COMPARISON OF TWO CONTEXTUAL POST-PROCESSING ALGORITHMS FOR TEXT RECOGNITION

Jonathan J. Hull and Sargur N. Srihari

Department of Computer Science
State University of New York at Buffalo
Amherst, New York 14226

ABSTRACT

The binary n-gram and Viterbi algorithms are alternative approaches to contextual post-processing for text produced by a noisy channel such as an optical character recognizer. The paper describes the underlying theory of each approach in unified terminology, presents a storage efficient data structure for the binary n-gram algorithm and a recursive formulation for the Viterbi algorithm. Relative merits of the two methods based on extensive experiments with each algorithm are given.

1. INTRODUCTION

A number of programs for utilizing some form of contextual knowledge for post-processing of garbled text produced by a noisy channel such as an Optical Character Recognition [OCR] machine are described in the literature. Among these one can discern two different types of representation of context: statistical and structural. This paper describes our experiences with two contextual post-processing algorithms which are based on each of the two different types of representation.

Statistical representation of context consists of models of word generation processes, e.g., Markovian model of text source, whose origins are in information theory. A particular algorithm of this class that we consider is the Viterbi algorithm [3] and its modifications [1,6].

The structural approach, which is closer to the artificial intelligence framework, is based on a deterministic representation of context; examples are implicit or explicit representations of a lexicon, syntax or semantics. The method of this class we consider is the binary n-gram algorithm [2,5] which represents the syntax of a lexicon in the form of a set of binary arrays.

2. BINARY n-GRAM ALGORITHM

The binary n-gram algorithm utilizes an abstraction of a lexicon containing all allowable input words. The method attempts to detect, as well as correct, words with errors. The representation used is a set of binary arrays (n-gram arrays) of 26^n elements each, where n is usually restricted to be 2, 3, or 4. The entry in an n-gram array is 1 if the letter combination corresponding to the location of the entry in the array occurs in some lexical word in positions that are associated with the array. A positional binary digram (n=2) for a sublexicon $\mathcal{D}_m$, consisting of words of length m, is defined as follows:

$B^m_{ij}(\alpha,\beta)=1$, if characters α,β occur in positions i,j of some word in $\mathcal{D}_m$,

$=0$, otherwise

where $1\leq i<j\leq m$. Each (i,j) specifies a binary digram array, B^m_{ij}, of 26^2 bits each. Thus there exist $\binom{m}{2}$ different digram arrays corresponding to $\mathcal{D}_m$, which we will denote as $\mathcal{B}(2,m)=\{B^m_{ij}\}$. The positional binary trigram $B^m_{ijk}(\alpha,\beta,\gamma)$ and the set of $\binom{m}{3}$ trigram arrays $\mathcal{B}(3,m)=\{B^m_{ijk}\}$ of 26^3 bits each are defined in a similar manner. An exact representation of $\mathcal{D}_m$ is given by $\mathcal{B}(m,m)$--which consists of a single array of 26^m bits. An approximation of the structure of $\mathcal{D}_m$ is given by $\mathcal{B}(n,m)$, $n<m$.

The positional binary n-grams refer to specific sets of n locations in each m-letter word. Non-positional (or position independent) binary n-grams refer to all sets of n contiguous positions in each word. A non-positional n-gram array is defined to have 27^n entries--by taking into account the word delimiter (blank) symbol as well as the 26 letters of the alphabet. Non-positional binary n-grams the represent a specific sublexicon $\mathcal{D}_m$ are word-length non-positional binary n-grams, which we refer to as Non-positional-1. Non-positional

82CH1761-6/82/0000/0146$00.75 © 1982 IEEE

Proceedings of the 1982 IEEE Conference on Pattern Recognition and Image Processing, Las Vegas, Nevada, June 1982.

binary n-grams that are derived from the complete dictionary, $\{\mathcal{D}_i\}$ are word-length independent, which we refer to as Non-positional-2.

Error Detection

The error detection procedure assumes the availability of B(n,m) for a given m and n. When digrams are used, the condition of legality of a word $\underline{X}=X_1X_2..X_m$ is stated as:

if $B_2^m(\underline{X})=\bigwedge_{i,j} B_{ij}^m(X_i,X_j)=1$, then $\underline{X}$ is considered legal,

$=0$, then $\underline{X}$ is considered illegal.

