Detection of word error position and correction using reversed word dictionary

Conference Paper · January 1998

CITATIONS
3

READS
29

2 authors, including:

Bidyut Baran Chaudhuri
Indian Statistical Institute
361 PUBLICATIONS **9,987** CITATIONS

SEE PROFILE

Detection Of Word Error Position And Correction Using Reversed Word Dictionary

B. B. Chaudhuri and T. Pal
Computer Vision & Pattern Recognition Unit
Indian Statistical Institute
203, Barrackpore Trunk Road
Calcutta - 700 035

Abstract

A new novel technique of localization and correction of non-word error is described. In this technique a candidate string S of n characters is searched in the conventional dictionary D_c. If S is a non-word, its first $k_1 \leq n$ characters will match with a word in D_c. (If $k_1 = n$ then the word in D_c must be longer than n). A reversed word dictionary D_r is also generated where the characters of the word are maintained in a reversed order. If the last k_2 characters of S match with a word in D_r then, for single error, it is located within the intersection region of first $k_1 + 1$ and last $k_2 + 1$ characters of S. We observed that this region is very small compared to word length for most cases and the number of suggested correct words can be drastically reduced using this information. We have used our approach in correcting Bangla text, where the problem of inflection is cleverly tackled.

1. Introduction

The problem of detecting error in words and automatically correcting them is a great research challenge. Its solution has enormous application potentials in text and code editing, computer aided authoring, Optical Character Recognition (OCR), machine translation, natural language processing, database retrieval and information retrieval interface, speech recognition, text to speech and speech to text conversion, communication system for the disabled (e.g. blind and deaf), computer aided tutoring and language learning, desktop publication, and pen-based computer interface.

The word error can belong to one of the two distinct categories namely, **nonword** error and **real word error**. Let a string of characters separated by spaces or punctuation marks be called **a candidate string**. A candidate string is a valid word if it has a meaning. Else, it is a nonword. By real word error we mean a valid but not the intended word in the sentence, thus making the sentence syntactically or semantically ill-formed or incorrect. In both cases the problem is to detect the erroneous word and either suggest correct alternatives or automatically replace it by the appropriate word.

There are several issues to be addressed in the error correction problem. The first issue concerns the error patterns generated by different text generating media such as typewriter and computer keyboard, typesetting and machine printing, OCR system, speech recognizer output, and of course, handwriting. Usually, the error pattern of one media does not match with that of the other. The error pattern issue of each media concerns the relative abundance of insertion, deletion, substitution and transposition error, run-on and split word error, single versus multiple character error, word length effect, positional bias, character shape effect, phonetic similarity effect, heuristic tendencies etc. The knowledge about error pattern is necessary to model an efficient spell checker.

Another important issue is the computerized dictionary which concerns the size of the dictionary, the problem of inflection and creative morphology, the dictionary file structure, dictionary partitioning, word access techniques and so on. Dictionary look up is one of the two principal ways of spelling error detection and correction. The other approach, popularly used in OCR problems, is the N-gram approach. Construction of appropriate N-gram from raw text data is an important issue in this approach.

The detection of real word error needs higher level knowledge compared to the detection of nonword error. In fact, detection of real word error is a problem that needs Natural Language Processing (NLP) tools. Quite often, it is not possible to separate the problem of real word error detection from that of correction.

Even for non-word errors, correction is a nontrivial task. Several approaches based on minimum edit distance, similarity key, rules, N-grams, probability and neural nets are proposed to accomplish the task [1-8,13,14]. Of these, minimum edit distance based approaches are the most popular ones. The minimum edit distance is the minimum number of editing operations (insertions, deletions and substitutions) required to transform one text string into another. The distance is also referred to as Damerau-

Levensthein distance after the pioneers who proposed it for error correction [4,8]. In its original form, minimum edit distance algorithms require m comparisons between the misspelled string and the dictionary of m words. After comparison, the words with minimum edit distance are chosen as correct alternatives. To improve the speed, a reverse minimum edit distance is used where a candidate set is produced by first producing every possible single-error permutation of the misspelled string and then checking the dictionary if any make up valid word. For a brief description of other methods, See [7].

