A nonparametric classifier for unsegmented text

George Nagy*[a], Ashutosh Joshi[a], Mukkai Krishnamoorthy[a], Yu Lin[c],
Daniel Lopresti[b], Shashank Mehta[d], Sharad Seth[c],
[a]Rensselaer Polytechnic Institute, Troy, NY USA 12180-3590;
[b]Lehigh University, Bethlehem, PA, USA 18015-3084;
[c]University of Nebraska, Lincoln, NE, USA 68588-0115;
[d]Indian Institute of Technology, Kanpur, India 208 016

ABSTRACT

Symbolic Indirect Correlation (*SIC*) is a new classification method for unsegmented patterns. SIC requires two levels of comparisons. First, the feature sequences from an unknown query signal and a known multi-pattern reference signal are matched. Then, the *order* of the matched features is compared with the order of matches between every lexicon symbol-string and the reference string in the lexical domain. The query is classified according to the best matching lexicon string in the second comparison. Accuracy increases as classified feature-and-symbol strings are added to the reference string.

Keywords: OCR, on-line handwriting retrieval, nonparametric classifier, symbolic indirect correlation.

I. INTRODUCTION

Nearest-neighbor (and k-NN) classifiers require no training, are simple to build, and have reasonable run-time characteristics after appropriate preprocessing of the reference data. They are often used for accuracy benchmarks when "large-enough" labeled reference sets are available. However, equivalent non-parametric classifiers have not been available for sequential signals like on-line handwriting and speech. *Symbolic Indirect Correlation* (SIC), the subject of this paper, is a new general method for exploiting the ordered correspondences between lexical transcripts of signals of arbitrary length and their feature representation. We call it *indirect* because it is based on a comparison of comparisons, and *symbolic* because it makes use of the ordered lexical (letter) polygrams. *Correlation* is meant to suggest sliding-window type comparisons.

SIC is applicable wherever lexically labeled reference signals can be compared to unlabeled signals. It avoids the usual integrated segmentation-by-recognition loop. In contrast to whole-word recognition, it does not require feature-level samples of the words to be recognized. Unlike the prevalent Hidden Markov Methods, it needs no estimates of an enormous number of classifier parameters by means of a fragile initial bootstrap.

SIC is a natural evolution of an existing classifier (*InkLink*) for electronic ink (on-line cursive writing). InkLink can recognize a word that it has never been seen before in ink if its transcript appears in its lexicon. In some sense, both of these classifiers bear the same relation to Hidden Markov Methods as k-Nearest Neighbors to the Gaussian classifier. Both compare the feature representation of an unknown (*query*) word with the feature representation of a string of labeled reference words. The comparisons are based on polygram-length segments because in handwriting many letters change shape depending on the preceding and following letter (i.e., exh ibit *allographs*). We describe the earlier classifier briefly, and then present a significant generalization that is the subject of our current research (preliminary results were first reported at a workshop in June 2003[1]). SIC is especially suitable for information retrieval from analog text representations because a change in retrieval target requires only substitution of new target words into the lexicon, replacing any previous target words, without any change in the reference string, or any retraining or re-estimation.

*nagy@ecse.rpi.edu; phone 1 518 276-7078; fax 1 518 276-6261

II. INK-LINK

For simplicity, we describe InkLink with a lexicon of only two words, a reference string of three words, and a set of four features. Consider the string of reference words *~period~ever~people~* and the lexicon *lever* and *perplex*. The two query words, *lever* and *perplex*, are represented by a string of local orientation features (1,2,3,4), with 3-5 features per letter. In the ideal case, the reference string might be:

3421213123332412124321~21341314213123~342121334211122213

and the unknown words have features

lever: 112221341314213123
perplex: 342121312334211222131334

The reference string has been segmented at the word level, and we can count the number of features in each word. Therefore given enough reference words we can estimate the number of features in each letter by least squares estimation[7]. In a lexical preprocessing stage on the transcripts of the lexicon and of the reference words we find the location of all letter polygram matches. We note, for example, that the fourth and fifth letters of *lever* match the second and third letter of *period*. The expected matches between lexicon words and reference words are underlined below.