This condition does not guarantee that $\underline{X}$ is in $\mathcal{D}_m$, it merely says that each of the digrams occurs someplace in $\mathcal{D}_m$.

Fixing Error Location

The n-grams that rejected $\underline{X}$ (i.e., their entries are zero for the appropriate letters of $\underline{X}$) must contain the location(s) of the error(s) among their position indices. Thus a set I (of indices) is determined that is the intersection of the indices of the rejecting n-grams. Assuming a single substitution error, if I contains a single element then the position of the error has been uniquely determined. If I contains more than one element, then the location of the error is given by one of the elements of I. If I is empty, there exists more than one substitution error--since no single position index occurs in all rejecting n-grams.

Error Correction

The procedure for correcting words where the position of a single substitution error has been fixed is based on the following principle. An n-gram array is an n-dimensional matrix where each dimension has 26 values. The one-values that lie along a 26-element vector of this matrix specify valid n-grams that differ from each other by a single letter; the remaining n-1 letters of the n-gram are specified by the n-1 dimensions needed to specify the vector. Thus the lexical n-grams that differ from the n-grams of $\underline{X}$ by one letter are easily extractable from the n-gram arrays.

The vector interpretation extends itself to the case when $\mathcal{D}_m$ is approximated by B(n,m), n<m. If it is possible to fix the position of a single substitution error then each n-gram that has this position as one of its position indices will contain information about allowable substitutions for that position. From each array of B(n,m) satisfying this condition a vector is extracted with coordinates determined by the n-1. characters in the appropriate positions of $\underline{X}$ that are assumed to be correct.

When these vectors have been determined, the information about substitutions in the position of the word about which these vectors were allowed to vary within each array is obtained by constructing a vector V that is the intersection of each of these vectors. A non-zero entry in this vector will correspond to a letter that can be substituted in the position that was hypothesized to contain an error and produce a word that will be acceptable to all n-gram arrays.

The Algorithm

The binary n-gram procedure for correcting single substitution errors is:

```
repeat
 get-word (X);
 if detect (X)
    then if card (index-set)=1
            then correct (X)
            else reject (X)
 word-out (X)
until end-of-file;
```

The procedure determines if a word contains an error and if the position of the error is fixed, then it is corrected, otherwise the word is rejected; where *get-word* reads a word from a text file and *card* returns the number of elements in its argument set. If the word does not contain any detectable errors then it is simply output by *word-out*.

Function *detect* returns true if any array of B(n,m) rejects $\underline{X}$, and also constructs index-set to contain the intersection of all sets of position indices of the rejecting n-grams. Given that index-set contains the position p of a single substitution error in word $\underline{X}$, procedure *correct* considers those arrays of B(n,m) that involve p and constructs the intersection vector V of the appropriate vectors from these arrays (the n-1 dimensions of each vector are determined by the assumption that the rest of the word is correct). If there is only one non-zero element in V then that letter is substituted in the ith position, otherwise the word is rejected. The reader is referred to [4] for a complete statement of this algorithm.

Storage Considerations

An error correction method based on positional binary n-grams calls for partitioning the dictionary by word length m and representing the subdictionary $\mathcal{D}_m$ by $\binom{m}{n}$ arrays of 26^n bits each. The storage requirements--particularly for trigrams--are usually excessive, thus methods of intelligent storage reduction are needed. One can reduce storage either by reducing the amount of data or by using an efficient data structure. The two approaches are

typified by the use of non-positional n-grams and a new data structure called linear marginal indexing, respectively.

Marginal indexing is a tree data structure for representing multi-dimensional arrays, where the maximal depth of the tree is equal to the dimensionality of the array, each level of the tree corresponds to one dimension of the array and each node is either a leaf or has a number of descendents equal to the subdivision along the corresponding dimension. In the representation of 3-d arrays, sons of the root denote the 2-d slices of the array, nodes at the next level specify the strips within a slice and nodes at the final level correspond to data values within a strip. An element of a node is either a data value or a pointer to a node at the next level. A pointer is used only if array values in the corresponding dimension are inconsistent and require the further description provided by a node at a lower level.