Irrespective of the technique used, one of the aims of a spell-checker is to provide a small set of correct alternatives for a erroneous string which includes the intended word. If the number of correct alternatives becomes one then the correction can be done automatically. Even if the number is small, it is manually convenient to choose from this small subset.

The set of correct alternatives can be drastically reduced if the error detection algorithm can pinpoint the position and nature of error (substitution or deletion etc.) occured in the misspelled string. Consider the case of single position error. Suppose the error detection algorithm could find that the error has occured at the k-th character position of a string. Suppose it could also find that the error is a substitution type. Then we accept only those valid words formed by replacing k-th character of the string by other characters. Thus, the correct alternatives are smaller in number than those obtained by considering substitutions at every character position of the string. Moreover, the generation of a smaller number of correct alternatives can be made much faster.

In this paper we propose a technique of error detection that can pinpoint the error position in a big majority of cases and thus reduce the number of correct alternatives to a large extent. The approach is based on matching the string in the normal as well as reversed dictionary that is elaborated below. To the best of our knowledge no work has been reported that tackles the detection of error position and error type in a misspelled string. From this standpoint our effort is a pioneering one.

In the following sections we describe our approach for correcting single position error although we shall show that our approach can correct multiple errors in a large number of situations.

While our approach is applicable to any alphabetic language, we concentrate here on text of Bangla, the fourth most popular language in the word. This language, being a reasonably inflectional one, poses some problems in error detection and correction effort. Our approach to solve the problems is also described.

2. Reversed word Dictionary and Error Positioning

2.1 Reversed word dictionary

For a valid word, its reversed word is a string of characters in reversed sequence. Thus, the reversed version of the words "read" and "copy" are the strings "daer" and "ypoc", respectively where the first character of the word goes to the last position, the second character occupies the last but one position and so on. In general, the reversed word of a word $W = x_1x_2 \ldots x_k$ is $W_r = x_kx_{k-1} \ldots x_2x_1$.

In a reversed word dictionary D_r, reversed version of all dictionary words are maintained. For quick access or retrieval, the words can be alphabetically ordered, partitioned in terms of word length and maintained in indexed flat file or in trie structure. The dictionary structure for our purpose can be indexed or trie depending on the system capability. We have used trie structure for our purpose, because it is computationally faster to access.

The purpose of reversed word dictionary is to look for match of a string S backwards from the last character. We shall show that search in conventional dictionary D_c as well as reversed word dictionary togather helps in finding the error position in S as well as in creating a small subset of correction candidate words which indeed contains the intended word.

2.2 Error detection and positioning

Here our aim is to detect the erroroneous word and also to find the position in the word where the error has occured. To start with, we have the following assumptions. Later on, we shall examine how much relaxation of the assumption No. 1 can be tackled by our method.

1. There can be only single error in the word which is one among insersion, deletion, substitution and transposition.
2. The correct word is in both the dictionary (conventional and reversed word) files.

We treat the case of insersion and transposition separately from the deletion and substitution. If the error is caused due to insertion (transposition), the correct word is a deleted (transposed) string. If there are n characters in a misspelled word, we can make n different strings by deleting one character at a time. Similarly, n - 1 strings can be generated by transposing one pair of neighboring characters. These $2n$ - 1 strings may be checked in the conventional dictionary and the strings that are valid words are included in the candidate set of correct words. The number of these words is not large, since for $n = 6$ (which is more than the average wordlength in many languages) $2n$ - 1 is a small number.

However, string generation in this way to consider deletion and substitution is not economical unless $n = 1, 2$ because for an alphabet size of N characters, $2nN$ strings may be generated. A large number of these strings may be valid words, thereby increasing the number of candidate set of correct words to a large extent. Here we propose an alternative approach guided by conventional and reversed word dictionaries so that the candidate set is substantially reduced, especially for large n. Moreover, we shall show that our approach can find, with reasonable accuracy, the position in the string where the error has occured.

Consider, an erroneous string S of n characters. Suppose we try to match the string in conventional dictionary D_c and check the dictionary word that matches at maximum number of character positions say k_1 in a sequence starting from left. For example, let the erroneous string be "forvune" where the error has occured at 4th position and the correct word is "fortune". In the dictionary, there are several words with "for ..." but no word with "forv ...". Thus, here $k_1 = 3$.

