Use of Lexicon Density in Evaluating Word Recognizers

Venu Govindaraju, *Senior Member*, *IEEE*, Petr Slavík, and Hanhong Xue, *Student Member*, *IEEE*

Abstract—We have developed the notion of lexicon density as a metric to measure the expected accuracy of handwritten word recognizers. Thus far, researchers have used the size of the lexicon as a gauge for the difficulty of the handwritten word recognition task. For example, the literature mentions recognizers with accuracy for lexicons of sizes 10, 100, 1,000, and so forth, implying that the difficulty of the task increases (and, hence, recognition accuracy decreases) with increasing lexicon sizes across recognizers. Lexicon density is an alternate measure which is quite dependent on the recognizer. There are many applications such as address interpretation where such a recognizer dependent measure can be useful. We have conducted experiments with two different types of recognizers. A segmentation-based and a grapheme-based recognizer have been selected to show how the measure of lexicon density can be developed in general for any recognizer. Experimental results show that the lexicon density measure described is more suitable than lexicon size or a simple string edit distance.

Index Terms—Classifier combination, handwritten word recognizer, lexicon density, performance prediction, edit distances.

1 Introduction

The task of word recognition is described as follows: Given an input word image and a lexicon of possible choices, the word recognizer must rank the lexicon in descending order of preference. The preference characterizes the "goodness of match" between the input image and a lexicon entry. Lexicon size has been the commonly used measure to categorize the difficulty of a recognizer's task [1].

Researchers have correctly observed that recognizers have more difficulty with large lexicons. The reason for this observation is simple—when lexicons are large, their entries are more likely to be "similar." The ability of a recognizer to distinguish among the entries in a lexicon clearly depends on how "similar" the lexicon entries are. The "similarity" among entries depends not only on the entries themselves but also on the recognizer.

Table 1 presents two lexicons of equal size (i.e., 5). However, the two lexicons present differing degrees of difficulty to the recognizers depending on the features they use. Assume, for example, that we have a word recognizer that recognizes only the first character of each word. Accuracy of such a recognizer is expected to be poor on a lexicon where all entries start with the same letter (Lexicon 2) and good on lexicons where starting letters of all entries are different (Lexicon 1). Similarly, a recognizer that estimates the length of each word performs well on lexicons where entries differ significantly in their length (Lexicon 2) and poorly on lexicons with entries of the same length (Lexicon 1).

We propose to address the relation between the difficulty posed by a lexicon and the features of the recognizer used. We will call the new measure that describes this relationship "Lexicon Density" or LD.

Central to the notion of lexicon density is the concept of "distance" between handwritten words. Distance between two words is usually measured in terms of the total cost of edit operations that are needed to transform one word into the other one. Previously, computing of such a distance was motivated by three edit operations: 1) *replacement* of one character by another character, 2) *deletion* of a single character, and 3) *insertion* of a single character [4] (where a *replacement* operation can be performed by a *deletion* followed by an *insertion*). While these edit operations are well-suited for applications where the characters in a string are nicely isolated (as in good quality machine printed text), they are inadequate in modeling applications with handwritten words. Additional operations that allow for merging and splitting of characters are necessary. We develop in this paper a new distance metric, called the *slice distance* (see Section 5), for this purpose.

A pertinent application with dynamically generated lexicons is presented by the street name recognition task in Handwritten Address Interpretation (HWAI). Here, lexicons are generally comprised of street name candidates generated from the knowledge of the zip code and the street number. Every instance of an address can have a different zip code and street number. Hence, every invocation of the word recognizer is presented with a different lexicon. For example, the zip code 60120 and street number 1121 produces a lexicon of 11 street names whereas the zip code 45405 and street number 3329 produces a lexicon of 2 street names. In fact, it is in such cases that the notion of lexicon density holds the greatest promise. If there are several recognizers to choose from, there should be a control mechanism that dynamically determines in any given instance which recognizer must be used. The determination can be based on the quality of the image, the time available, and the lexicon density. It could be decided, for instance,

- *The authors are with the Center for Excellence in Document Analysis and Recognition (CEDAR), Department of Computer Science and Engineering, State University of New York at Buffalo (SUNY), Amherst, NY 14228. E-mail: govind@cedar.buffalo.edu.*

Manuscript received 18 July 2000; revised 4 June 2001; accepted 22 Oct. 2001. Recommended for acceptance by T.K. Ho.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 112535.

TABLE 1
Recognizer 1 Recognizes only the First Characters of Words
Recognizer 2 only Counts the Number of Characters in Words

Lexicon 1:	Lexicon 2:
1: He	1: Me
2: Me	2: Memo
3: No	3: Memory
4: So	4: Memorize
5: We	5: Memoryless

Lexicon 1 is favorable to Recognizer 1 (less dense) and Lexicon 2 is favorable to Recognizer 2.

that, if the image is noisy a particular recognizer should be favored based on training data. Similarly, a specific recognizer might be rendered ineffective if the lexicon density is high from the recognizer's viewpoint. This could happen if the recognizer depends heavily on a feature, say the length, and all the lexical entries have the same length.

Another possible use of lexicon density is in evaluating recognition results. Imagine that we have to assign some confidence to the first choice. We could compare the matching scores of the first and second choices to determine how confident the recognizer is in its response. It would however be more meaningful to also consider how likely it is for the top few choices to be confused by the recognizer, i.e., compute the "local" density.

2 Previous Work

Hamming in 1982 [2], defined the distance between two strings of equal length as the number of symbol positions at which they differ. This is like finding the minimum cost of transforming one string into another by using only substitution operations of equal cost. The Levenshtein metric [4] allows for insertions and deletions and can handle strings of variable length.

Computing distances between two strings using dynamic programming has been independently discovered by several researchers [6]. Wagner and Fisher [7] describe a dynamic program that computes the minimum edit distance between two ASCII strings as the cost of a cheapest "trace" of the three elementary edit operations. Intuitively, a *trace* is a special sequence of elementary edit operations that transforms the first string into the second, while requiring at most one edit operation per character and enforcing a strict left-to-right order of elementary edit operations [7]. Under the assumption that the cost of elementary edit operations satisfy the triangle inequality, Wagner and Fisher have proven that the cost of the cheapest trace is indeed the same as the cost of the cheapest sequence of elementary edit operations.

Seni et al. [5] have studied the problem of finding the minimum edit distance between handwritten words by considering additional edit operations: *substitution* of a pair of characters, *merge* of two characters into one, and a *split* of one character into two. Similar to the work of Wagner and Fisher [7], Seni et al. computed the elm edit distance as the cost of the cheapest trace. The costs of elementary edit operations were determined empirically by observing samples of handwritten characters and examples of misrecognition. Based on these observations, the authors decide whether an elementary transformation is VERY LIKELY, LIKELY, UNLIKELY, or IMPOSSIBLE and then assigned a (rather arbitrary) cost to each edit operation based on the type of operation and the level of likelihood. According to Seni et al., the weights of the elementary transformations are independent of any particular recognizer and the generalized edit distance represents a *generic* measure of possible confusion between two handwritten words.

While the additional edit operations accounts for some of the errors in handwriting recognition (like confusing "cl" and "d"), it is still not general enough to explain errors such as the confusion of "rst" with "on" in Fig. 4. We describe in this paper the slice distance which adequately addresses the typical misrecognitions of handwritten words and phrases.

3 Lexicon Density

Distance between ASCII words ($w1$ and $w2$) with respect to a given recognizer is central to the definition of LD. It reflects the propensity of confusing words $w1$ and $w2$. To determine a distance that captures this notion is relatively easy for a holistic recognizer that treats the entire word as a single object. One simply computes in the feature space of all recognizable words, the distances between all prototypes of $w1$ and all prototypes of $w2$ and defines the distance between $w1$ and $w2$ as minimum (or average) of all such feature distances.

