one out of the several suggestions returned by the ASpell routines. In order to do so, the suggestion "closest" to the original word can be found by using the standard Levenshtein (also called edit) distance metric [10]. It is defined for strings of potentially different lengths and considers the least-cost path from one string to another, in terms of character insertion, deletion and substitution. The importance of this versatile behavior lies in the fact that recognition can incorrectly separate single characters into several or merge a couple of characters into one.

The Levenshtein metric requires an auxiliary function r that compares two characters within a string, as defined by:

$$r(c_1, c_2) = 0, \text{ if } c_1 = c_2; r(c_1, c_2) = 0, \text{ if } c_1 \neq c_2 . \tag{8}$$

where c_1 and c_2 are string characters. Let w_1 and w_2 be two word strings and $w(i)$ be the i-th character in word string w. To compute the Levenshtein value for w_1 and w_2, of lengths m and n, respectively, a $(m+1) \times (n+1)$ matrix D must be built, with row indices $i = 0, 1, \ldots, m$ and column indices $j = 0, 1, \ldots, n$. The actual Levenshtein distance $L(w_1, w_2)$ is finally stored in $D_{m,n}$. Matrix D is initialized by:

$$D_{i,0} = i; i = 0, 1, \ldots, m; D_{0,j} = j; j = 0, 1, \ldots, n . \tag{9}$$

All other remaining values are calculated recursively by:

$$D_{i,\mathrm{j}} = \min(D_{i-1,j} + 1, D_{i,j-1} + 1, D_{i-1,j-1} + r(w_1(i), w_2(j))) . \tag{10}$$

Conventional spell checking systems are optimized for typing mistakes. It is useful to take advantage of specialized information on the errors found when performing recognition on ancient documents. By manually proof-reading and correcting raw, unfiltered recognized data, statistical information is built about the most common errors made by the OCR software. This knowledge is used to improve the results of the two developed post-processing filters: a spelling corrector and a word splitter. Word confidence is computed as the average of its character confidence values and only words below a given confidence threshold are post-processed. The notation $c_1 \rightarrow c_2$ refers to an error consisting of misrecognizing true character c_1 as character c_2, while $n(c_1, c_2)$ represents the number of registered $c_1 \rightarrow c_2$ errors and $N(c_2)$ the number of $c \rightarrow c_2$ errors in the whole registry for every character c. The base concept of character error probability p for given characters c_1 and c_2 is defined as follows:

$$p(c_1, c_2) = 0, \text{ if } N(c_2) = 0;\ p(c_1, c_2) = \frac{n(c_1, c_2)}{N(c_2)}, \text{ if } N(c_2) \neq 0 . \tag{11}$$

The error probability $e(c)$ for character c is defined as the maximum of $p(c_1, c)$ for every character c_1. A character with error probability lower than a set threshold is not considered by the fuzzy algorithm. Its original FineReader recognition is kept instead.

If a given word w is not found in the spell checker dictionary, spell correction minimizes a given function across all suggested corrections s for the word w. This function assigns a value to a possible correction according to the likelihood of the recognition process making such a mistake. It should be greater for a lesser error likelihood. The proposed function combines error probability information with the

Levenshtein algorithm. It is computed in exactly the same way as the standard Levenshtein expression, except instead of (10), the following equation is used:

$$D_{i,j} = \min(D_{i-1,j} + 1, D_{i,j-1} + 1, D_{i-1,j-1} + 1 - p(w_1(i), w_2(j))) \ . \qquad \textbf{(12)}$$

The third argument to the min operator corresponds to a character substitution. This expression allows for the use of the Levenshtein distance in association with the previously collected data concerning common recognition errors.

Word splitting is also a word filter and its objective is potentially dividing a word into two or more strings, correcting frequent pre-processing mistakes that join adjacent words. These are mostly caused by extraneous graphical elements or printing defects. Fig. 4 shows two words joined due to their proximity and smudges.

