ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 12, December 2016

A Novel and Adaptive Approach for String Transformation

Dr. B Sankara Babu

Professor, Department of CSE, Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad, Telangana, India

ABSTRACT: There are several problems in NLP, data mining, information retrieval can be formalized as string transformation, which is a task as follows. Given an input string, the system generates the k similar strings corresponding to the given string. We propose an approach to find a string using string transformation, which is both accurate and efficient. The approach includes the use of 0-1 Knapsack problem, a method for training the model, and an algorithm for generating the nearest string, whether there is or is not a predefined dictionary. The learning method employs maximum likelihood estimation for parameter estimation. The proposed method is applied to correction of spelling errors in queries as well as reformulation of queries in web search. Experimental results on large scale data shows that the proposed approach is very accurate and efficient improving upon existing methods in terms of accuracy and efficiency in different settings.

KEYWORDS: Natural language processing, Levenshtein distance, Knapsack problem, edit distance

I. INTRODUCTION

In computer science, edit distance is a way of measuring how to differentiate two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one to another string. Edit distance finds applications in natural language processing, where automatic spelling correction can determine candidate corrections for a misspelt word by selecting words from dictionary that have a low distance to the word in the query. In bioinformatics, it can be used to measure the similarity of DNA sequence, which can be viewed as string of letters A, C, G and T. Multiple definitions of an edit distance use different sets of string operations. The operations of Levenshtein distance are removal, insertion, and substitution of a character in the given string.

A. LEVENSHTEIN DEFINITION:

Given two strings a and b on an alphabet ∑ (e. g., the set of ASCII characters, the set of bytes [0…255], the edit distance d(a, b) is the minimum-weight series of edit operations that transforms a into b. One of the simplest sets of edit operations is that insertion of single symbol.

If a=uv, then inserting symbol X produces UXV. This can be said as ε ->X, using ε to denote the empty string.

Deletion of a single symbol changes UXV to UV (X -> ε).

Substitution of a single symbol X for symbol Y ≠ X changes UXV to UYV (X ->Y).

II. LITERATURE REVIEW

Edit distance finds applications in computational biology and natural language processing, e.g. the correction of spelling mistakes or OCR errors, and approximate string matching, where the objective is to find matches for short strings in many longer texts, in situations where a small number of differences is to be expected. Various algorithms exist that solve problems beside the computation of distance between a pair of strings, to solve related types of problems. Hirschberg's algorithm computes the optimal alignment of two strings, where optimality is defined as minimizing edit distance.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 12, December 2016

Approximate string matching can be formulated in terms of edit distance. Ukkonen's 1985 algorithm takes a string p, called the pattern, and a constant k; it then builds a deterministic finite state automaton that finds, in an arbitrary string s, a substring whose edit distance to p is at most k[10] (cf. the Aho–Corasick algorithm, which similarly constructs an automaton to search for any of a number of patterns, but without allowing edit operations). A similar algorithm for approximate string matching is the bitmap algorithm, also defined in terms of edit distance.
Levenshtein automata are finite-state machines that recognize a set of strings within bounded edit distance of a fixed reference string.[4]

III. PROPOSED METHOD

There are two versions of problem

a) Fractional knapsack problem The setup is same, but the thief can take fractions of items, meaning that the items can be broken into smaller pieces so that thief may decide to carry only a fraction of xi of item i, where $0 \leq xi \leq 1$.
b) 0-1 knapsack problem The setup is the same, but the items may not be broken into smaller pieces, so thief may decide either to take an item or to leave it (binary choice), but may not take a fraction of an item.

