www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 7 July 2017, Page No. 21916-21921
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i7.05

Lexical Analysis of Devanagari Hindi Language

Alok Kumar[1], Saurabh Sharma[2], Mushahid Raza[3]
[1]Assistant Professor, Department of Computer Science and Engineering
University Institute of Engineering and Technology,
C.S.J.M. University, Kanpur, India
akumar.uiet@gmail.com
[2]B.Tech Student, Department of Computer Science and Engineering
University Institute of Engineering and Technology,
C.S.J.M. University, Kanpur, India
join2saurav3@gmail.com
[3]B.Tech Student, Department of Computer Science and Engineering
University Institute of Engineering and Technology,
C.S.J.M. University, Kanpur, India
raza.uiet2k13@gmail.com

Abstract: *With recent increase in uses of internet and development of web technologies, proportion of web content in Hindi is increasing at a lightning impetus. These particulars can play a vital role for researchers and computer science engineers in devising systems and real-world application for government organisation and private sector which could ease their decision making process. In this paper, we present a modus operandi to perform lexical analysis on Devanagari Hindi language. This was done by building a dictionary containing Hindi words using few information retrieval techniques and implementing some error recovery strategies to recover from lexical errors (if occurred). We achieved an accuracy of approximately 88% while performing Lexical analysis on different Hindi words. Main advantage of our work is, entire processing of the Lexical analysis phase can be completed in offline mode (i.e. without any usage of internet).*

Keywords: Lexical analysis of Hindi Text, Lexical errors recovery strategies for Hindi Sentences, Tokenization of Hindi, Hindi text editor with error correction strategies, Natural language processing.

1. Introduction

Natural language processing is the branch of Computer Science with two basic goals:

i. Understanding grammar and rules to work upon the specified Natural language.
ii. Build the system that analyze the Natural language from different aspects and miniaturised the man-machine gap.

Natural language Hindi has more than 500 million native speakers which are spread across the world and with the increase in usage of internet these native speakers are actively playing an important role in increasing the amount of information in Hindi present in web pages across the internet. Data from these web pages can be mined for useful insight in developing various applications. In the field of Natural Language processing, there is not much work done for Hindi language and our work is among the first few works that have used Hindi as an input Natural language.

The main task of Lexical analyzer is to read the sequence of characters from the source language and produce as output the meaningful sequence of characters and when we consider Natural language Hindi, meaningful sequence of characters is nothing but the meaningful words of Hindi language so our work mainly focuses upon implementing various strategies and techniques to cover all the aspects of Lexical analyzer by taking Hindi as an input Natural language and we have implemented these strategies in such a way that entire processing of the Lexical analysis phase can be done offline (i.e. without any usage of internet). This feature makes our work stand apart from other works done in the same area. Our research work contributed in development of dictionary of Hindi words and then performing Lexical analysis to detect errors (if any) in the words typed on text editor that was created by us using Unicode's of Hindi. We then used some error recovery algorithms to find most suitable suggestions that could be used in place of an erroneous word.

Assumption: We have assumed that dictionary always contains collection of meaningful words present in Hindi.

Rest of the paper is structured as follows: Section 2 gives exhaustive view of proposed system along with the detailed description of the mechanism followed to create dictionary of Hindi words, approach used to detect the lexical error and algorithms designed to recover from the lexical errors. Section 3 shows descriptive analysis of the results obtained. Section 4 discusses related work in this area. Section 5 presents the conclusion along with direction for future works. Section 6 mentions references that gave us direction to finish our work.

2. Proposed System

Entire system proposed by us can be divided into following phases:

i. Creation of dictionary of meaningful Hindi words.
ii. Developing a Hindi text editor using Unicode's of Hindi language.
iii. Detection of Lexical errors.
iv. Implementing error recovery strategies.
v. Generating a pop-up list showing 10 most probable suggestions for a word having lexical error.