The speech recognition community realized the need for such a measure and defined the notion of perplexity [1] as the expected number of branches (possible choices of the next event) in a stochastic process. It is possible to compute a similar quantity for our purpose. One would use the given lexicon to build a prefix tree and calculate the average branching factor. This will yield another possible measure for evaluating word recognizers and would be a competing alternative to lexicon density. However, it would be somewhat inadequate as it would favor words sharing prefixes and ignore those sharing suffixes.

The methodology for defining LD is nontrivial when dealing with segmentation based recognizers, where each letter is considered a separate entity. One possibility is to use the minimum edit distance between $w1$ and $w2$ [7]. However, this approach is limited to cases where recognizers are able to correctly segment a word image into characters *without* having to recognize the characters first, as is the case with recognizers in the machine print domain. One uses samples of training words and training characters to determine the cost of elementary edit operations (deletion, insertion, and substitution) with respect to a given recognizer. This paper is focused on handwritten words where it is usually not possible to segment the word into its constituent characters unambiguously.

Given a word recognizer R, we denote by $d_R(w1, w2)$ the distance between two ASCII words $w1$ and $w2$. The distance is image independent and recognizer dependent. It is supposed to measure the propensity of confusing words $w1$ and $w2$ by recognizer R. We will define d_R in Section 5. Clearly, if words are "closer" the density should be large.

Given a recognizer R and a lexicon L of words $w_1, \ldots, w_n$, lexicon density is defined as

$$\rho_R(L) = v_R(L)(f_R(n) + \delta_R),$$

where

$$v_R(L) = \frac{n(n-1)}{\sum_{i \neq j} d_R(w_i, w_j)}$$

is the reciprocal of the average distance between word pairs, n the lexicon size, $f_R(n)$ an increasing function of n, and δ_r a recognizer dependent constant. The use of δ_R allows easy examination of sets of functions for $f_R(n)$. For example, $\ln \frac{n}{2} = \ln n - \ln 2$ can be easily examined by letting $\delta_R = \ln 2$.

The performance $p_R(L)$ of recognizer R on lexicon L changes approximately linearly with lexicon density $\rho_R(L)$, which means there exist some a and c such that $a\rho_R(L) + c = p_R(L)$ approximately holds. Consider the square error of the approximation over a set of lexicons Γ.

$$E_R = \sum_{L \in \Gamma} \left(a\, v_R(L) f_R(n) + a\, v_R(L)\delta_R + c - p_R(L)\right)^2. \quad (1)$$

We minimize the square error E_R by selecting $f_R(n)$ and δ_R. Suppose $f_R(n)$ is already known. The minimization of E_R can be obtained by letting $b = a\delta_R$ and finding the solution for the following linear equations,

$$\begin{cases} \frac{\partial E}{\partial a} = 2(a \sum v^2 f^2 + b \sum v^2 f + c \sum vf - \sum vfp) = 0 \\ \frac{\partial E}{\partial b} = 2(a \sum v^2 f + b \sum v^2 + c \sum v - \sum vp) = 0 \\ \frac{\partial E}{\partial c} = 2(a \sum vf + b \sum v + c \sum 1 - \sum p) = 0, \end{cases}$$

which is equivalent to a multiple regression problem (subscripts R have been omitted for better readability). Therefore, the first step is to determine the proper $f_R(n)$ and the second step is to determine δ_R given $f_R(n)$. We will consider two sets of functions, $\{f_R(n) = ln^p n | p\}$ and $\{f_R(n) = n^p | p\}$.

4 Word Recognizers

We will use two word recognizers for illustration. The first recognizer is a segmentation based word recognizer [3] (Fig. 1), henceforth referred to as WR-1. The second recognizer is a grapheme-based recognizer which simply extracts and recognizes certain subcharacter features without any explicit character segmentation [8] (Fig. 3). Henceforth, this recognizer will be referred to as WR-2.

Given an image of handwritten word, WR-1 segments the word at potential character boundaries. An over-segmentation strategy is used wherein it is acceptable if a character is segmented into subcharacters (oversegmentation) but no two characters should remain connected after the segmentation process (Fig. 2). It is assumed that the segmentation process never segments a character into more than four parts. A graph is obtained where each segment is a node. Possible traversals of this graph through all the nodes constitute various interpretations of the segments. Instead of passing on combinations of segments to a generic OCR, lexicon is brought into play early in the process. A combination of adjacent segments (up to a maximum of 4) are compared to only those character choices which are possible at the position in the word being considered. The approach can be viewed as a process of accounting for all the segments generated by a given lexicon entry. Lexicon entries are ordered according to the "goodness" of match.

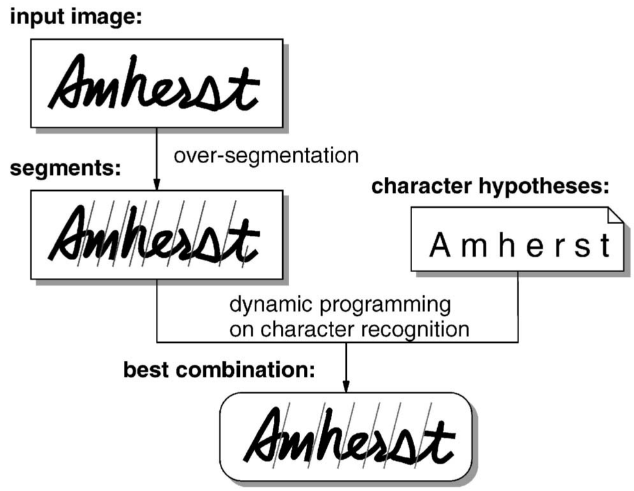


Fig. 1. Segmentation-based recognizer, WR-1.

WR-2 can be briefly described as follows: Suppose Σ is the set of all possible features. A character model is an automaton $M =< S, T, l, \pi, \sigma >$, where

- S is a finite set of states.
- $T \subseteq S \times S$ is the set of transition arcs.
- $l : S \rightarrow \Sigma$ is the labeling function.
- $\pi \subseteq S$ is the set of starting states.
- $\sigma \subseteq S$ is the set of accepting states.

Character models are concatenated to obtain word models. Suppose a word consists of N[1] characters with corresponding character models $m_i = < S_i, T_i, l_i, \pi_i, \sigma_i >, 1 \leq i \leq N$. The word model $M = < S, T, l, \pi, \sigma >$ is defined as follows:

- $S = \cup_{i=1}^{N} S_i$.
- $T = \left(\cup_{i=1}^{N} T_i\right) \cup \left(\cup_{i=1}^{N-1} \sigma_i \times \pi_{i+1}\right)$.
- $l(x) = l_i(x)\ if\ x \in S_i$.
- $\pi = \pi_1$.
- $\sigma = \sigma_N$.

During recognition, the input feature sequence, which can be viewed as a degraded automaton, is matched against word models one by one using the same dynamic programming procedure described in Section 5.1.2.

5 Computation of Distance, d_R

The definition of LD depends on the word recognizer. In the following sections, we will describe the computation of the distance d_R for the two recognizers WR-1 and WR-2. Henceforth, we will call the distances corresponding to WR-1 as "slice distance" and the distance corresponding to WR-2 as "grapheme distance." Based on our illustrations, it should be apparent to the reader that d_R can be computed for any given word recognizer based on an understanding of its methodology. It should be also noted that for a word

1. We are using n for the size of a lexicon and N for the number of characters in a word.

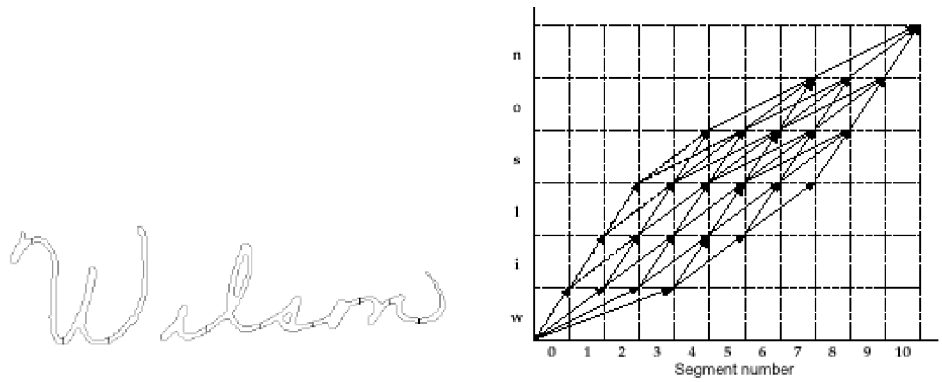


Fig. 2. Segmented handwritten image in contour representation and corresponding graph. Details of the method can be seen in [2].

recognizer whose methodology uses character segmentation (such as WR-1) the process of computing the distance d_R is more involved when compared to recognizers that do not rely on character segmentation (such as WR-2). This is primarily because it is the process of character segmentation that adds the operations of splitting and merging characters to the standard string edit operations.