lever		perplex	
lev<u>er</u>	*p<u>er</u>iod*	*<u>per</u>plex*	*p<u>er</u>iod*
l<u>ever</u>	*<u>ever</u>*	*p<u>er</u>plex*	*ev<u>er</u>*
<u>le</u>ver	*peop<u>le</u>*	*<u>perple</u>x*	*<u>peo</u>ple*

The location of these lexical matches allows us to predict the location and length of the feature level matches. For example, if the unknown word is *lever,* then the string of 6 features at its end should match a feature string near the beginning of *period* in the reference feature string, as shown:

lev<u>er</u>: 112221341314<u>213123</u>
p<u>er</u>iod: 3421<u>213123</u>332412124321

Similarly, we can check for all the other expected matches. We use a Viterbi algorithm constrained by the expected location of the matches[8]. Of course, with text recorded on line with a tablet and stylus, the matches won't be exact. We therefore estimate the distribution of the length of the observed feature matches conditioned on the predicted length. Finally, we use a Bayesian estimate of the probability of the ensemble of observed matches against the entire reference feature string to determine which lexical hypothesis provides the most probable match at the predicted locations[9]. The query is then assigned the label of the most probable lexicon word.

InkLink has been applied to lexicons of up to 1000 words with reference strings of a few hundred words, and a set of two dozen local directional maxima, cusp and crossing features[2,3,4,5]. On a typical (sloppy) writer, InkLink's average recognition accuracy on four data sets of 100 test words, with a 100-word randomly selected reference set, was 77%. Adaptation increased the recognition rate to 94%, which is almost the same as that achieved with a 500-word reference set without adaptation. On datasets of eleven entirely new writers, provided by the Pen Technology Group of IBM's Watson Research Center, accuracy on multiple 100-word test sets and lexicons and 500 reference words ranged from 63% to 98.7%. On the four writers with characteristics similar to our development writer (no slant, normal writing speed), the error rate was comparable to those obtained by IBM on the same test sets (>98%)[6].

III. SYMBOLIC INDIRECT CORRELATION

As shown earlier, the first polygram match in *lever*, *le*, is the third match in the reference string. The second match, *ever*, is also the second match in the reference string. And the last match, *er*, is the first match in the reference string. Therefore the order of the matches can be described by the permutation (*3,2,1*). Similarly, *perplex* is characterized by the permutation (*1,3,2,4*). Therefore if feature extraction is perfect, it is sufficient to determine which lexical candidate induces the same permutation as the query.

For imperfect matches, the comparison is much more difficult. Let us call the query-reference permutation Q, and the permutations of the lexical words against the transcript of the reference string P1, P2, P3,… . The classification is then performed by finding the longest common subpermutation between Q and P1, Q and P2, Q and P3, etc. Finding the longest subpermutation S* can be shown to be an exponential computation in the length of the longer of the two strings[11].

Another way of looking at the problem is that of finding the largest common subgraph of two bipartite graphs. In the above example, the edges of the graph for *lever* are (1,17), (2,8), (4,2), and for *perplex* (1,1), (1,13), (2,10), 5,17), where the first index is the first letter of the matching polygram in the lexicon word, and the second in the transcript of the reference string.