The method of marginal indexing can be improved upon for representing binary n-gram arrays. Consider the representation of a binary trigram array of 26^3 bits. Such an array usually does not contain any strips (and consequently slices) of all 1's, due to the fact that many letter combinations never occur in English words. This fact provides the motivation for the linearization of the marginal indexing tree and the use of binary valued elements that represent either pointers or strips and slices of all 0's. Further details of the method are given in [4].

3. VITERBI ALGORITHM

The Viterbi algorithm is a method of computing the most probable word that could have caused the observed word. This probability is computed by taking into account the probabilities of confusion between letters (which represents channel constraints) and the probabilities of cooccurring n-grams (which represents contextual knowledge). Given an m-letter input word, the method can be viewed as a shortest path algorithm through a graph of 26xm nodes called a trellis.

The Algorithm

The formal statement of the text enhancement procedure based on the Viterbi algorithm follows.

```
repeat
    get-word (X);
    generalized-viterbi (X,Z);
    word-out (Z)
until end-of-file;
```

The procedure *generalized-viterbi* to be stated below is for the case of a first-order Markovian assumption.

Symbols and Data Structures

$L_1 \ldots L_{26}$ represents symbols A...Z, and the delimiter symbol is $\emptyset$.

C is a vector of 26 real numbers called the cost vector;

S is a vector of 26 character strings called the survivor vector;

$\underline{X} = X_1 \ldots X_m$ is the input character string and

$\underline{Z} = Z_1 \ldots Z_m$ is the output character string.

$\bar{A}$ is a 26xm binary matrix of alternatives whose columns are labelled $A_1 \ldots A_m$;

element $A_i(j)$ represents the jth element of A_i such that if $A_i(j) = 1$ then letter L_j is a possibility for Z_i, otherwise L_j is considered impossible for Z_i.

max$(a_1 \ldots a_{26})$ returns the maximum of $\{a_1 \ldots a_{26}\}$, *max-index* $(a_1 \ldots a_{26})$ returns index of maximum of $\{a_1 \ldots a_{26}\}$, *concat* (s, L_j) concatenates character L_j at the end of string s.

```
procedure generalized-viterbi
    (X1...Xm, Z1...Zm);

(* given an m-letter string X, produce an
   m-letter string Z *)

begin
  initialize (Ā);
  trace (Ā, C, S, X1...Xm);
  Z:=select (Ā, C, S)
end;(*generalized-viterbi*)
```

Procedure *initialize* selects either a fixed number of alternatives (d) or a variable number of alternatives for each letter of the input word. This is done by setting $A_i(j)$ to 1 if log $P(Xi|L_j)$ + log $P(Lj)$ is among the d best out of the 26 possibilities or if the same sum exceeds a threshold value.

Next we describe the recursive path-tracing step in procedure *trace*, which returns a set of character strings in S where costs are defined by C.

```
procedure trace (Ā, C, S, X1...Xi);
begin
  if i>1 then
  begin
    trace (Ā, C, S, X1...Xi-1);
    C1:=C; S1:=S; (* C1 and S1 are local *)
    for j:=1 to 26 do
      if Ai(j) =1 then
        begin
```

```
            for k:=1 to 26 do
              if A_{i-1}(k) =1 then g_k:=C1(k)+
                    logP(X_i/L_j)+logP(L_j/L_k)
              else g_k:=-∞
            C(j):=max(g_1,...,g_26);
            u:=max-index(g_1,...,g_26);
            S(j):=concat(S1(u), L_j)
          end
        end
      else  (*i=1*)
            for j:=1 to 26 do
              if A_1(j) =1 then
                begin
                 C(j):=logP(X_1/L_j)+logP(L_j/∅);
                 S(j):=L_j
                end
end; (*trace*)
```

The final step of the algorithm is performed by function *select* that returns the number of S with the least overall cost.