5.1 Slice Distance d_R for WR-1

Assume that WR-1 is presented with a word image (Fig. 2) and the lexicon consists of just two entries—"Wilson" and "Amherst." After dynamically checking all the possible segment combinations, WR-1 would correctly determine that the best way to match the image and the word "Wilson" is to match segments 1-4 with "W," segment 5 with "i," segment 6 with "l," etc. The best way to match the image against "Amherst" would be to match segment 1 with "A," segments 2-5 with "m," segment 6 with "h," segment 7 with "e," segment 8-9 with "r," segment 10 with "s," and, finally, segment 11 with "t," (Fig. 4). The score of the second matching would be lower than the score of the first matching leading to the recognizer correctly choosing "Wilson" as its first choice.

Fig. 4 illustrates how confusions could possibly arise in determining the best possible answer. Letter "A" was matched with the same slice of the image as the left part of letter "W," left part of letter "m" was matched with the same "slice" of the image as the right part of "W," right part of letter "m" was matched with the same slice of the image as letter "i," etc. Hence, to determine the propensity of confusing "Wilson" and "Amherst," we have to first determine the propensity of confusing "A" with the left part of "W," left part of "m" with the right part of "W," right part of "m" with a complete letter "i," and so forth. In general, we need to determine the propensity of confusing a slice of one character with a slice of another character. Furthermore, since the slice distance is image independent, we have to consider all the possible ways of confusing slices of characters over all writing styles. In other words, we have to consider all possible segmentation points that can occur in any image of a given word and all possible ways of matching them with lexicon entries. Then we choose the worst-case scenario (i.e., the smallest distance) among all possible combinations. This measure depends ultimately on the distance between character "slices."

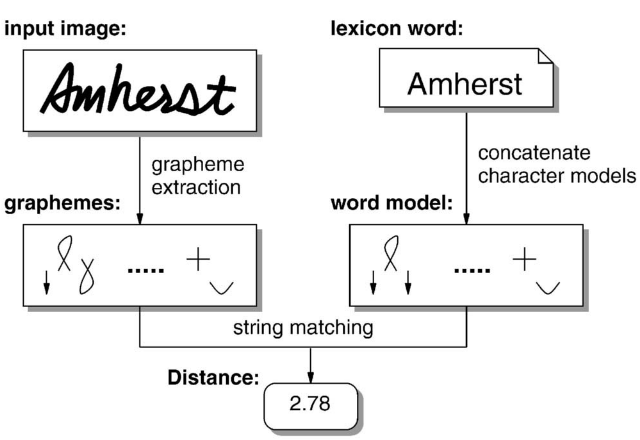


Fig. 3. Grapheme-based recognizer, WR-2.

Computation of the slice distance involves the following two steps: 1) determining the elementary distances between all meaningful slices of characters and 2) using these elementary distances as weights in a dynamic program that computes the slice distance between any two ASCII words.

5.1.1 Elementary Distances

Elementary distances between slices of different characters are computed during the training phase of WR-1 and stored in several 26 by 26 confusion matrices. These matrices are a natural generalization of confusion matrices between complete characters.

During training, WR-1 is presented with several thousand images of handwritten words. WR-1 oversegments each image and the elementary segments are then *manually* combined into complete characters. These characters then serve as character templates.[2]

The training program not only stores the complete characters, but also all the slices. *Slice* is a part of a character consisting of several consecutive segments. Each slice is stored together with the information about the parent character class, number of segments in the slice, and the part of the character being considered (left, right, or middle).

2. To be precise, characters are first clustered based on the similarity of their features. Then, for each cluster, the average of all characters in that cluster is used as template.

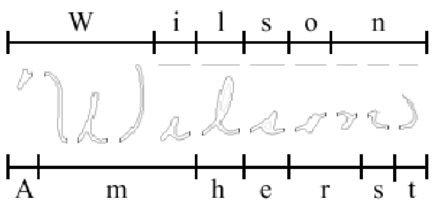


Fig. 4. Matching of ASCII words (from lexicon) with image.

The slices are used to compute all meaningful elementary distances. For example, to compute the elementary distance between the left part of "A" and the right part of "B" such that both consist of exactly two elementary segments, distances between all 2-segment left slices of "A" and all 2-segment right slices of "B" are computed, and the elementary distance is set to their minimum.

Since WR-1 allows between 1 and 4 elementary segments per character, for each character, we have extracted left and right slices of 1 to 3 elementary segments, middle slices of 1 and 2 segments (from characters of size 3 and 4, respectively), and 1-segment middle-left and middle-right slices (from characters of size 4). Using such slices, we compute 17 26 x 26 confusion matrices.

- $SS1$, $SS2$, $SS3$, $SS4$: elementary distances between complete characters of sizes 1, 2, 3, and 4, segments, respectively;
- $LR1$, $LR2$, $LR3$: elementary distances between left slices and right slices of sizes 1, 2, and 3, segments, respectively;
- $LS1$, $LS2$, $LS3$: elementary distances between left slices and complete characters of sizes 1, 2, and 3, segments, respectively;
- $RS1$, $RS2$, $RS3$: elementary distances between right slices and complete characters of sizes 1, 2, and 3, segments, respectively;
- $MS1$, $MS2$: elementary distances between middle slices and complete characters of sizes 1, and 2, segments, respectively;
- LMS, RMS: elementary distances between left and right middle slices and complete characters of size 1 segment.

Note, that there is no point computing distances like $RM1$, $LL2$, etc. If, for example, right part of one character is being confused with a middle part of another character, both parts can be "extended" to the left, to get one of the following three cases already covered by the elementary distances—RLi, SMi, or SLi, where $i \geq 2$.

Definitions. A *hypothetical* word image in the context of the discussions in this paper, refers to the following two parts of a word image: the chosen form of the word and its chosen segmentation. Thus, a hypothetical image can potentially represent a word image of any form and any segmentation. The number of forms in which an image of a word can appear is infinite. However, the possible segmentations are limited by the assumptions made by the word recognizer under consideration. For example, both WR-1 and WR-2 guarantee that a single character is split into at most four characters and no two characters are left merged by the segmentation methodology. This assumption limits the possible segmentations of a word image. We assume that in a hypothetical image all the forms of a word are equivalent as long as their segmentations do not differ.

The *Elementary Slice Distance* is the minimum (Euclidean) distance between feature vectors of slices of character templates obtained during the training phase.

The *Base Slice Distance* is the sum of elementary distances between character slices for a particular way of confusing two ASCII words. We use hypothetical images of a particular number of segments and a particular way of matching these segments with individual characters for this purpose.

The *Slice Distance* between words $w1$ and $w2$ is the minimum over all possible base distances considering all possible ways of segmenting a hypothetical image and all possible ways of matching segments with individual characters.

We denote the minimum slice distance between two ASCII words $w1$ and $w2$ by $msd(w1, w2)$.