Note that since the error is a single substitution or deletion, the correct word will lie in the dictionary words of n and $n + 1$ characters. While searching in D_c and D_r, we look only for words of length n and $n + 1$, unless otherwise stated.

Now, the following proposition is true for any erroneous string S.

***Proposition* 1**: If for an erroneous string S the longest word match in conventional dictionary D_c occurs for the first k_1 characters then the error has occured in the first $k_1 + 1$ characters of S.

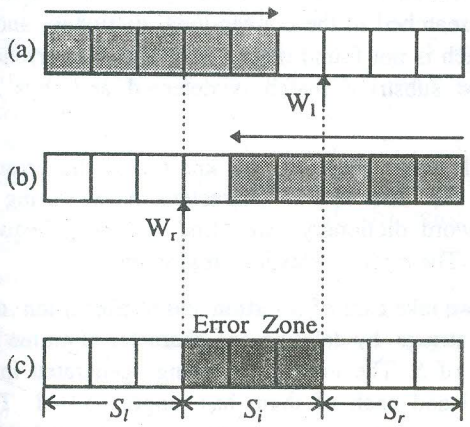

Fig. 1. Error localization by conventional and reverse dictionary.
(a) Error localization by conventional dictionary.
(b) Error localization by reverse dictionary.
(c) Error localization when both dictionaries are used.

For an illustration of the proposition, see Fig. 1(a). To prove the proposition, let it be false i.e. let the error be not in the first $k_1 + 1$ characters. Since the first $k_1 + 1$ characters are error-free, then we could find at least one word in the dictionary, whose $k_1 + 1$ characters match with those of the string S (ensured by the assumption 2 whereby the correct word is in the dictionary). This is a contradiction, since the longest dictionary match occurs for first k_1 characters and not for $k_1 + 1$ characters. •

If we use reversed word dictionary D_r for finding the longest match, then we can find, say the last k_2 characters matching with those of S and the error must have occured within last $k_2 + 1$ characters of S. The argument about this fact is in the same line as that of proposition 1. See also Fig. 1(b). We may consider the above inference as reversed dictionary version of proposition 1.

Thus, proposition 1 gives an idea of where the error has occured in S. If $k_1 = 0$ i.e. if there is no valid word in the conventional dictionary whose first character is the same as that of S then the error has occured at (in case of substitution error) or before (in case of deletion error) the first character – a perfect positioning of the error. To find correction candidate words we use the reversed word dictionary and look for words whose tail end match with the character string of S excepting the first character (which takes care of substitution error) and if possible, including the first character (which takes care of deletion error). The set of candidate words is small if n is reasonably large say, $n \geq 3$.

If $k_2 = 0$ i.e. there is no word whose last character matches with that of S then too we know that the error occured at or after the last character of S. The other characters of S are correct and we can use them as key for selecting the correction candidate words, from the conventional dictionary.

The above approach is effective, if k_1 or k_2 is small compared to n. We can use a rule of thumb that if $n - k_1 > 2$ or $n - k_2 > 2$ then we use either conventional dictionary or reversed word dictionary to find correction candidates. However, k_1 or k_2 may not be small, especially when the error occurs at the middle position of the word. To take care of the situation we use the following stronger proposition.

***Proposition* 2**: If for an erroneous string S the longest match in conventional dictionary occurs for the first k_1 characters and the longest match in the reversed word dictionary occurs for the last k_2 characters, then the error has occured at the intersection of the first $k_1 + 1$ and the last $k_2 + 1$ characters of S.

For an illustration, see Fig.1. To prove the proposition we note that if the error is outside the intersection region then we could get a longer match either in conventional dictionary or in the reversed word dictionary, which is a contradiction. •

Proposition 2 allows us to pinpoint the error in S to a large extent. To get an estimate about the width of the intersection region, say S_i, we made a simulation study on conventional dictionary of 55,000 Bangla words and its reversed word version. About one hundred words were randomly chosen. Deletion and substitution errors were

generated at all positions of them, making about 30,000 erroneous strings. For each such string S, its S_i was detected. It was found that cardanality of S_i is one character (perfect pinpointing) in 41.36% cases, two characters in 32.96% cases, three characters in 16.58% cases. See Table 1.