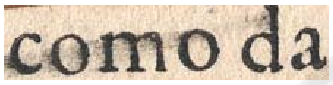


Fig. 4. Two words, “como” and “da”, joined by the extraction process

Word splitting works in text space and returns a list containing the words the original is divided into. The notation $w_1|w_2$ stands for a split of a given word w into two words w_1 and w_2; w is the concatenation of w_1 with w_2 and w_2 can be empty, in which case w and w_1 are equal. The pseudo-code for the algorithm is presented next:

```
WordSplit(Word w)
    Find best division w1|w2 of w;
    If w2 is not empty, ResultList = WordSplit(w2);
    Else, ResultList = new empty word list;
    Add w1 to front of ResultList;
    Return ResultList.
```

A word can be recursively sub-divided by this process until the best possible division is no division at all. Finding the best division of any given word is the crucial step. Since evaluating word splits is not straightforward, a heuristic function v was developed. The best division $w_1|w_2$ of w minimizes $v(w, w_1)$; w_2 is considered in the recursive call to *WordSplit*. This heuristic uses a function s defined as:

$$s(w) = \min \{L(w, w_s) : w_s \in S_w\} \ . \qquad \textbf{(13)}$$

where L is the weighted Levenshtein applied in spell correction and S_w is a set containing all possible suggested corrections to w. Finally, the heuristic used is:

$$v(w, w_1) = \exp(a \times s(w_1)) \times \log(b \times n) - c \times n_1 \ . \qquad \textbf{(14)}$$

where a, b and c are positive parameters, adjusted empirically, and n and n_1 are the lengths of words w and w_1 respectively. This heuristic combines three criteria, assigning each one a certain degree of relevance. The priority factor is the value of $s(w_1)$, i.e. the likelihood that w_1 corresponds to a legitimate isolated word, relying on suggested words distance, since some characters may have been misrecognized. Next, the new word length is taken into account. Splitting into many small words is avoided, aiming instead towards larger, correct strings. Finally, the original word length is considered, in order to scale the other values, which otherwise led to disparate results among a set of words. The use of exponential-, logarithmic- and linear-growth factors is based on the relative importance of each, adjusted through experimentation. Although spell correction was presented first, it is executed last, so that improperly separated words do not interfere with the spell correction, which was built assuming that a given string is similar, in length and content, to the written word.

6 Results

In this section, test results, gathered for the purpose of analyzing the software performance and correctness, are presented. Before large tests could be conducted, a training character database was created from 1980 alphabetic characters, classified manually. Recognition is possible once the training structure has been generated. In order to test various development options, the training data from this database is used in the classification of another previously identified 1580 character set, which works as a validation set. The final system achieved a per-character success rate of 87.5%.

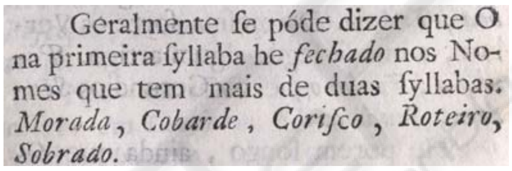

Geralmente ſe póde dizer que O
na primeira ſyllaba he *fechado* nos No-
mes que tem mais de duas ſyllabas.
Morada, *Cobarde*, *Coriſco*, *Roteiro*,
Sobrado.

Fig. 5. Sample from 20 page test set

Verifying the results is very time-consuming, so the executed tests were selected carefully within practical limits in order to be representative and closely convey the application performance in an actual common usage environment. The main test set consists of 20 pages acquired with variable scanning conditions, namely skewing and paper see-through, with both non-italic and italic text. It contains 1886 words consisting in 8034 characters, as segmented by the FineReader engine. The source book [11] concerns Portuguese language orthography, providing a large variety of characters and formatting properties. Fig. 5 shows a sample paragraph.

Table 1. Per-character success rates

System	**20 page set success (%)**
FineReader engine	86.9
Fuzzy recognizer	88.0

Per-character results for this test set are summarized in Table 1. Both systems successfully classified between 87% and 88% of the 8034 characters. The improvement introduced by the fuzzy recognizer is very slight, although consistent. Many errors occur due to printing defects and strong similarity between certain key characters. The fuzzy recognizer was unable to handle these problems, possibly owing to its holistic origins and the limited distinctive feature set of character images. On the other hand, the tests show that its performance is comparable to the FineReader engine OCR, a sophisticated commercial software, virtually matched here by this new fuzzy application.

An additional 12 page, 1590 word set, from a different book discussing the Portuguese language, was also used for some tests. It has diverse typesets and several printing problems, even though scanning quality is quite high. The two test sets were used to assess the standard FineReader and fuzzy recognizer outputs with and without post-processing filters. Table 2 shows the success rate for each of these cases in the two test sets. Recognition output was analyzed on a word basis; any word with at least one misclassified character is considered wrong; checking is case-insensitive and graphical accents are ignored. The available Portuguese dictionary contains strictly modern spelling, so it is unable to correct archaic words. An ancient Portuguese dictionary would be an immense improvement; several well-recognized words were actually damaged because their spelling is unknown to contemporary checkers.