Overall processing of the above mentioned phases is described in sub-sections below:

2.1 Creation of Dictionary

For creating a dictionary of meaningful Hindi words we extracted information from web pages of the websites (mainly Hindi newspaper websites) and then accordingly adjusted information extracted from online sources in such a way that we got a file containing Hindi words as an end result. The entire task of creation of dictionary involved four steps; these were executed in the sequence as described in the figure shown below.

2.1.1 Crawling & Scrapping Website

We designed an algorithm for crawling websites and scrapping Hindi content from it. Steps of algorithm are described below:

i. A website address that contains Hindi text is passed as an input, our algorithm looks for the HTML source/script of the home page of specified website and then it searches for head reference tags (<href>).
ii. After finding head reference tag, algorithm copies its value (as value of head reference tag is link to next webpage of the website) on a separate file (say FILE_1).
iii. Repeat Step ii for all head reference tags present in homepage of the website.
iv. Take a unique link as input from FILE_1 (file containing links to next web pages from homepage).
v. Open HTML source/script of the webpage link taken as input and then search for head reference tags (<href>).
vi. When a head reference tag is found, copy its value if it is not present in FILE_1.
vii. Repeat Step vi for all head reference tags present in the link taken as input in step iv.
viii. If there are some links present in FILE_1 that have not been opened yet then goto step iv else goto step ix
ix. Take pointer to starting index of FILE_1 and open the HTML source of the link present at start index.
x. Search for paragraph tags (<p>) and copy value of paragraph tag (as value of paragraph tag is all the text written on the webpage) on a separate file (say FILE_2).
xi. Repeat Step x for all paragraph tags present in the link whose HTML source has been opened.
xii. Move pointer to next index of FILE_1, open its HTML source and Repeat Step x and Step xi until EOF for FILE_1 is encountered.

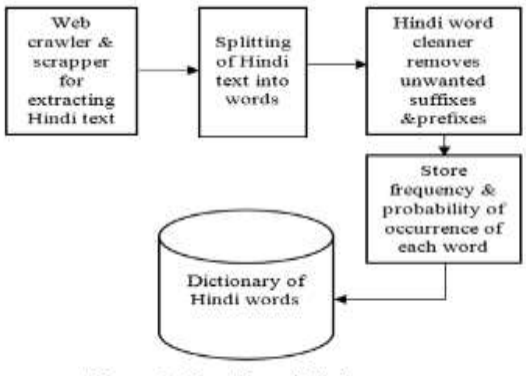


Figure 1: Creation of dictionary

After completion of these steps we got a file (i.e.FILE_2) containing raw Hindi data.

2.1.2 Breaking Of Raw Hindi Text Into Words

Raw Hindi data obtained after completion of algorithm described in section 2.1.1 was broken into individual words of Hindi Language by following a very simple algorithm described below:

i. Read line by line the entire file obtained after successful execution of algorithm described in section 2.1.1 until EOF is encountered.
ii. Split the line read into words by finding white spaces between the sequences of characters present in that line.
iii. Store all such sequences obtained after splitting in a separate file (say FILE_3).

After successful execution of these steps we got a file containing Hindi words (i.e.FILE_3).

2.1.3 Cleaning Of Hindi Words

Hindi words obtained after execution of algorithms described section 2.1.2 contained unwanted suffixes and prefixes. For ex- Words "**..राम"** , "**23राजा"**, "**रातef"** contain unwanted prefix ".." in "**राम"** and **"23"** in **"राजा"** respectively and unwanted suffix "**ef**" in **"रात".**

These kind of unwanted suffixes and prefixes lead to meaningless word formation so we removed these kind of unwanted suffixes and prefixes by designing an algorithm described below:

i. Read line by line the entire file(FILE_3)obtained after successful execution of algorithm described in section 2.1.2 until EOF is encountered.
ii. Use regular expression to eliminate any unwanted symbol from the line/string read and Store the resultant string obtained in a separate file (say FILE_4).

Line read in Step i of above mentioned algorithm was a string, storing a Hindi word that had possibility of having unwanted suffixes and prefixes symbols (Unwanted symbols can be symbols like: ';', '.', '/',a-z,A-Z,0-9 etc.).