***Proposition* 3**: If $|S_i| = 1$ the error occured at S_i must be due to substitution.

To prove proposition 3 we note that here we are dealing with substitution and deletion error (transposition and insertion error is already taken care of) and the error occurs at one position only.

Now if a deletion error has occured then the deleted character must have been immediately to the left or to the right of S_i. If the deleted character was to the right of S_i, then S_i is not erroneous and starting from left of S we could get a match of the substring upto and including S_i in the conventional dictionary. This is a contradiction and hence the error cannot be due to deleted character to the right of S_i. Similar argument holds about deletion to the left of S_i. Therefore, the error is due to substitution only. •

Thus according to proposition 3 we can not only pinpoint the error position in some cases, we can also discover the type of error occured in those cases.

On the other hand, if $|S_i| = 2$ the error may have occured at one of the two characters of S_i (substitution) or in between the two characters (deletion). In general, if $|S_i| = m$ the error may have occured at one of the m characters of S_i (substitution) or at one of the $m - 1$ positions between neighboring characters of S_i (deletion). Note that in the worst case $|S_i| = n + 2$ is possible.

3. Error Correction

3.1 Correction Strategy

The region of S excluding S_i is error-free. Let the error free region to the left and right of S_i be S_l and S_r, respectively (See Fig. 1(c)). Now, we can search in the dictionary for the words which matches with S_l at the beginning and with S_r at the end. This candidate set must contain the correct word. If S_i is short compared to S (in number of characters) then the candidate set will also be small in number.

Note that this approach can take care of multiple errors occured within S_i region if the condition of checking only words having length n and $n + 1$ is relaxed. However, it cannot work if one of the errors occurs in S_l or S_r regions and the other occurs in S_i.

If S_i is large then a different approach may be used to contain the size of candidate set. This approach is invoked if $|S_i| > \lfloor n/2 \rfloor$. In the above condition, we take the first $\lfloor n/2 \rfloor = m$ characters of S and find the set W_1 of all valid words in the dictionary whose first m characters match with them. We also take the last $n - m$ characters of S and find the set W_2 of all valid words in the reversed word dictionary whose last $n - m$ characters match with them. Union of these two sets of valid words is the candidate set which contains the (intended) correct word. This is certainly so because if the error has occured in the last half of S then the correct word belongs to W_1. If the error is in the first half, then the correct word belongs to W_2. Note that either W_1 or W_2 but not both can be null subsets.

Table 1

Error zone length (in no. of characters)	% of words
1	41.36
2	32.94
3	16.58
4	7.10
5	1.78
6	0.24
Error located at either end of error zone.	90.77

The candidate set generated in this way can take care of multiple errors occured either in the first m or last $n - m$ positions of S provided we do not put any wordlength constraint during dictionary search. However, it cannot take care of the situation where one error occurs in the first m and the other error occurs in the last m characters.

3.2 Algorithmic procedure

Error detection is a relatively straightforward task. The string S is searched in the conventional dictionary and if a perfect match is not found then S is declared as erroneous. The longest substring match is detected and thus k_1 is known.

For single error situation, we know that the error has occured in the first $k_1 + 1$ characters. Now, using the reversed word dictionary we find k_2, and subsquently compute S_i. The error resides in S_i region only.

At first, we take care of insertion and transposition error. We create strings by deleting one character at a time from S_i portion of S. The number of strings generated in this way is $|S_i|$ and each of them has length $|S| - 1$. These strings are searched in the conventional dictionary and if some of them matches with the dictionary word then they are accepted as the set of correction candidate words. If an insertion error occured then this set certainly contains the correct word. However, if the error is of type other than insertion and if this set is not null (which is highly unlikely but not impossible) then this set may not contain the intended word. Let this set of words be W_1.

To take care of transposition we transpose the neighboring pair of characters and generate strings. Thus, at most $n - 1$ strings of length $|S|$ are formed. The strings

are searched in the conventional dictionary and those matching with valid words are accepted. Let this set of words be W_2. If the error is a transposition then W_2 would contain the intended word. However, if the error is not a transposition and if W_2 is non-null then it may not contain the intended word.