5.1.2 Dynamic Program

Checking *all* the possible ways of confusing words $w1$ and $w2$ is not very efficient. Fortunately, there is a dynamic program that computes the slice distance in time $O(|w1| \cdot |w2| \cdot (|w1| + |w2|))$.

Let us first define some concepts needed to describe the dynamic program. Given words $w1$ and $w2$, we denote by $n1$ and $n2$ the number of characters in each word, i.e., $n1 = |w1|$ and $n2 = |w2|$. We denote by c_i the ith character of word $w1$ and by d_j the jth character of word $w2$. Hence, $w1 = c_1 c_2 \ldots c_{n1}$ and $w2 = d_1 d_2 \ldots d_{n2}$. We denote by max_K the maximum number of segments of a hypothetical image matching both $w1$ and $w2$.

Let w be a word of n characters and let m be a matching between w and a hypothetical image of K segments. We say that ith character of word w *ends* at segment k if k is the last segment of the supersegment that matches the ith character. For a particular way of matching the word and the image, we denote this ending segment by $m(i)$, $1 \leq i \leq n$. Clearly $m(|w|) = K$. Additionally, we define $m(0) = 0$.

Since WR-1 allows a complete character to match between 1 and 4 consecutive segments of the image, $1 \leq m(i) - m(i-1) \leq 4$ and max_K $= 4 * \min(|w1|, |w2|)$.

We store the partial results of our computation in the following 3- or 4-dimensional matrices ($i = 1, \ldots, n_1$, $j = 1, \ldots, n_2$, $k = 1, \ldots,$ max_K and $e = 1, \ldots, 3$):

fese[i][j][k]: The smallest possible distance between the first k segments of word $w1$ and the first k segments of word $w2$ under the condition $k = m(i) = m(j)$. The name fese stands for "first-even, second-even," and corresponds to those matchings where the ith and jth characters of words $w1$ and $w2$ are aligned, i.e., their ending segments coincide. For example, in Fig. 4, the cost of matching "Wi" with "Am," is stored in fesc[2][2][5].

fesc[i][j][k][e]: The smallest possible distance between the first k segments of word $w1$ and the first k segments of word $w2$ under the condition $k = m(i) = m(j) - e$. The name fesc stands for "first-even, second-cut" and corresponds to those matchings where the ith character of $w1$ ends within jth character of $w2$, and there are exactly e segments of jth character left "sticking out" beyond the last segment of the ith character. For example, in Fig. 4, the cost of (partially) matching "Wilso" with "Amher," is stored in fesc[5][5][8][1].

fcse[i][j][k][e]: The smallest possible distance between the first k segments of word $w1$ and the first k segments of word $w2$ under the condition $k = m(i) - e = m(j)$. The name fcse stands for "first-cut, second-even" and corresponds to those matchings where the jth character of $w2$ ends within the ith character of $w1$, and there are exactly e segments of ith character left "sticking out" beyond the last segment of jth character. For example, in Fig. 4, the cost of (partially) matching "Wilson" with "Amher," would be stored as fcse[6][5][9][2].

Most of the elements of matrices fese[][][], fesc[][][][], and fcse[][][][] do not contain meaningful values since there are no images and matchings that satisfy the required conditions. There are two ways of dealing with the situation.

1. Initialize all the elements of all the three matrices to ∞ (with the exception of fese[0][0][0]) and then compute all the elements starting with $k = 1$.
2. Limit the ranges of k, i, j, and e to meaningful values, thus avoiding unnecessary computations.

Since most of the elements of matrices fese[][][], fesc[][][][], and fcse[][][][] are meaningless, we have used the second approach.

The dynamic program now consists of the following three steps:

1. Initialization:

$$\texttt{fese}[0]\,[0]\,[0] = 0.$$

2. Compute the (meaningful) values of matrices fese[][][], fesc[][][][], and fcse[][][][] for k segments (starting with $k = 1$, $k = 2$, up to $k = \texttt{max\_K}$) from the (meaningful) values for $k-1$, $k-2$, $k-3$, and $k-4$ segments using the following formulas:

$$\texttt{fese}[i][j][k] = \min \begin{cases} \min_{r=1,\ldots,4} \texttt{fese}[i-1][j-1][k-r] + \texttt{SS}r[c_i][d_j] \\ \min_{r=1,\ldots,3} \texttt{fesc}[i-1][j][k-r][r] + \texttt{SR}r[c_i][d_j] \\ \min_{r=1,\ldots,3} \texttt{fcse}[i][j-1][k-r][r] + \texttt{RS}r[c_i][d_j] \end{cases}$$

$$\texttt{fesc}[i][j][k][e] = \min \begin{cases} \min_{r=1,\ldots,4-e} \texttt{fese}[i-1][j-1][k-r] + \texttt{SL}r[c_i][d_j] \\ \min_{r=1,\ldots,4-e} \texttt{fcse}[i][j-1][k-r][r] + \texttt{RL}r[c_i][d_j] \\ \min_{r=1,2} \texttt{fesc}[i-1][j][k-r][r + \texttt{SM}r[c_i][d_j] \text{ for } e = 1 \\ \texttt{fesc}[i-1][j][k-1][1] + \texttt{SMR}[c_i][d_j] \text{ for } e = 1 \\ \texttt{fesc}[i-1][j][k-1][1] + \texttt{SML}[c_i][d_j] \text{ for } e = 2. \end{cases}$$

$$\texttt{fcse}[i][j][k][e] = \min \begin{cases} \min_{r=1,\ldots,4-e} \texttt{fese}[i-1][j-1][k-r] + \texttt{LS}r[c_i][d_j] \\ \min_{r=1,\ldots,4-e} \texttt{fesc}[i-1][j][k-r][r] + \texttt{LR}r[c_i][d_j] \\ \min_{r=1,2} \texttt{fcse}[i][j-1][k-r][r] + \texttt{MS}r[c_i][d_j] \text{ for } e = 1 \\ \texttt{fcse}[i][j-1][k-1][1] + \texttt{MRS}[c_i][d_j] \text{ for } e = 1 \\ \texttt{fcse}[i][j-1][k-1][1] + \texttt{MLS}[c_i][d_j] \text{ for } e = 2. \end{cases}$$

3. Compute the minimum slice distance between $w1$ and $w2$. Assuming that we know all values in matrices fese[][][], fesc[][][][], and fcse[][][][] the slice distance between $w1$ and $w2$ is given by

$$msd(w1, w2) = \min_k \texttt{fese}[n1][n2][k].$$

The formulas above are all straightforward and simply enumerate all the possible ways of matching two characters to the same parts of the image. Now what remains is to provide the reader with the details of determining meaningful ranges of i, j, k, and e.

Step 1: Determine the range of K—the total number of elementary segments. Since WR-1 allows between 1 to 4 elementary segments per complete character, two words $w1$ and $w2$ can be possibly confused on a hypothetical image that gets segmented by WR-1 into K segments, where

$$\texttt{min\_K} = \max(|w1|, |w2|) \leq K \leq \texttt{max\_K} = 4 * \min(|w1|, |w2|).$$

If no such K exists (that is, $\texttt{min\_K} > \texttt{max\_K}$, one word is more than times times longer than the other), we set $msd(w1, w2) = \infty$.

Step 2: Compute arrays "min_dur[]" **and** "max_dur[]." Given a word w with $|w| = n$ and the values min_K and max_K such that $\texttt{min\_K} \geq n$ and $\texttt{max\_K} \leq 4n$, we define the arrays min_dur[] and max_dur[] as

$$\texttt{min\_dur}[i] = \min \left\{ m(i) \middle| \begin{array}{l} m \text{ varies over all possible matchings between } w \\ \text{and } K \text{ segments with } \texttt{min\_K} \leq K \leq \texttt{max\_K} \end{array} \right\}$$

and

$$\texttt{max\_dur}[i] = \max \left\{ m(i) \middle| \begin{array}{l} m \text{ varies over all possible matchings between } w \\ \text{and } K \text{ segments with } \texttt{min\_K} \leq K \leq \texttt{max\_K} \end{array} \right\}$$

for $i = 0, \ldots, n$.