2.1.4 Evalutating Probability Of Occurrence Of Words

FILE_4 obtained in section 2.1.3 gave us clean Hindi words and in order to generate a pop list showing 10 most probable

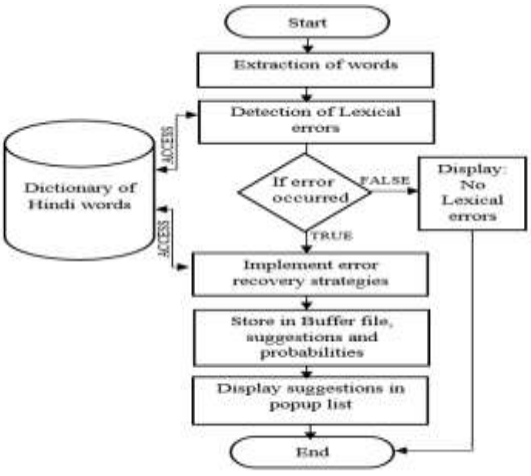


Figure 3: Detection & correction of Lexical errors

suggestions (as described in section 2.5), we evaluated frequency of each word present in FILE_4 and then divided

frequency value of each word with total number of unique words present in FILE_4. This evaluation gave us probability of occurrence of each word. Then probability values computed and frequency of occurrence of each word were also stored along with corresponding word in FILE_4.We then sorted this file in an order (from all the words starting with 'अ' to all the words starting from 'ज्ञ'). Sorting was essential because we used binary search (which requires data to be in sorted order) to search suggestion words from dictionary for an erroneous word. Binary search reduced time taken to search possible suggestions from dictionary in case of Lexical errors. We call this file (FILE_4) obtained after sorting as dictionary of Hindi words.

By taking any Hindi newspaper website as an input, we got approximately 15000 meaningful Hindi words by following the procedure described in Section 2.1. After creating dictionary, internet connection was no longer required and entire Lexical analysis was done without it.

2.2 Developing Hindi Text Editor

We developed a Hindi text editor by using Unicode's of characters of Hindi language as keys. This text editor acted as front end of our entire Lexical processing. It contained all basic functionality of cut, copy, paste, edit, save etc. that any text editor possesses. It contained an additional key named "LEX" that detected erroneous word present in text area of text editor.

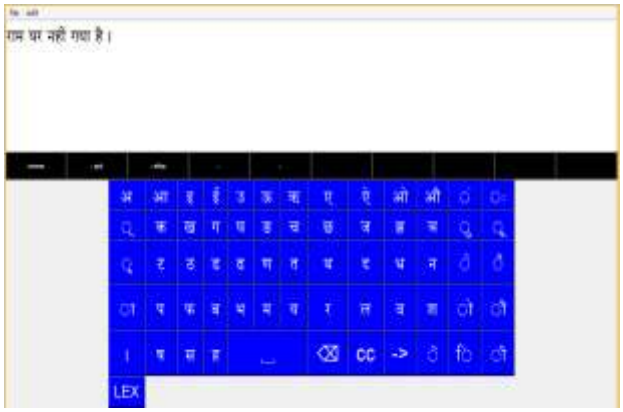


Figure 2: Screenshot of our text editor.

2.3 Detection of Lexical Errors

Once we got dictionary containing Hindi words Lexical error were detected by the algorithm described below:

i. Read the entire sentence typed by user on the text editor's text area.
ii. Split the sentence into individual words (by looking at white spaces).
iii. Take a word from splatted sentence and perform binary search to find matching word in the dictionary (i.e.FILE_4).
iv. If match is found for the word then, leave the word as it and increase its frequency of occurrence by one in dictionary and accordingly compute its new probability value else underline it.

Thus our work successfully detected invalid tokens (i.e. basically meaningless words in context with the Hindi language) by underlining words for which possible match had not been found.

After detection of the Lexical errors next step was to recover from such errors this was done by implementing error recovery strategies only on underlined words as described in the next section.

2.4 Implementation Of Error Recovery Strategies

Error recovery strategies found possible suggestion words for a word having lexical errors (underlined word).