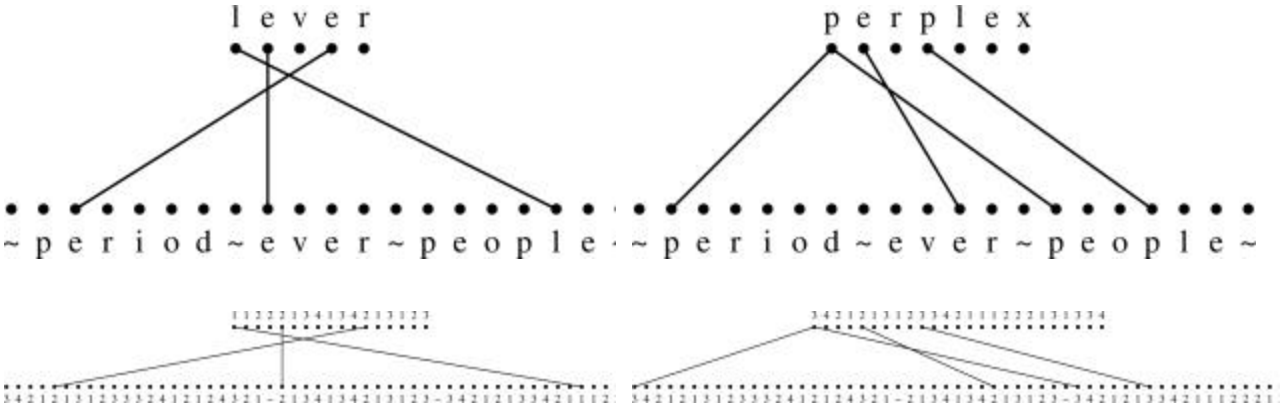

Fig. 1. Lexical and feature match graphs against the reference string.

The two bipartite match graphs represent the lexical and signal relationship, respectively, between each lexical candidate word and the reference string. One approach we are beginning to explore to extract these match graphs involves variations on the well-known edit distance model of sequence comparison[10]. This technique is mathematically rigorous and allows a wide range of possibilities for representing the similarities between two sequences of interest. In a single, efficient computation, we can locate all subsequence matches above a desired threshold. Through a proper choice of cost functions, the same algorithm can be applied to our lexical graphs and our feature graphs. The four graphs for our example are shown in Figure 1: it is seen that the connectivity of the lexical graphs is identical to that of the feature graphs.

To find the largest common subgraph or subpermutation we have implemented a Branch-and-Bound (B&B) algorithm that is much more efficient than brute force computation. Nevertheless, it is also necessarily worst-case exponential. An upper bound on the size of the search space is ${}^{n+m}C_m$, where m and n are the lengths of the two strings. However, we can reasonably expect that the lexical word P* that corresponds to the query will share a subpermutation S with the query-reference permutation Q that is almost as long as min_length[P*,Q]. Even the average-case computation is exponential, but only in (min_length[P*,Q] – S). We are also exploring a number of heuristic approximations that are much faster than the above B&B algorithm.

In the next section, we present a formal statement of our approach to sequence recognition.

IV. FORMAL PROBLEM STATEMENT

Assumptions:

Input signals can be captured and represented as linear sequences of features over a predetermined alphabet:

$$V = v_1 v_2 ... v_m \quad \ni \quad v_i \in \Sigma_v$$

Every feature sequence has a unique lexical transcription over another alphabet:

$$V = v_1 v_2 ... v_m \quad \Rightarrow \quad A = a_1 a_2 ... a_n \quad \ni \quad a_i \in \Sigma_a$$

Given an unknown input feature sequence V^*, the goal is to determine the correct lexical transcription A^*.

Further Assumptions:

There exists a comparison measure C_v that quantifies the similarity between any two feature subsequences $u_i...u_j$ and $v_k...v_l$:

$$C_v (u_i...u_j, v_k...v_l) \quad \rightarrow \quad \Re$$

Given two input feature sequences U and V, let a match set M_v be the set of all subsequences of U and V such that the comparison measure surpasses some threshold τ_v:

$$M_v(U,V) = \{(u_i...u_j, v_k...v_l) \quad \ni \quad u_i...u_j \in U, v_k...v_l \in V, \text{ and } C_v (u_i...u_j, v_k...v_l) \quad \tau_v$$

(Here $\in$ represents the subsequence relationship. Note that recording just the starting indices of the members of M_v, yields the familiar match graph representation. This formulation is a less general than the one given above.)