Dynamic-Programming Solution to the 0-1 Knapsack Problem
Let i be the highest-numbered item in an optimal solution S for W pounds. Then S` = S - {i} is an optimal solution for W - wi pounds and the value to the solution S is Vi plus the value of the subproblem.We can express this fact in the following formula: define c[i, w] to be the solution for items 1,2, . . . , i and maximum weight w. Then

$$c[i,w] = \begin{cases} 0 & \text{if } i = 0 \text{ or } w = 0 \\ c[i-1, w] & \text{if } w_i \geq 0 \\ \max [v_i + c[i-1, w-w_i], c[i-1, w]\} & \text{if } i>0 \text{ and } w \geq w_i \end{cases}$$

This says that the value of the solution to i items either include i^{th} item, in which case it is v_i plus a sub problem solution for (i - 1) items and the weight excluding wi, or does not include i^{th} item, in which case it is a subproblem's solution for (i - 1) items and the same weight. That is, if the thief picks item i, thief takes vi value, and thief can choose from items w - w_i, and get c[i - 1, w - wi] additional value. On other hand, if thief decides not to take item i, thief can choose from item 1,2, . . . , i- 1 upto the weight limit w, and get c[i - 1,w] value. The better of these two choices should be made.
Although the 0-1 knapsack problem, the above formula for c is similar to LCS formula: boundary values are 0, and other values are computed from the input and "earlier" values of c. So the 0-1 knapsack algorithm is like the LCS-length algorithm given in CLR for finding a longest common subsequence of two sequences.
The algorithm takes as input the maximum weight W, the number of items n, and the two sequences v = <v1, v2, . . . , vn> and w = <w1, w2, . . . , wn>. It stores the c[i, j]values in the table, that is, a two dimensional array, c[0 . . n, 0 . . w] whose entries are computed in a row-major order. That is, the first row of c is filled in from left to right, then the second row, and so on. At the end of the computation, c[n, w] contains the maximum value that can be picked into the knapsack.

```
Dynamic-0-1-knapsack (v, w, n, W)
FOR w = 0 TO W
  DO  c[0, w] = 0
  FOR i=1 to n
   DO c[i, 0] = 0
   FOR w=1 TO W
     DO IFf wi ≤ w

THEN IF  vi + c[i-1, w-wi ]
THEN c[i, w] = vi + c[i-1, w-wi]
  ELSE c[i, w] = c[i-1, w]
```

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 12, December 2016

ELSE

c[i, w] = c[i-1, w]

The set of items to take can be deduced from the table, starting at c[n. w] and tracing backwards where the optimal values came from. If c[i, w] = c[i-1, w] item i is not part of the solution, and we are continue tracing with c[i-1, w]. Otherwise item i is part of the solution, and we continue tracing with c[i-1, w-W].

III. RESULTS

String transformation is defined as required transformations that are necessary for the replacement of source string with different destination strings. In Natural language Processing, String Transformation is illustrated as Misspelt word correction, word metamorphosis, word transition. In the process of string transformation, we applied knapsack method.

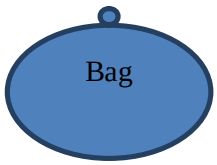

Fig.4.1. Bag can accommodate words

Fig 4.1. Represents the bag which can accommodate maximum weight, using the Natural Language Processing and weights are defined using 'w'. The String transformation is applied using the weighted words called Misspelt word correction, word metamorphosis, word transition.

The set of strings with the weights are represented in table 4.1. These weights help us to find the best fit into the destination bag(String) in the string transformation process.

Table 4.1 Bag of words with weight

A	**c**	**c**	**O**	**u**	**n**	**t**	
1	3	3	15	21	14	20	77
A	**c**	**c**	**U**	**r**	**a**	**t**	**E**
1	3	3	21	18	1	20	5	72
A	**c**	**c**	**u**	**r**	**a**	**t**	**E**	**I**	**Y**
1	3	3	21	18	1	20	5	12	25	109
A	**c**	**c**	**u**	**s**	**e**	
1	3	3	21	19	5	52

In the above table 4.1 represents the assigning of the weights to the individual characters and the sum is calculated for each word. For example A,C,C,O,U,N,T are assigned with weights 1, 3, 3, 15, 21, 14, 20 and the total weight of the word ACCOUNT is 77 is indicated at the end of its column.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 12, December 2016