4. EXPERIMENTAL RESULTS

Experiments with the two algorithms were conducted on word lists that were extracted from two corpora: Corpus A (of 6296 words) was drawn from an artificial intelligence text while corpus B (of 90,235 words) consists of corpus A plus the text of Pascal and Lisp manuals on an on-line documentation system. Two data-bases to simulate OCR garbled words were generated as follows. The first is a data-base of words with single substitution errors obtained by introducing errors into the above word lists by a random choice of position and letter to be substituted. The second is a data-base of words with a variable number of substitution errors per word (which includes no errors) using the same correct word lists; these errors were generated using the probabilistic noise model assumed by the Viterbi algorithm.

4.1 Performance of binary n-gram algorithm

A Pascal version of the binary n-gram algorithm for n=2 and 3, and the single substitution error hypothesis was implemented on a CDC Cyber 174. The algorithm was tested using corrupted versions of $\mathcal{D}_5$, $\mathcal{D}_6$ and $\mathcal{D}_7$ generated by the single and variable substitution error noise models. These tests consisted of three sets of experiments (I, II and III) corresponding to the use of positional, nonpositional-1 and nonpositional-2 n-grams respectively. In each of these trials, the set of binary digrams and trigrams used were compiled from the subdictionary of interest. The percentage of errors detected and corrected are given in Table 1.

Experiment I was conducted with the complete positional binary n-grams, $B(2,m)$ and $B(3,m)$. Experiment II used a single non-positional n-gram array per subdictionary, i.e., length-dependent n-grams. Experiment III was conducted with the least memory overhead, a single non-positional n-gram array (length-independent) representing $\mathcal{D}_5$, $\mathcal{D}_6$ and $\mathcal{D}_7$.

As expected, overall performance was best in experiment I which used the highest level of local contextual knowledge. In this case both digrams and trigrams provided reliable error detection (over 83%) and only trigrams provided acceptable error correction (over 70%). With non-positional binary n-grams, only the trigrams provided acceptable error detection (over 70%). It was observed that algorithm performance varies directly with the storage required for the n-gram arrays. Also, the performance of the algorithm deteriorates with increasing size of the subdictionary $\mathcal{D}_m$. The behavior of the positional binary trigram algorithm with increasing dictionary size is shown in Fig. 1 which represents results of experiments in which the base dictionary consisted of all seven letter words from corpus B and two additional sources (a hardware user's manual and a spelling dictionary). This yielded 5920 words from which ten lists were randomly selected using a uniform distribution. The algorithm was tested using garbled words obtained by randomly introducing a single substitution error in each word as before. Although the detection rate remains reasonably high with increasing dictionary size, the correction rate decreases rapidly.

The storage requirements of linear marginal indexing (LMI) to represent the positional trigram arrays extracted from corpus B are shown in Fig. 2 together with the storage requirements for exhaustive storage. The exhaustive method requires 2,197 bytes per array and an overall storage requirement of 1,570,855 bytes. Using LMI the same arrays derived from corpus B were represented with 316,419 bytes, thus reflecting an 80% decrease in storage; the degree of compaction achieved by LMI is highest for very short and very long word lengths and lowest for middle word lengths.

Performance of Viterbi algorithm

Experiments were conducted with a Pascal implementation of the Viterbi algorithm. The necessary transitional probability arrays were estimated from the text that was input to the noisy channel. It is noted that probability estimates from source text will differ from estimates from the lexicon of the source text, which is in contrast to binary n-gram arrays which are identical for both.

Error correction percentages resulting from three experiment sets (IV,V and VI) corresponding to WLPD, WLDPI and WLPI transitional probabilities are reported in Table 2. The results pertain to both the single and variable number of substitution errors per word. All experiments were run with d=5, a choice found to be experimentally optimal for the noise model.

The highest error correction percentage resulted with second-order WLDPI probabilities (experiment V). With only first-order probabilities, the best performance was with WLPD probabilities. Performance improves, though not linearly, with storage requirements for the probability arrays. Although first-order WLPI probabilities required the least storage, the performance was significantly worse than with second-order WLPI and WLDPI probabilities. Second-order probabilities yield considerably better results than first-order probabilities for WLDPI and WLPI cases, but are not significantly better for WLPD.