Likewise, there exists a comparison measure C_a that quantifies the similarity between any two lexical subsequences:

$$C_a(a_i...a_j, b_k...b_l) \quad \rightarrow \quad \Re$$

Given two input lexical sequences A and B, let a match set M_a be the set of all subsequences of A and B such that the comparison measure surpasses some threshold τ_a:

$$M_a(A,B) = \{(a_i...a_j, b_k...b_l) \quad \ni \quad a_i...a_j \in A, b_k...b_l \in B, \text{ and } C_a (a_i...a_j, b_k...b_l) \quad \tau_a$$

Finally, there exists a comparison measure C_m that quantifies the similarity between any two match sets M_v and M_a:

$$C_m (M_v(U,V), M_a(A,B)) \quad \rightarrow \quad \Re$$

Problem Statement for a Single Reference String:

Given a reference feature string along with its associated transcription: $V_1 \Rightarrow A_1$ and a lexicon L of candidate transcriptions for unknown inputs: $L = \{L_1, L_2, ... L_i, \ldots, L_k\}$ for a specific unknown input feature sequence V^*, determine the correct lexical transcription A^*.

Assertion:

$$A^* = \underset{i}{\text{argmax}} \{C_m (M_v(V_1, V^*), M_a(A_1, L_i))\} \quad \ni \quad L_i \in L\}$$

Problem Statement for Multiple Reference Strings:

Given a collection of (one or more) feature strings along with their associated transcriptions:

$$V_1 \Rightarrow A_1$$
$$V_2 \Rightarrow A_2$$
$$\ldots$$
$$V_n \Rightarrow A_n$$

and a lexicon L of candidate transcriptions for unknown inputs:

$$L = \{L_1, L_2, ... L_i, \ldots L_k\}$$

for a specific unknown input feature sequence V^*, determine the correct lexical transcription A^*.

Assertion:

$$A^* = \underset{i}{\text{argmax}} \{h(C_m (M_v(V_j, V^*), M_a(A_j, L_i)))\} \quad \ni \quad 1 \le j \le n, L_i \in L\}$$

where h is some function for combining comparison values.

V. EXPERIMENTS

Figure 2 shows that the expected time increases rapidly (note the logarithmic time scale) with the dissimilarity D (edit distance) of the graphs being compared. With the B&B algorithm, we cannot hope to determine the exact size of the largest common subgraph unless the two subgraphs are quite similar. Fortunately, other experiments (not reported here) show that our match graphs exhibit far more similarity than random graphs.

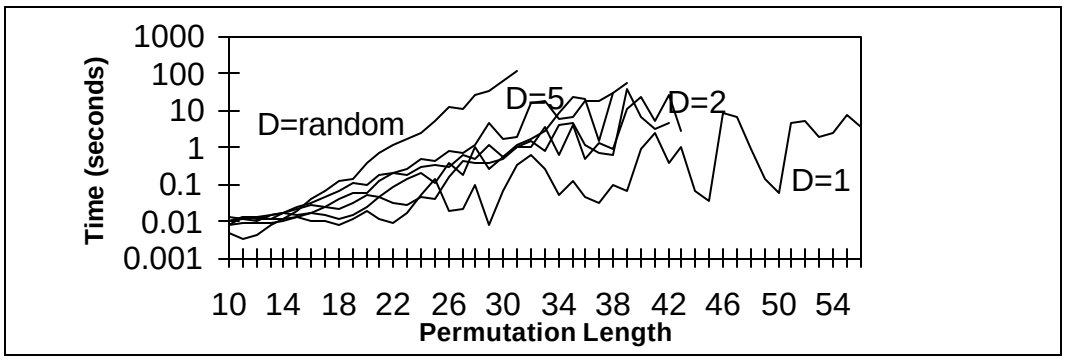

Figure 2. Comparison of B&B algorithm run-time for pairs of random graphs versus pairs of similar graphs (10 samples per value). The number of edits (edge swaps) indicate the degree of dissimilarity.