Given a word w and an image consisting of K elementary segments, the last segment of the first character can be segments 1, 2, 3, or 4; the last segment of the second character can be segments 2, 3, ..., up to segment 8, etc. Hence, $\texttt{min\_dur}[1] = 1$, $\texttt{max\_dur}[1] = 4$, $\texttt{min\_dur}[2] = 2$, $\texttt{max\_dur}[2] = 8$, etc. Similarly, the last segment of the last character must be segment K, last segment of the previous character can be any of the segments $K-4$, $K-3$, $K-2$, or $K-1$, and so forth. Hence, $\texttt{min\_dur}[n] = \texttt{max\_dur}[n] = K$, $\texttt{min\_dur}[n-1] = K-4$, $\texttt{max\_dur}[n-1] = K-1$, etc. All this, assuming that the word is not too long or too short compared to the number of elementary segments. If the word is too short, many characters would have to contain more than one segment, if the word is too long, many characters would have to contain less than four segments.

The following are the formulas for arrays min_dur[] and max_dur[], given that K is in the range $\texttt{min\_K} \leq K \leq \texttt{max\_K}$ and $|w| = n$, with $\texttt{min\_K} \geq n$ and $\texttt{max\_K} \leq 4n$.

$$\texttt{min\_dur}[i] = \max\{i, \texttt{min\_K} - 4 * (n - i)\}$$

and

$$\texttt{max\_dur}[i] = \min\{4 * i, \texttt{max\_K} - (n - i)\}.$$

Step 3: Compute arrays "min_char[ ]" **and** "max_char[ ]." Given a word w with $|w| = n$, variables min_char[k] and max_char[k] for $k = 0, \ldots,$ max_K contain the ID of the first and the last character of word w that can end at segment k. In other words,

$$\texttt{min\_char}[k] = \min \{i \mid m(i) = k \text{ and } m \text{ varies over all possible matchings}\}$$

and

$$\texttt{max\_char}[k] = \max \{i \mid m(i) = k \text{ and } m \text{ varies over all possible matchings}\}.$$

Arrays min_char[ ] and max_char[ ] can be easily computed using the formulas

$$\texttt{min\_char}[k] = i \quad \text{for } \texttt{max\_dur}[i-1] < k \leq \texttt{max\_dur}[i],$$
$$\texttt{max\_char}[k] = i - 1 \quad \text{for } \texttt{min\_dur}[i-1] \leq k < \texttt{min\_dur}[i].$$

Notice that min_char[0] = max_char[0] = 0.

It should be evident now that the values of "fese[i][j][k]" are meaningful only when $0 \leq k \leq$ max_K,

$$\texttt{min\_char1}[k] \leq i \leq \texttt{max\_char1}[k],$$

and

$$\texttt{min\_char2}[k] \leq j \leq \texttt{max\_char2}[k],$$

the values of "fesc[i][j][k][e]," are meaningful only if $0 \leq k$, $k + e \leq$ max_K,

$$\texttt{min\_char1}[k] \leq i \leq \texttt{max\_char1}[k],$$

and

$$\texttt{min\_char2}[k+e] \leq j \leq \texttt{max\_char2}[k+e],$$

and similarly the values of "fcse[i][j][k][e]," are meaningful only if $0 \leq k$, $k + e \leq$ max_K,

$$\texttt{min\_char1}[k+e] \leq i \leq \texttt{max\_char1}[k+e],$$

and

$$\texttt{min\_char2}[k] \leq j \leq \texttt{max\_char2}[k].$$

It is now possible to compute the meaningful elements of matrices "fese[ ][ ][ ]", "fesc[ ][ ][ ][ ]," and "fcse[ ][ ][ ][ ]" for a given k applying the formulas above only on the meaningful elements for $k-1$, $k-2$, $k-3$, and $k-4$. In particular, there is no need to initialize any values in the matrices "fesc[ ][ ][ ][ ]" and "fcse[ ][ ][ ][ ]" since the first segments of the first characters are *always* aligned.

5.1.3 Remarks

The word recognizer used [3] imposes variable limits on the number of segments per character (Table 2). For example, "j" can have 1 or 2 segments, "w" can have between 1 and 4 segments, etc. Thus, the "real" formulas for max_K, min_dur[ ], max_char[k], etc., are slightly more complicated and different for each word. We have chosen not to consider this variability in order to simplify our exposition.

Our slice distance is designed to quantify how likely it is for two words to be confused by our recognizer—the larger the distance between two words, the less likely they can be confused. Thus, one expects to have a small distance between words like "i" and "e," and e and f (since they could be easily confused by our recognizer), while a large distance is expected between words i and f (since they do not get easily confused).

TABLE 2
The Maximum Number of Segments Possible for Different Characters Varies from 1 to 4 When Using the Segmenter of WR-1

Lexicon 1:	Lexicon 2:
1: He	1: Me
2: Me	2: Memo
3: No	3: Memory
4: So	4: Memorize
5: We	5: Memoryless

5.2 Grapheme Distance d_R for WR-2

Consider two words consisting of c_1 and c_2 characters, respectively. Suppose their models are $M_1 = < S_1, T_1, l_1, \pi_1, \sigma_1 >$ and $M_2 = < S_2, T_2, l_2, \pi_2, \sigma_2 >$. A dynamic programming table $d(x, y)$ is built for $x \in S_1$ and $y \in S_2$.

$d(x, y) = 0$ when $\{(x', x) \in T_1\} = \emptyset, \{(y', y) \in T_2\} = \emptyset$. Otherwise,

$$d(x,y) = min \begin{pmatrix} \{d(x',y) + s(l_1(x), \epsilon) | (x', x) \in T_1\} & \cup \\ \{d(x,y') + s(\epsilon, l_2(y)) | (y', y) \in T_2\} & \cup \\ \{d(x',y') + s(l_1(x), l_2(y)) | (x', x) \in T_1, (y', y) \in T_2\} & \end{pmatrix},$$

where $s(f, g)$ is the predefined distance function between feature f and feature g. $s(l_1(x), \epsilon)$ is equivalent to the deletion of $l_1(x)$ and $s(\epsilon, l_2(y))$ is equivalent to the deletion of $l_2(y)$.

The final distance between the two word models is defined as

$$d_R(M_1, M_2) = min(\{d(x,y) | x \in \sigma(M_1), y \in \sigma(M_2)\})/N_1,$$

which is normalized by the number of characters (N_1) in the first word and, thus, not symmetric.

6 Experiments

We have designed a simple yet very effective procedure to evaluate the dependence of the accuracy of a word recognizer [3] on lexicon density as computed in this paper. We used a set of 3,000 images from the CEDAR CDROM. This set contains images of words extracted from handwritten addresses on US mail and is used as a standard for evaluating word recognizers.