Now we consider the deletion and substitution errors. Note that if S is a string with deletion error then its correct version must be found among words of length $|S|+1$. Similarly, correct version of a substitution error must be a word of length $|S|$.

If S is small, say $|S| \leq 3$ then we adopt the following approach. Let $|S| = 2$ and $S = x_1x_2$. Then we find all valid dictionary words of length 2 and 3 that either started with x_1 or ended with x_2. Then, this set must contain the intended word, if the error is either deletion or substitution. For $|S| = 3$ let $S = x_1x_2x_3$. Then we find all valid dictionary words of length 3 and 4 that (i) either started with x_1x_2 (ii) or has the first character x_1 and x_2 (iii) or has last two characters x_2x_3.

In general, let W_3 be the set of correction candidates obtained while considering the deletion and insertion error. The size of W_3 (i.e. the number of words in W_3) can be made smaller by doing additional check that the error occured at only one character position. Thus, if a correction candidate word say S' differ from S at more than one character position then S' is not included in W_3. In this way, some candidates of length $|S|+1$ are deleted.

If $|S_i| > |S|$ then we correct S according to Cases 1-3 described above.

Now, let S be of large size say $|S| \geq 4$. If $|Si| \leq \frac{1}{2}|S|$ then we find the S_l and S_r and find the candidates whose first characters are S_l and last characters are S_r and which satisfy the constraint that the candidate S' differ from S in single position only. If $|S| \leq |S_i| > \frac{1}{2}|S|$ then we partition S into two segments S_l' and S_r' so that $\{|S_l'| - |S_r'|\}^2 \leq 1$. Candidates are generated whose first characters are S_l' or last characters are S_r' and which satisfy the constraint that the candidate S' and S differ in single position only. Here too, search is made among words of length $|S|$ and $|S+1|$ only. The union of W_1, W_2 and W_3 must contain the intended word if the error in S has occured in single position.

4. Bangla text error correction

Bangla is an inflectional language. A word in the text may contain rootword appended with suffixes (with a probability of 0.7). The distinct set of surface words may be 100 times those of root words (Nouns and Verbs). Hence the dictionary size may be huge if all surface words are maintained. Creation of new surface words in this language is an easy task, and thus the dictionary can never be completed.

So, we compiled our dictionaries with root words only. We maintained also several suffix files, each file containing suffixes for a class of words, depending on their grammatical and semantic category.

Consider now a candidate string S. If it is a valid word then either it can be matched in D_c which indicates that S is a root word or part of S can be matched in D_c which indicates that S can be an inflected word. In the latter case the rest of S should be searched and matched in the appropriate suffix file.

Suppose S is not a valid word. If S contains single error then it has occured at the suffix or root-word part. Thus, if we have a root-word match then the suffix lexicon is searched. If a failure is reported, attempt is made to correct the suffix part. If no valid suffix exists with unit edit distance then the corrected version of S could be a root-word. Using D_c and D_r, the list of possible correction words are generated and their minimum edit distances are computed. Those having unit distance are accepted as alternatives.

If S does not match (fully) with any word in D_c then at first alternatives are generated using D_c and D_r. Those having unit distance are accepted as alternatives. If none is found then suffix match is attempted, which must succeed if S contains single error. Now again correction is attempted using D_c and D_r on the rest of S.

Some discussions on the suffix files are in order. We have initially divided them into two main groups namely verb and non-verb files. Among non-verbs, noun and adjective suffix files are further distinguished. For nouns, again case and non-case suffix (e.g. plurality, gender etc.) are distinguished. Even animate, inanimate, human, non-human distinctions are made. Now, in the main dictionary D_c, the parts of speech and other informations about the word as well as the suffix file it should refer to, are maintained. Thus, when a part of S is matched in D_c as valid root word, the pointers attached to the root word points to the suffix files that should be searched for a suffix match in S. In this way the search can be speeded up and false grammatical agreement between the root-word and suffix can also be detected. As a result, our spell-checker acts as a morphological parser.

Note that the scheme stated above can detect any error and correct single error accurately. For multiple errors, correction (i.e. preserving a set of alternative words that must contain the intended error) may not always be possible.