Experimentation with a heuristic algorithm on simulated features with Q% i.i.d. noise, with a lexicon comprising the most frequent words of the Brown Corpus[12], and a reference set of 1000 randomly selected words from the Brown Corpus, yielded the following results:

Q (% noise)	10	20	30	40
% recognition rate	99.2	99.1	98.3	96.0

Q% noise means that the locations of Q% of the edges are altered by random addition and deletion of edges in such a way that the total number of edges remains approximately constant. We were pleasantly surprised by the recognition rates obtained with heavy noise. We expect the real feature noise to be less, but far from independently distributed.

The average number of polygram matches between two words in the English language is about 0.3, so with a 1000-word reference string we expect 300 edges for a typical lexicon word. The observed distribution appears to be approximately Gaussian, with a standard deviation of about 60. Our first implementation of the B&B algorithm required excessive time for such large graphs. We are currently implementing a more efficient version.

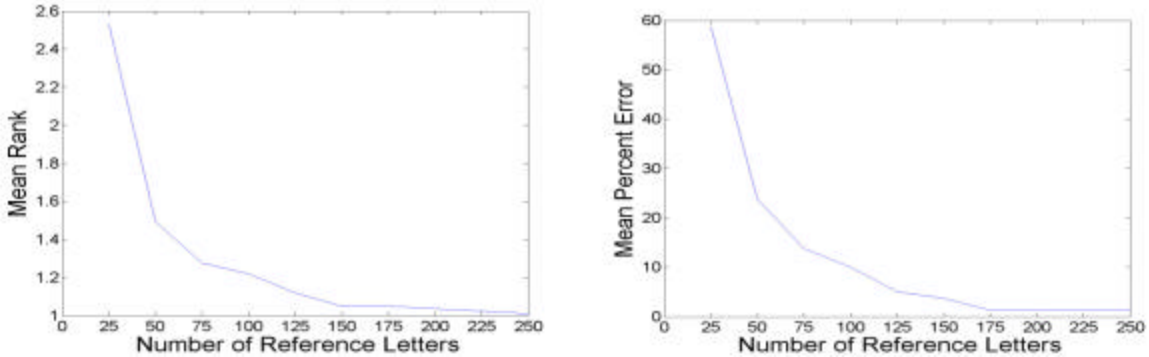

Figure 3. The mean rank of the correct five-letter string, and the error rate decrease with the length of the reference string.

We tried SIC on scanned printed characters, with a lexicon of ten five-letter strings and reference strings of up to 250 letters. Both the lexicon strings and the reference strings were chosen randomly. The alphabet consisted of only 4 letters

(a, e, c, r), with the signal representation of each letter chosen at random from 10 different scanned version of the character. The features were simply groups of three consecutive vertical columns of the character bitmaps. The feature level comparison operated on feature strings corresponding to about two-letters (i.e., bigrams). Matches between the unknown feature string and the feature string of the reference word were registered whenever the Hamming distance between 5 or more consecutive triplets of columns exceeded a threshold. The threshold was set to equalize the average length of the feature graphs with the average length of the lexical graphs. Figure 3 shows the results from 30 such experiments. It confirms our expectation that the error decreases with the reference string, albeit only on a toy problem.

We are currently conducting experiments on cursive on-line handwriting from the same database as the InkLink experiments reported earlier. On the theoretical side, we are investigating the following questions:

What is the right model (cost function and algorithm) for building the match graphs? So far we have been experimenting only with simple string-matching algorithms that allow skipping over superfluous, missing, or mismatching features in either string. The lexical matching need not, of course, make provisions for errors.

What are the computational characteristics of the branch and bound algorithm? Is it efficient enough to be practical? What kind of heuristics can we introduce that won't hurt the quality of the results? Under some simplifying conditions we have found a polynomial time transformation from the Clique problem. Such a transformation may allow using the wealth of approximate clique algorithms for an approximate solution to the SIC problem.

Does the error rate converge to the Bayes error as the reference list grows to infinity? Even for the simple "edge-flipping" noise model described above, so far we have been able to calculate the Bayes conditional error associated with each match graph (which is needed to calculate the overall Bayes error) only exhaustively, and therefore only for very short query and reference strings.