5. COMPARISON AND CONCLUSIONS

The binary n-gram and Viterbi algorithms are alternative approaches to using context at the word-level in OCR. Since both algorithms are time-efficient, their storage requirements (for representing contextual knowledge) provide a measure of algorithm complexity. Whereas the binary n-gram algorithm utilizes a set of n-gram arrays to represent the legality of letter combinations with respect to a lexicon, the Viterbi algorithm (and its generalization) utilizes transitional probability arrays to represent the frequency of occurrence of contiguous letter combinations of a source text.

The binary n-gram algorithm is unable to adapt to channel characteristics, on the other hand channel constraints in the form of confusion probabilities are effectively used by the Viterbi algorithm. For example, the Viterbi algorithm performs significantly better with variable number of errors than with single substitution errors whereas the binary n-gram algorithm shows little or no difference between the two noise models.

The principal strength of the binary n-gram algorithm is in its use as an effective error detection procedure. The reported experiments indicate the performance vs. storage trade-off in implementing the algorithm. With positional binary trigrams, the percentage of errors detected is very high (over 98%) even with large lexicon size. The error detection capability is good (over 90%) even with positional digrams and non-positional-1 trigrams. The error correction capability of the algorithm is acceptable (over 70%) only with positional trigrams. However, the storage requirements of the positional trigram algorithm (which is 1.6 mbytes with exhaustive representation and typically .3 mbytes with the proposed method of linear marginal indexing) may preclude its usage in practice.

The error correction capability of the Viterbi algorithm is superior to that of the binary n-gram algorithm while operating under similar amounts of storage (excluding the exorbitant positional trigrams for the binary n-gram algorithm and second-order WLPD probabilities for the Viterbi algorithm). Furthermore, the Viterbi algorithm can be used in feature vector classification unlike the binary n-gram algorithm which can only be used with hard decisions provided by a classifier; however, the Viterbi algorithm can only be used for error correction unlike the binary n-gram algorithm which can be used either for error detection or correction.

The fact that the correction rates of these algorithms for large vocabularies (or source text) are low in absolute terms shows that either further contextual information must be brought to bear or at least a hybrid approach that utilizes both structural and statistical information is needed. A hybrid method of using the generalized-Viterbi algorithm with a lexicon is considered in [7] where it is shown that performance is significantly improved by simultaneously searching the Viterbi trellis as well as a lexicon.

ACKNOWLEDGEMENTS

The authors wish to thank Ms. Eloise Benzel who prepared the manuscript. This work was supported by the National Science Foundation under grant IST-80-10830 which is gratefully acknowledged.

REFERENCES

1. W. Doster and J. Schurmann, An application of the modified Viterbi algorithm used in text recognition, Proc. 5ICPR, 853-855, 1980.
2. E.G. Fisher, The use of context in character recognition, Ph.D. dis., U.Mass.1976.
3. G.D. Forney, Jr., The Viterbi Algorithm, Proc. IEEE, 61, 268-278, 1973.
4. J.J. Hull and S.N. Srihari, Experiments in text recognition with binary n-gram and Viterbi algorithms, IEEET-PAMI, to appear.
5. E.M. Riseman and A.R. Hanson, A contextual post-processing system for error correction using binary n-grams, IEEET-C, 20, 480-493, 1974.
6. R. Shinghal and G.T. Toussaint, Experiments in text recognition with the modified Viterbi algorithm, IEEET-PAMI, 184-192,1979.
7. S.N. Srihari, J.J. Hull and R. Choudhari, Integrating diverse knowledge sources in text recognition, Proc. ACM-SIGOA Conf., Phila., PA, June 1982.

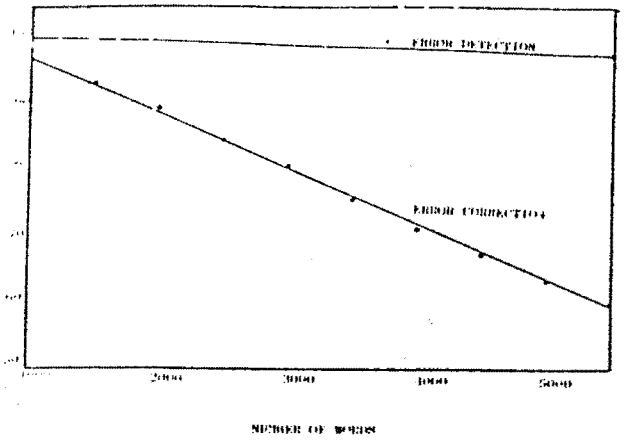

Fig.1. Performance of positional trigram algorithm with increasing lexicon size for 7-letter words.