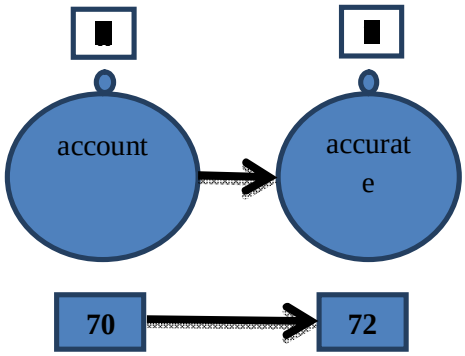

Fig.4.2. String Transformation with nearest weight

In Fig. 4.2.,We can accommodate the maximum weight in the Y which is nearest to the correct string. Fig 4.2 shows X=> "acconnt" string but it is incorrect string with weight (70). In the string transformation process Y =>"accurate" which is not the desired string (72).

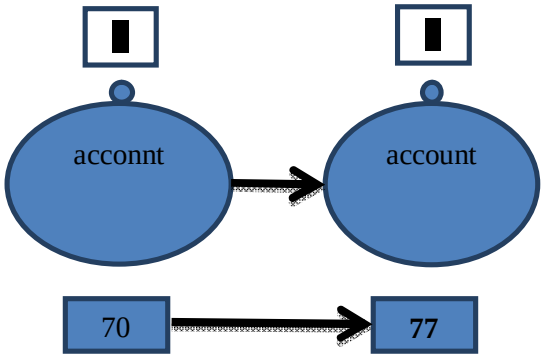

Fig. 4.3. String Transformation with correct weight

Fig 4.3 shows X=> "acconnt" string but it is incorrect string with weight (70). In the string transformation process Y =>"account" which is the desired string (77). In fig 4.2, it shows X=>Ystring, which is not nearer to the characters in the destination string but fig 4.3 shows the exact match.

IV. CONCLUSION

We proposed an approach for finding the useful patterns in the text documents which is referred to as the text mining and then when any misspelt word is present in them, then we must replace them with the correct word. And many misspelt words can be present and thus we should initially find the number of exact word present and then replace them with the correct word. So, we proposed a Knapsack based in String Transformation. In this approach each alphabet is allocated with unique precision value. Data sets are trained by calculating the total weight of the words and validating through the dictionary called database. Now we apply the knapsack method to replace a character to determine the new string. The System generates the output strings which can be similar to Misspelt Word. Accuracy is very high when compared to the existing methods.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 12, December 2016

REFERENCES

1. Misspell word correction, word metamorphosis, word transition Hadjieleftheriou, Marios, and Chen Li. "Efficient approximate search on string collections." Proceedings of the VLDB Endowment 2.2 (2009): 1660-1661.
2. Arasu, Arvind, SurajitChaudhuri, and RaghavKaushik. "Learning string transformations from examples." Proceedings of the VLDB Endowment 2.1 (2009): 514-525.
3. Tejada, Sheila, Craig A. Knoblock, and Steven Minton. "Learning domain-independent string transformation weights for high accuracy object identification." Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002.
4. Zobel, Justin, and Philip Dart. "Phonetic string matching: Lessons from information retrieval." Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1996.
5. Vernica, Rares, and Chen Li. "Efficient top-k algorithms for fuzzy search in string collections." Proceedings of the First International Workshop on Keyword Search on Structured Data. ACM, 2009.
6. Dreyer, Markus, Jason R. Smith, and Jason Eisner. "Latent-variable modeling of string transductions with finite-state methods." Proceedings of the conference on empirical methods in natural language processing. Association for Computational Linguistics, 2008.
7. Navarro, Gonzalo, et al. "Faster approximate string matching over compressed text." Data Compression Conference, 2001. Proceedings. DCC 2001.. IEEE, 2001.
8. Bergsma, Shane, and GrzegorzKondrak. "Alignment-based discriminative string similarity." Annual meeting-Association for Computational Linguistics. Vol. 45. No. 1. 2007.
9. Yang, Zhenglu, Jianjun Yu, and Masaru Kitsuregawa. "Fast Algorithms for Top-k Approximate String Matching." AAAI. 2010.
10. McCallum, Andrew, KedarBellare, and Fernando Pereira. "A conditional random field for discriminatively-trained finite-state string edit distance." arXiv preprint arXiv:1207.1406 (2012).