Table 2. Per-word success rates

System	**20 page set success (%)**	**12 page set success (%)**
FineReader engine	62.9	34.6
Fuzzy recognizer	64.1	35.6
FineReader + post-processing	65.0	40.0
Fuzzy recognizer + post-processing	63.6	39.9

Per-word results can be considered unfair towards the FineReader and fuzzy recognizers, because these are character-based and not word-based. Most wrong words had few incorrect characters, explaining why analytic success rates are higher than holistic rates. However, a per-word check was performed to enable a comparison between the raw recognizer classification and the post-processed output. Table 2 shows that splitting and spell correction applied directly to the FineReader output improved the results noticeably when the recognizer performance was weaker (a nearly 16% improvement for the 12 page set), although there was little progress in the larger experience. However, the fuzzy recognizer output did not equally improve when post-processed. Results worsened for the 20 page set and were nearly identical to the post-processed FineReader results for the smaller set. This may be explained by fuzzy recognizer errors introduced when feature extraction fails or is insufficient. These can misguide the spell checker by introducing unexpected characters, more so because of the ancient spelling, unknown to modern dictionaries. Some post-processing side-effects, such as the excessive splitting of misrecognized words, could be minimized through parameter adjustments and, again, a more adequate dictionary.

7 Conclusions

This paper proposed an OCR system for 17th century documents based on fuzzy pattern recognition. The processing sequence was presented, from the training stage to the classification process, followed by post-processing according to language and error probability data. Finally, test results and procedure were summarized.

Building upon the FineReader engine, recognition improvements were noticeable with both fuzzy recognizer and dictionary-based post-processing. The former system achieved a success rate comparable to that of a mature commercial software package and is open to further enhancement. The output filters developed can increase the output trustworthiness, especially if the appropriate dictionary resources are available. Combining these two systems compatibly has not yet been fully accomplished.

Further work can include the development of an automatic parameter adjustment system based on measurable properties of the documents being processed, the definition of better word distance metrics, the introduction of more accurate heuristics and the development of an ancient word dictionary for improved spell checking.

Acknowledgements

This work was partly supported by: the “Programa de Financiamento Plurianual de Unidades de I&D (POCTI), do Quadro Comunitário de Apoio III”; the FCT project POSI/SRI/41201/2001; “Programa do FSE-UE, PRODEP III, no âmbito do III Quadro Comunitário de apoio”; and program FEDER. We also wish to express our acknowledgments to the Portuguese Bibioteca Nacional, whose continuous support has made possible this work.

References

1. R. Buse, Z.Q. Liu, J. Bezdek, “Word Recognition Using Fuzzy Logic”, in *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 1, Fev. 2001, pp. 65-76
2. João M.C. Sousa and Uzay Kaymak, "Fuzzy Decision Making in Modeling and Control", *World Scientific*, Singapore and UK, Dec. 2002
3. R. Buse, Z.Q. Liu, T. Caelli, “A structural and relational approach to handwritten word recognition”, in *IEEE Trans. Syst., Man, Cybern.*, vol. 27, no. 25, Oct. 1997, pp 847-861
4. Parker, J.R., “Algorithms for Image Processing and Computer Vision”, John Wiley & Sons, New York, USA, 1998
5. N. Otsu, "A threshold selection method from gray level histograms", *IEEE Transactions on Systems*, *Man, and Cybernetics*, vol. 9, pp. 62-66, 1979
6. Godfried Toussaint, “Solving Geometric Problems with the Rotating Calipers”, in *Proc. IEEE MELECON'83*, 1983, pp. A10.02/1-4
7. James D. Foley et al, “Computer Graphics – Principles and Practice”, Second Edition in C, Addison-Wesley, Reading, Massachussets, USA, 1990
8. C. L. Hwang, K. Yoon, “Multiple Attribute Decision Making, Methods and Applications, A State-of-the-Art-Survey”, Springer-Verlag, Berlin, Germany, 1981
9. K. Atkinson, GNU Aspell Homepage, *http://aspell.net*, GNU Project
10. V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", *Soviet Physics Doklady*, vol. 6, pp. 707-710, 1966
11. Álvaro Ferreira de Véra, “Orthographia ou modo para escrever certo na lingua Portuguesa”, 17th century, available at Biblioteca Nacional
12. ABBYY FineReader Homepage, http://www.abbyy.com, ABBYY Software House