For each image, we randomly generated 10 lexicons of sizes 5, 10, 20, and 40. Each lexicon contains the truth (the correct answer). For any specific size, the lexicons are divided into 10 groups depending on their density—the most dense lexicons for each image were collected in the first group, the second most dense lexicons for each image were collected in the second group, and so forth. We have

TABLE 3
Performance of WR-1 and WR-2 on a Set of 3,000 Images with 40 Different Lexicons for Each Image

WR-1						WR-2					
Slice distance			Edit distance			Grapheme distance			Edit distance		
n	v_R	p_R %	n	v	p_R %	n	v_R	p_R %	n	v	p_R %
5	278.7	83.12	5	0.5658	83.41	5	104.1	65.59	5	0.5658	64.55
5	223.7	85.72	5	0.4882	86.21	5	94.6	67.52	5	0.4882	67.49
5	192.5	87.82	5	0.4371	87.68	5	87.4	68.59	5	0.4371	69.73
5	168.6	89.22	5	0.3958	89.15	5	80.2	73.77	5	0.3958	73.30
5	146.8	91.12	5	0.3505	91.32	5	72.3	75.50	5	0.3505	75.27
5	130.7	92.29	5	0.3186	92.26	5	64.3	78.07	5	0.3186	79.24
5	116.6	93.16	5	0.2919	94.26	5	56.2	81.74	5	0.2919	81.61
5	104.0	95.10	5	0.2686	94.43	5	47.3	84.81	5	0.2686	84.08
5	91.5	95.66	5	0.2459	95.39	5	36.7	85.35	5	0.2459	85.81
5	76.4	96.70	5	0.2184	96.09	5	19.6	89.19	5	0.2184	89.05
10	224.6	72.37	10	0.4679	73.03	10	94.2	45.23	10	0.4679	45.26
10	193.4	76.04	10	0.4241	77.20	10	87.0	47.30	10	0.4241	48.73
10	172.9	78.24	10	0.3898	77.50	10	81.1	51.70	10	0.3898	51.30
10	155.6	81.01	10	0.3604	80.31	10	75.2	55.34	10	0.3604	55.84
10	133.1	85.55	10	0.3149	84.08	10	65.7	61.28	10	0.3149	62.42
10	119.6	85.82	10	0.2920	86.82	10	58.4	66.79	10	0.2920	65.62
10	107.2	88.72	10	0.2699	88.85	10	50.0	69.49	10	0.2699	69.16
10	97.0	90.52	10	0.2526	90.95	10	42.0	73.87	10	0.2526	73.26
10	86.6	92.13	10	0.2356	92.02	10	31.9	76.03	10	0.2356	76.03
10	75.7	93.49	10	0.2151	93.42	10	18.4	78.81	10	0.2151	78.20
20	199.9	60.03	20	0.4201	61.68	20	88.6	30.44	20	0.4201	30.14
20	181.2	62.60	20	0.3933	62.88	20	83.4	32.51	20	0.3933	32.84
20	165.6	66.87	20	0.3660	66.09	20	78.6	35.35	20	0.3660	36.15
20	151.9	68.70	20	0.3457	68.26	20	73.8	38.02	20	0.3457	38.05
20	126.8	74.81	20	0.2958	75.30	20	62.4	48.33	20	0.2958	48.40
20	115.8	78.14	20	0.2786	78.10	20	56.1	51.27	20	0.2786	51.00
20	103.0	82.98	20	0.2571	82.84	20	46.4	56.61	20	0.2571	58.74
20	94.7	85.22	20	0.2438	84.78	20	39.4	61.78	20	0.2438	60.75
20	85.3	86.59	20	0.2295	86.38	20	29.0	65.55	20	0.2295	64.89
20	77.1	88.29	20	0.2135	88.15	20	18.9	68.96	20	0.2135	67.86
40	190.2	47.58	40	0.3947	47.86	40	85.6	18.06	40	0.3947	17.96
40	176.3	49.42	40	0.3773	50.33	40	81.8	19.43	40	0.3773	20.19
40	162.2	53.89	40	0.3511	52.34	40	77.3	21.03	40	0.3511	20.79
40	151.7	53.69	40	0.3365	54.41	40	73.4	22.33	40	0.3365	22.10
40	123.8	64.83	40	0.2842	65.05	40	60.6	33.04	40	0.2842	33.78
40	115.4	67.27	40	0.2718	67.32	40	55.4	36.25	40	0.2718	35.85
40	101.5	74.61	40	0.2486	74.50	40	44.3	43.69	40	0.2486	43.46
40	94.7	76.14	40	0.2385	76.97	40	38.8	47.16	40	0.2385	48.16
40	84.9	80.38	40	0.2241	79.57	40	27.4	52.84	40	0.2241	51.90
40	78.9	82.32	40	0.2123	81.98	40	20.2	55.14	40	0.2123	54.77

Three different groups of lexicons were considered: 1) "edit distance" based, 2) "slice distance" based, and 3) "grapheme distance" based.

tested the performance of the word recognizer on each of these groups. Notice that such ordering of lexicons depends on the definition of lexicon density $\rho_R(L)$ and the same ordering will be obtained using $v_R(L)$.

A natural alternative of lexicon density is a measure based on the string edit distance—the minimum number of insertions, deletions and substitutions needed to change one string to another. It is defined as

$$\rho_R(L) = v(L)(f_R(n) + \delta_R),$$

where

$$v(L) = \frac{n(n-1)}{\sum_{i \neq j} d(w_i, w_j)}$$

is the reciprocal of average edit distance between word pairs.

Table 3 shows the performance of WR-1 and WR-2 on 40 different groups of lexicons, together with reciprocals of average distances. The performance numbers are first choice correct rates in percentage. Multiple regression is performed

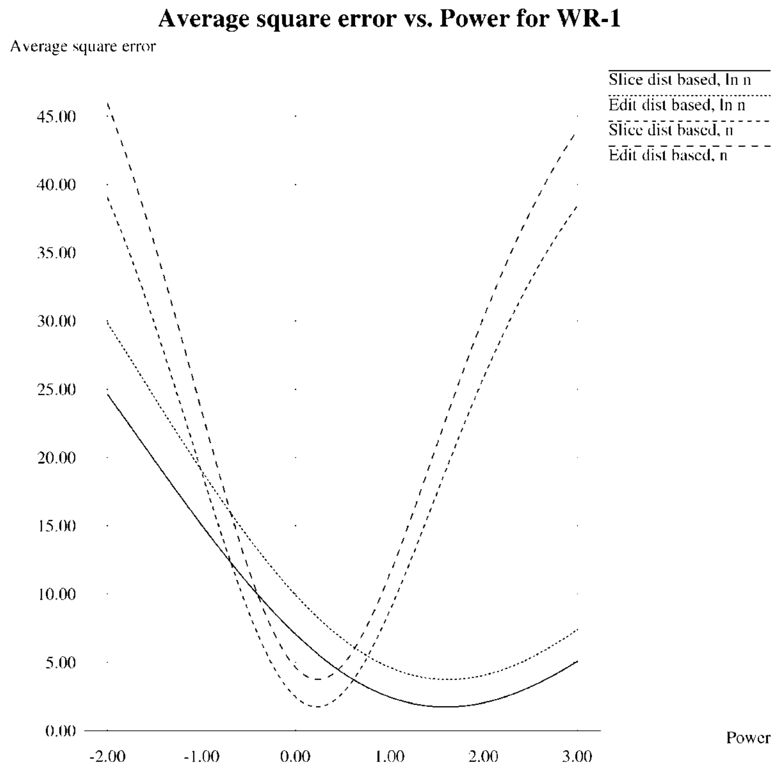


Fig. 5. The effect of $f_R(n)$ on the average square error for WR-1. Two sets of functions, $\{f_R(n) = ln^p n|p\}$ and $\{f_R(n) = n^p|p\}$ are examined for "slice distance" based and "edit distance" based lexicon densities.