Error recovery strategies used by us can be divided into four sections:

i. Recovery by swapping
ii. Recovery by deletion
iii. Recovery by insertion
iv. Recovery by replacement

Each of these algorithm uses of a file buffer named FILE_BUFFER 1 to store possible suggestion words.

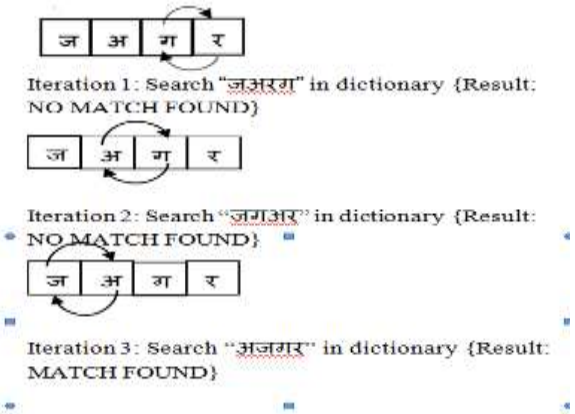
2.4.1 Recovery By Swapping

Basic idea behind Recovery by swapping is, some characters of word might have been swapped.

We implemented recovery by swapping by designing an algorithm described below:

i. Swap two adjacent characters from the end of the word that has Lexical error.
ii. Perform binary search to find possible match of the word formed after swapping in the dictionary.
iii. If the possible match is found then, store it along with its probability value in a FILE_BUFFER 1 else swap next two adjacent characters and goto Step ii. Step iii terminates after starting two characters had been swapped and binary search had been performed for the resulting word.

For ex-Instead of word "अजगर" user typed word "जअगर" which is meaningless. Our algorithm can find correct word after three iteration as described below:



2.4.2 Recovery By Deletion

Basic idea behind recovery by deletion is, some extra characters might be present in erroneous word.

We designed an algorithm for recovery by deletion in such a way that it executes in two phases as described below:

PHASE 1:

i. Delete a character from end of the word that has lexical error.

ii. Perform binary search in the dictionary to find possible match of the word formed after deleting character from the end.
iii. If match is found then, store word along with its probability value in FILE_BUFFER 1 else delete the next character from end of remaining word and repeat Step ii. Deletion of the characters in step iii continues until remaining word is left with two characters.

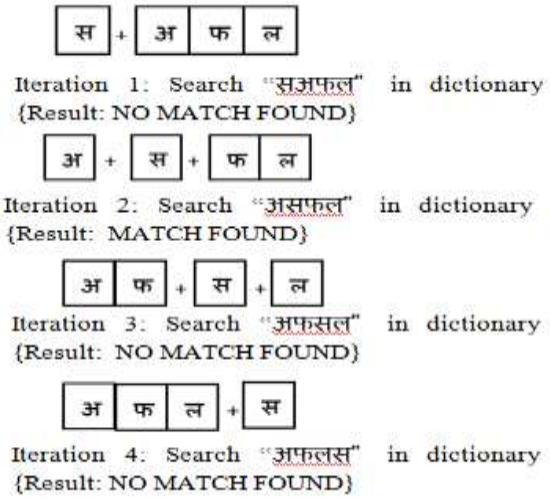
PHASE 2:

i. Delete a character from start of the word that has lexical error.
ii. Perform binary search in the dictionary to find possible match of the word formed after deleting character from the start.
iii. If match is found then, store word along with its probability value in FILE_BUFFER 1else delete the next character from start of remaining word and repeat Step ii. Deletion of the characters in step iii continues until remaining word is left with two characters.

2.4.3 Recovery By Insertion

Basic idea behind recovery by insertion is, a character might be missing from erroneous word.
Algorithm designed by us for recovery by insertion makes use of regular expressions to find possible matches for erroneous word.
Algorithm involves following steps:
Global Boolean variable: check
i. Initialize check ← FALSE
ii. Take pointer to beginning of erroneous word.
iii. Insert a character of Hindi at current pointer position.
iv. Perform binary search in dictionary to search for all possible matches of a word formed after insertion of character and store results along with corresponding probabilities in FILE_BUFFER 1.
v. Move pointer by one position on right hand side.
vi. If pointer is at not at end of word then goto Step ii else goto Step vii.
vii. If check = FALSE then assign check ← TRUE and goto Step ii else terminate algorithm.