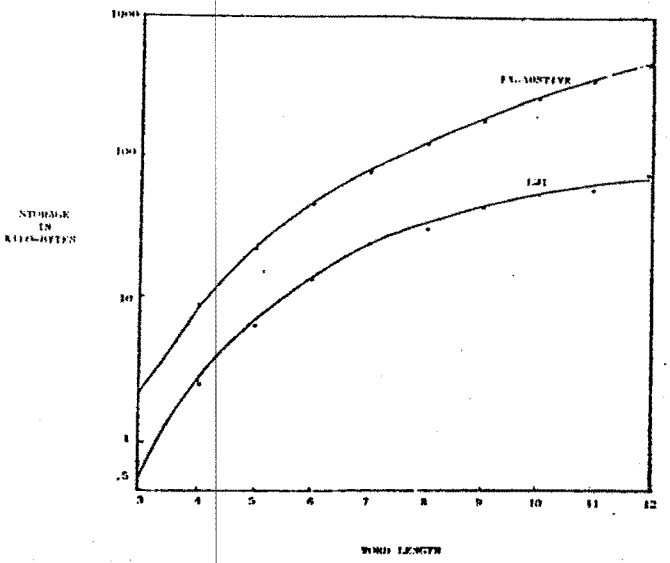

Fig.2. Storage requirements for positional trigrams of corpus B.

TABLE I

Performance of binary n-gram algorithm

(Each pair of entries represents percentage of substitution errors detected and corrected)

Word Length n	Text Source Corpus	EXPERIMENT I: Positional — Digrams: Variable	Digrams: Single	Trigrams: Variable	Trigrams: Single	EXPERIMENT II: Non-positional-1 (word length dependent) — Digrams: Variable	Digrams: Single	Trigrams: Variable	Trigrams: Single	EXPERIMENT III: Non-positional-2 (word length independent) — Digrams: Variable	Digrams: Single	Trigrams: Variable	Trigrams: Single
5	A	100/35	97/30	100/79	100/92	75/0	70/0	99/30	95/33	40/0	31/0	70/1	85/4
	B	92/6	83/7	100/70	99/80	49/0	45/0	90/13	90/17	35/0	28/0	83/5	80/5
6	A	99/39	97/38	100/83	100/97	66/1	60/1	100/27	94/36	38/0	33/0	82/8	84/4
	B	83/8	85/6	100/76	100/93	50/0	45/0	91/12	90/15	38/0	32/0	85/5	88/8
7	A	99/31	97/32	100/78	100/97	62/0	58/0	97/28	95/31	44/0	28/0	84/3	93/3
	B	90/8	90/10	100/77	100/95	42/0	41/0	91/11	91/10	34/0	30/0	85/5	89/6

TABLE II

Performance of Viterbi algorithm

(The entries are percentage of words corrected)

Word Length n	Text Source Corpus	EXPERIMENT IV: Word-length and position dependent — 1st Order: Variable	1st Order: Single	2nd Order: Variable	2nd Order: Single	EXPERIMENT V: Word-length dependent and position independent — 1st Order: Variable	1st Order: Single	2nd Order: Variable	2nd Order: Single	EXPERIMENT VI: Word-length and position independent — 1st Order: Variable	1st Order: Single	2nd Order: Variable	2nd Order: Single
5	A	61	23	58	25	31	14	74	25	20	9	35	19
	B	16	13	19	14	19	9	40	19	15	8	25	15
6	A	63	25	64	28	39	15	68	30	19	12	40	20
	B	25	15	40	16	12	9	44	19	16	9	35	17
7	A	63	23	--	--	34	15	72	28	17	11	38	20
	B	27	15	--	--	13	10	39	19	15	10	37	18