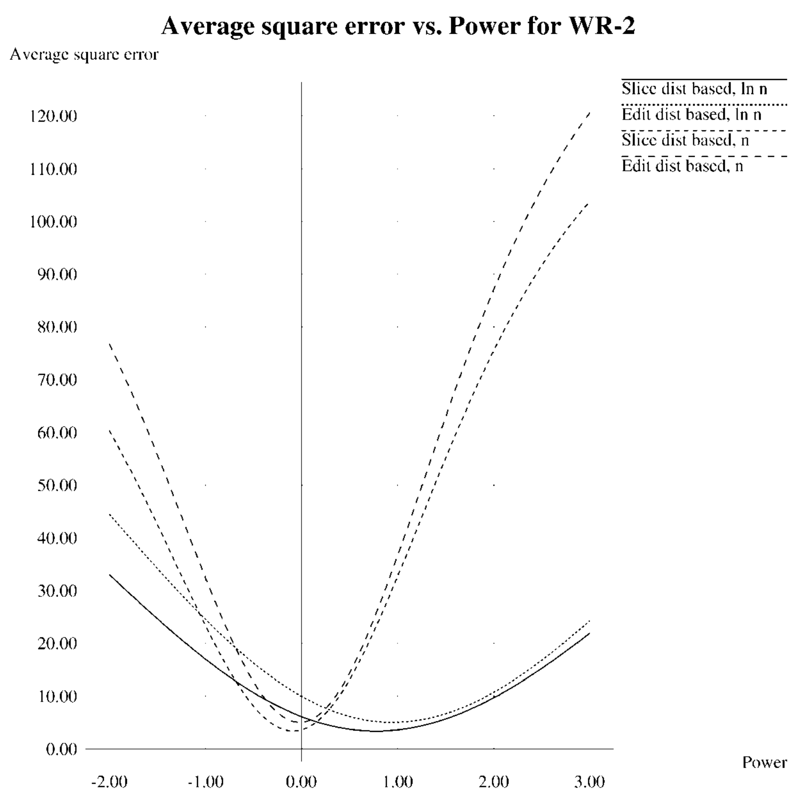


Fig. 6. The effect of $f_R(n)$ on the average square error for WR-2. Two sets of functions, $\{f_R(n) = ln^p n|p\}$ and $\{f_R(n) = n^p|p\}$ are examined for "grapheme distance" based and "edit distance" based lexicon densities.

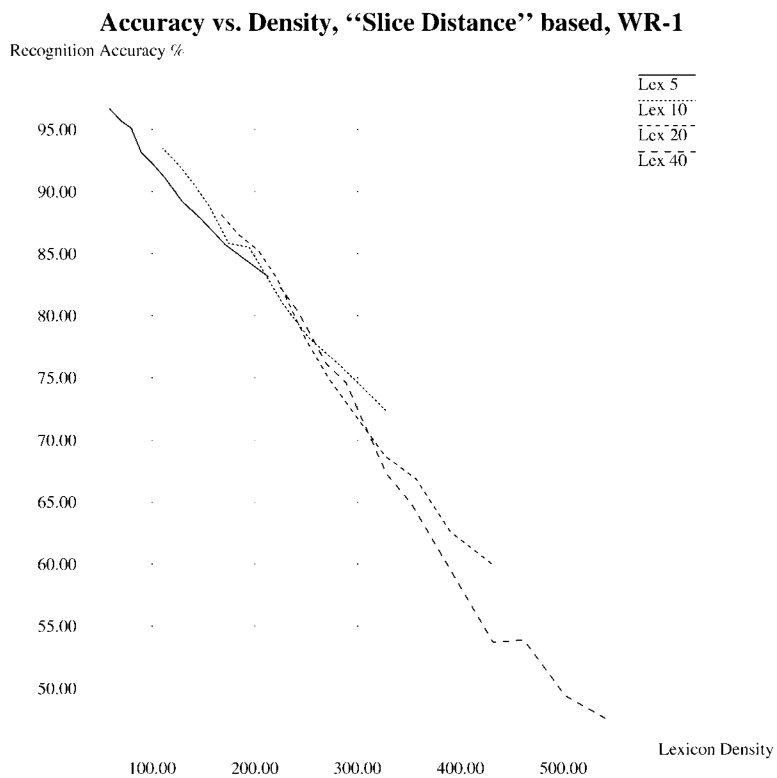


Fig. 7. Dependence of the performance of WR-1 on "slice distance" based lexicon density when $f_R(n) = \ln n$ and $\delta_R = -0.8432$. The average square error is 2.4693 (1).

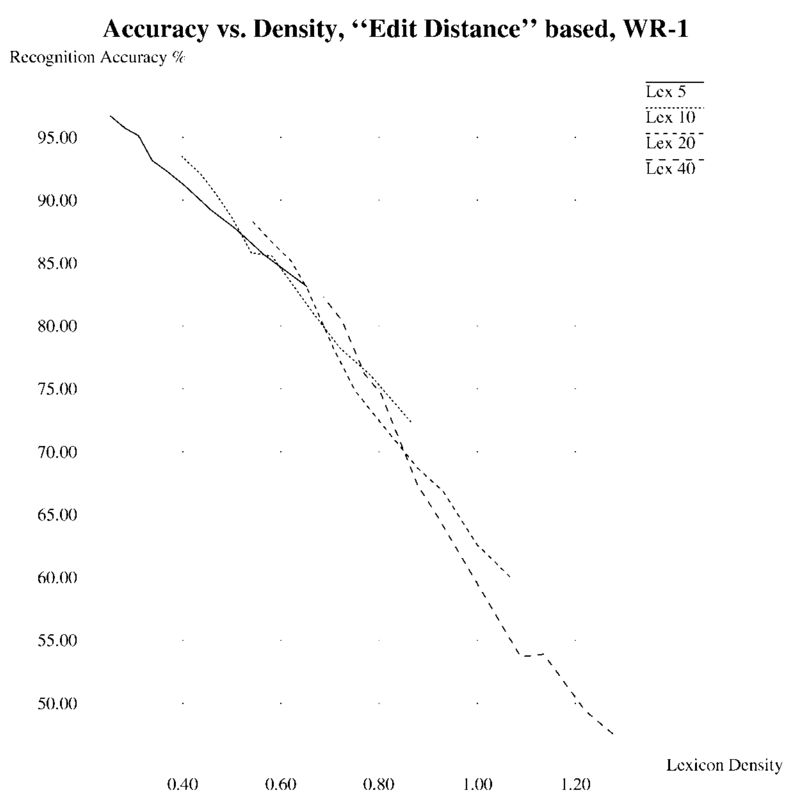


Fig. 8. Dependence of the performance of WR-1 on "edit distance" based lexicon density when $f_R(n) = \ln n$ and $\delta_R = -0.4546$. The average square error is 4.6456 (1).

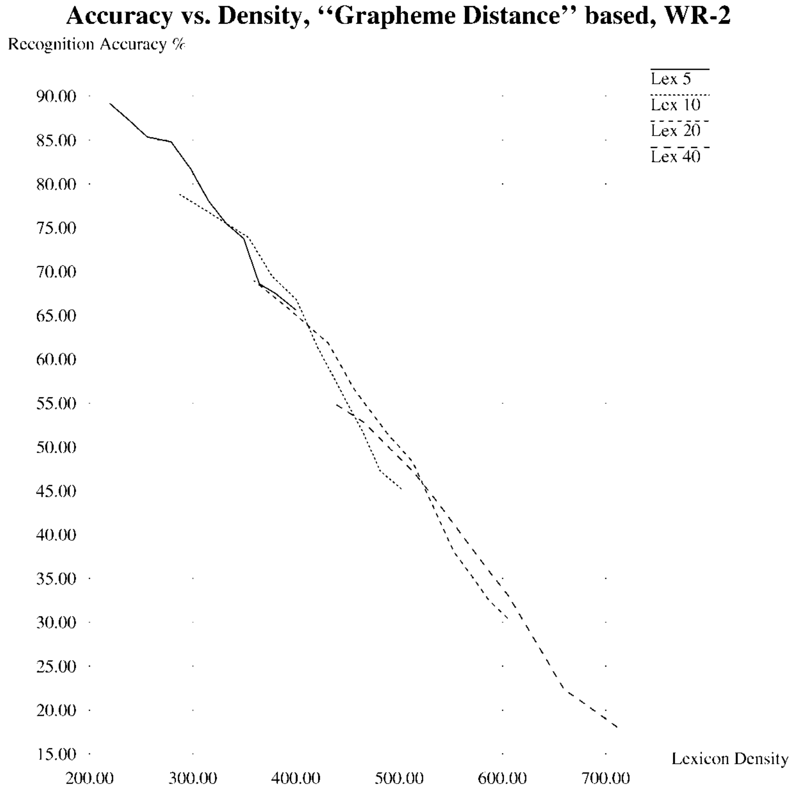


Fig. 9. Dependence of the performance of WR-2 on "grapheme distance" based lexicon density when $f_R(n) = \ln n$ and $\delta_R = 0.5298$. The average square error is 3.6086 (1).