5. Discussion

We have implemented our approach in PC under Windows-95 environment. The main dictionary of about 60,000 root words are stored in the PC. To this another 20,000 inflected words are added. These words occupy the top 20,000 positions in the word-frequency count of a corpus of 3 million words obtained from various sources. These 80,000 words are maintained in a trie structure. The reason of using 20,000 top-ranking words is to avoid suffix

search as much as possible. Our experience is that the system works fast if we use this scheme.

We have an option of getting warning while entering the corpus. If an invalid character is typed at certain position of a word, a warning is issued. For example, only 90 compound characters (out of about 280) can occupy the first position of a Bangla word. Also, no Bangla word contains a character repeated thrice. Similarly, many bigrams are impossible in Bangla words. These and errors like incomplete brackets, punctuation marks etc. are served with on-line warning.

For off-line word error detection and correction we followed the format of WORDSTAR. A temporary dictionary is maintained where the user may enter words which are not in the main dictionary. The temporary dictionary takes care of proper nouns and specialized terms. The program asks for the parts of speech of such words since in Bangla, they can also be inflected.

While running on a corpus of 250,000 words we found that our system works with high accuracy. The non-word errors are all correctly detected (i.e. false rejection is 3000) but the system makes about 5% of false acceptance. They are mainly due to conjunct words formed due to euphony and assimilation as well as proper nouns in the corpus. We are planning to take care of euphony and assimilation in near future.

Acknowledgement: The authors wish to thank Mr. P. Kundu of Vidyasagar College, Calcutta and Mr. N. S. Dash of this department for providing the suffix list and for valuable discussions.

References

1. R. C. Angell, G. E. Freund, and P. Willett, "Automatic spelling correction using a trigram similarity measure", *Inf. Process. Manage*, Vol.19, pp.255-261, 1983.
2. V. Cherkassky and N. Vassilas, "Back-propagation networks for spelling correction", *Neural Net.*, Vol.1, No.3 (July), pp. 166-173, 1989.
3. K. W. Church and W. A. Gale, "Probability scoring for spelling correction", *Stat. Comput.*, Vol.1, pp. 93-103, 1991.
4. F. J. Damerau, "A technique for computer detection and correction of spelling errors", *Commun. ACM*, Vol.7, No.3(March), pp. 171-176, 1964.
5. R. E. Gorin, "SPELL : A spelling checking and correction program", Online documentation for the DEC-10 computer, 1971.
6. S. Kahan, T. Pavlidis, and H. S. Baird, "On the recognition of characters of any font size", *IEEE Trans Patt. Anal. Machine Intell.* PAMI-9, No.9, pp. 274-287, 1987.
7. K. Kukich, "Techniques for automatically correcting words in text", *ACM Computing Surveys*, Vol.24, No.4.(Dec.), pp. 377-439, 1992.
8. V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", *Sov. Phys. Dokl.*, Vol.10(Feb.), pp. 707-710, 1966.
9. U. Pal and B. B. Chaudhuri, "Computer recognition of printed *Bangla* script", *Int. J. Syst. Sci.*, Vol.26, No.11,pp. 2107-2123, 1995.
10. J. J. Pollock and A. Zamora, "Automatic spelling correction in scientific and scholarly text", *Commun. ACM-27*, No.4(April), pp. 358-368, 1984.
11. P. Sengupta and B. B. Chaudhuri, "A morpho-syntactic analysis based lexical subsystem", *Int. J. Pattern Recog. and Artificial Intell.*, Vol.7, No.3, pp. 595-619, 1993.
12. P. Sengupta and B. B. Chaudhuri, "Projection of Multi-Worded lexical entities in an inflectional language", *Int. J. Pattern Recog. and Artificial Intell.*, Vol.9, No.6, pp.1015-1028, 1995.
13. R. Singhal and G. T. Toussaint, "Experiments in text recognition with the modified Viterbi algorithm", *IEEE Trans. Patt. Anal. Machine Intell.*, PAMI-1, No.4(April), pp.184-193, 1979.
14. E. J. Yannakoudakis and D. Fawthrop, "An intelligent spelling corrector", *Inf. Process Manage*, Vol.19, No.12, pp.101-108, 1983.