If we assume perfect matching at the character level, it is easy to find tight bounds on the length of the reference strings. Using a suitably chosen reference string that satisfies the conditions, we are able to distinguish all the words. What influence does the choice of words in the reference list have on the performance under the limitations imposed by imperfect polygram feature matching and natural language? Is there a good strategy for choosing the reference words under these conditions?

VI. CONCLUSION

We claim that our method offers the following advantages over current techniques.

1. In contrast to HMMs and the earlier polygram-based scheme, no parameter estimation is required. Therefore recognized words can be added immediately to the reference list, improving subsequent recognition.

2. The method is intrinsically more accurate than letter-based schemes, because polygram-length segments of scanned typescript and handwriting, and of electronic ink, are more distinctive than letter-length segments.

3. Symbolic Indirect Correlation is eminently suitable for information retrieval applications based on a finite set of keywords. In this case, the lexicon will contain only the keywords, and a reject criterion will be introduced to eliminate all other words.

4. In contrast with whole-word recognition, new words can be added to the lexicon without changing the reference list. Accuracy declines gracefully as the ratio of the size of the lexicon to that of the reference list increases. How quickly it declines is a topic of future research.

5. If the query consists of an unsegmented phrase rather than a single word, the subgraphs corresponding to each word remain localized in the query-reference match graph. Therefore SIC can be extended to phrase recognition (an important consideration in information retrieval), with only a linear increase in computation.

6. The feature set chosen must be appropriate to the script, but SIC does not depend on any characteristic of script and language other than the existence of an alphabetic transcript. It is applicable to other scripts and languages, and to speech signals with phonemic transcripts. Because it exploits feature sequence rather than position, SIC should be immune to the non-linear temporal stretching or contraction of speech signals.

REFERENCES

1. G. Nagy, S. Seth, S.K. Mehta, Y. Lin, Indirect symbolic correlation approach to unsegmented text recognition, *Proc. Workshop on Document Analysis and Information Retrieval* (CD-ROM only), Madison, WI, June 2003.

2. A. El Nasan, G. Nagy, "InkLink," *Procs. ICPR-XV, Vol. 2*, IEEE Computer Society Press, 573-576, Barcelona, 2000.

3. A. El Nasan, S. Veeramachaneni, G. Nagy, "Word discrimination based on bigram co-occurrences," *Procs. ICDAR-01,* IEEE Computer Society Press, pp. 149-153, Sept. 2001.

4. A. El Nasan, G. Nagy, "On-line handwriting recognition based on bigram co-occurrences," *Procs. ICPR XVI, Vol. III*, IEEE Computer Society Press, pp. 740-743, Aug. 2002.

5. A. El Nasan, G. Nagy, S. Veeramachaneni, "Handwriting recognition using position sensitive n-gram matching," *Procs. ICDAR-03,* IEEE Computer Society Press, Edinburgh, August 2003, pp. 577-582.

6. A. El Nasan, *InkLink: a writer-dependent, unconstrained on-line handwriting recognition system*, Rensselaer Polytechnic Institute PhD dissertation, August 2003.

7. Y. Xu, G. Nagy, "Prototype Extraction and Adaptive OCR," *IEEE Trans. PAMI-21*, 12, pp. 1280-1296, Dec. 1999.

8. A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm," *IEEE Trans. Information Theory* **IT-13**, pp. 260-269, 1967.

9. A. El Nasan, M. Perrone, "On-line handwriting recognition using character bigram match vectors," *Procs. IWFHR-8,* IEEE Computer Society Press, August 2002, pp. 67-71.

10. D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison,* Addison-Wesley, 1983.

11. P. Bose, J. F. Buss, and A. Lubiw, “Pattern Matching for Permutations”, *Proc. Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science 709,* Springer Verlag pp. 200-209, 1993.

12. W. N. Francis and H. Kucera, *Brown Corpus Manual*, available at: www.hit.uib.no/icame/brown/bcm.html, revised and amplified 1979.