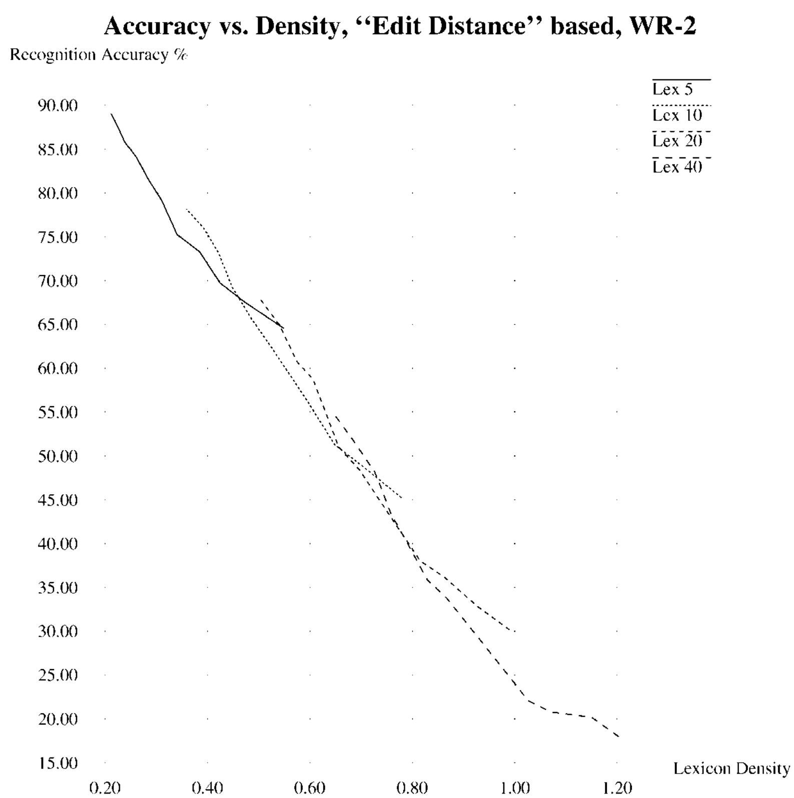


Fig. 10. Dependence of the performance of WR-2 on "edit distance" based lexicon density when $f_R(n) = \ln n$ and $\delta_R = -0.6388$. The average square error is 5.0602 (1).

on this data set to discover the approximate linear dependence between recognition performance and lexicon density.

In order to find evidence of preferring one $f_R(n)$ over another, we consider two sets of functions, $\{f_R(n) = ln^p n|p\}$ and $\{f_R(n) = n^p|p\}$. Figs. 5 and 6 show the average square error ($E_R/|\Gamma|$ after multiple regression computed from (1)) versus the power p, for WR-1 and WR-2, respectively. As illustrated, the minimum error occurs around $p = 1$ for the $\ln^p n$ set and $p = 0$ for the n^p set. However, $p = 0$ implies $f_R(n) \equiv 1$ and the problem degrades to linear regression that consequently yields much larger error. (It can be easily seen in Table 3 that there is no strong linear relation between recognition accuracy and any reciprocal of average distance.) The variation of error for the $\ln^p n$ set is also less sharp than that for n^p, which allows more accurate estimation of performance when there is a small error in choosing the best p. Based on the above analysis, we choose $f_R(n) = \ln n$.

Figs. 7 and 8 show the best linear dependence of recognition accuracy on lexicon density when $f_R(n) = \ln n$ for WR-1, with the corresponding (best) δ_R given. The results here combined with Fig. 5 also show that the recognizer dependent definition of lexicon density is generally more accurate than the recognizer independent one such as that based on string edit distance. Figs. 9 and 10 combined with Fig. 6 show similar results for WR-2.

The results seem to be conforming to the intuitive notion of lexicon density we set out to define. Recognition accuracy decreases with increasing lexicon density and if the density is the same, although the lexicon sizes may be different, the recognition accuracy stays about the same.

7 Summary

In this paper, we present a new measure, LD to evaluate the difficulty of a given lexicon with respect to a given recognizer. Lexicon Density (LD) depends both on the entries in the lexicon and on a given recognizer. Intuitively, the higher the lexicon density the more difficult it is for the recognizer to select the correct lexicon entry.

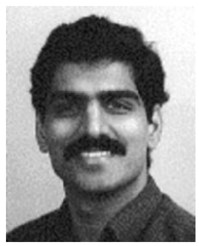
We have described how to compute the slice distance between two ASCII words for a segmentation based recognizer. Recognizers sometimes use probability measures instead of distances. For such recognizers, our algorithm could easily be modified to output the probability of confusing two words. It can be obtained by multiplying the elementary probabilities of confusing character slices (instead of adding the elementary distances) and then maximizing (instead of minimizing) the total probability over all possible slice-matchings.

Acknowledgments

The authors would like to thank their colleagues at CEDAR, in particular Evie, Jaehwa, and Krasi, for numerous discussions and feedback. We also got some very useful suggestions from the anonymous reviewers.

References

[1] L.R. Bahl, F. Jelinek, and R.L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 5, no. 2, Mar. 1983.
[2] R. Hamming, *Coding and Information Theory.* Prentice Hall, 1982.
[3] G. Kim and V. Govindaraju, "A Lexicon Driven Approach to Handwritten Word Recognition for Real-Time Applications," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 4, pp. 366-379, Apr. 1997.
[4] V.I. Levenshtein, "Binary Codes Capable of Correcting Insertions, Deletions, and Reversals," *Cybernetics and Control Theory,* vol. 10, no. 8, pp. 707-710, 1966.
[5] G. Seni, V. Kripasundar, and R.K. Srihari, "Generalizing Edit Distance to Incorporate Domain Information: Handwritten Text Recognition as a Case Study," *Pattern Recognition,* vol. 29, no. 3, pp. 405-414, 1996.
[6] G. Stepehen, *String Searching Algorithms.* World Scientific, 2000.
[7] R.A. Wagner and M.J. Fischer, "The String-to-String Correction Problem," *J. ACM,* vol. 21, no. 1, pp. 168-173, Jan. 1974.
[8] H. Xue and V. Govindaraju, "Handwritten Word Recognition Based on Structural Features and Stochastic Models," *Proc. Int'l Conf. Document Analysis and Recognition,* 2001.



Venu Govindaraju received the PhD degree in computer science from the State University of New York at Buffalo (SUNY) in 1992. He has coauthored more than 120 technical papers in various international journals and conferences and has one US patent. He is currently the associate director of Center for Excellence in Document Analysis and Recognition (CEDAR) and concurrently holds the associate professorship in the Department of Computer Science and Engineering, SUNY. He won the ICDAR Outstanding Young Investigator Award in September 2001. He is the program cochair of the upcoming International Workshop on Frontiers in Handwriting Recoignition in 2002. He is a senior member of the IEEE.

Petr Slavík received the MS degree in computer science and the PhD degree in mathematics from the State University of New York at Buffalo (SUNY) in 1998. From 1998 to 2000, he was a research scientist at Center for Excellence in Document Analysis and Recognition (CEDAR), SUNY Buffalo, where he worked on offline handwriting recognition. He is currently a memeber of the online handwriting recognition team at Microsoft Corporation. His resarch interests include handwriting recognition, theory of algorithms, and combinatorial optimization.

Hanhong Xue received the BS and MS degrees in computer science from University of Science and Technology of China, in 1995 and 1998, respectively. He is currently pursuing the PhD degree in computer science and engineering at the State University of New York at Buffalo. His research interests, include image processing, pattern recognition and computer vision. He is a student member of the IEEE.