For ex- Instead of word "असफल" user typed "अफल" which is meaningless then above mentioned algorithm will try to insert each character of Hindi in between every character of erroneous word and also at start and end positions respectively but for simplicity, let us consider case of insertion only for Hindi language character "स".



2.4.4 Recovery by Replacement

Basic idea behind recovery by replacement is, neighboring key from the keyboard might have been pressed instead of correct key while typing the word that is showing error. We implemented recovery by replacement by designing an algorithm described below:
Assumption: Neighboring keys are keys positioned at immediate left, right, top, bottom of a key in keyboard.
i. In each pass iterate a loop that searches in dictionary for all possible matches of a word formed after replacing a character at i^{th} position of erroneous word with a character corresponding to its neighbouring key. Results for possible matches along with their probability values are stored in FILE_BUFFER 1. (Initially i= 0)
ii. If number of pass is less than or equal to number of character then increase value of 'i' by one (i←i+1) else terminate algorithm.

For ex- Let's say portion of text editor's keyboard looks as shown in Figure 4 and user typed "जगथ" instead of "जगत" then Pass 1, Pass 2, Pass 3 will replace characters 'ज', 'ग' and 'थ' respectively . Let us see iterations of pass 3 (i.e. pass of replacing 'थ'):

Figure 4

Iteration 1: Replace 'थ' with 'त'

ज	ग	त

Search "जगत" in dictionary {Result: MATCH FOUND}

Iteration 2: Replace 'थ' with 'द'

ज	ग	द

Search "जगद" in dictionary {Result: NO MATCH FOUND}

Iteration 3: Replace 'थ' with 'छ'

ज	ग	छ

Search "जगछ" in dictionary {Result: NO MATCH FOUND}

PASS 3: Neighbours of 'थ'='त', 'द', 'छ', 'र'

Iteration 4: Replace 'थ' with 'र'

ज	ग	र

Search "जगर" in dictionary {Result: NO MATCH FOUND}

2.5 GENERATION OF POPUP LIST

Popup list contained two things:

i. List of 10 most probable suggestions
ii. Option to add erroneous word into dictionary

2.5.1 LIST OF 10 MOST PROBABLE SUGGESTIONS

As error recovery algorithms described in section 2.4 were storing all possible words/suggestion in FILE_BUFFER 1, we selected 10 most probable suggestions by an algorithm having following steps:

i. Sort in descending order all words present in FILE_BUFFER 1 with respect to their probability values.
ii. Select top 10 words from FILE_BUFFER 1 and add them to popup list.
iii. Add option of "add to dictionary" at the end of popup list.
iv. Display popup list on text editor.
v. Once user chooses a suggestion from popup list, increase frequency value of corresponding word in dictionary by one and compute its new probability value accordingly.

Popup list gave flexibility to user to choose any word shown on the list in place of word for which Lexical error had been detected.

2.5.2 Option To Add Errorneous Word Into Dictionary

Detection of Lexical error as described in section 2.3 involves matching of words present in text area of text editor with words present in dictionary but since dictionary was created by retrieving words present in websites therefore it only contains words that are frequently used by native Hindi speakers. As exact amount of words present in Hindi language is incalculable so their might be a situation that word typed by user was correct but due to its absence in dictionary it had been underlined (i.e. lexical error detected) thus option of "add to dictionary" for such words gave us flexibility to add a newly encountered words into our dictionary. This flexibility makes the system designed by us more and more powerful with each time of its use. Our system automatically learns new words that are being added hence it keeps on increasing accuracy of detecting lexical errors. Once option of add word into dictionary from popup list is chosen, we designed a small algorithm that performed two things:

i. Remove underline from erroneous word.
ii. Add word at proper place in dictionary along with frequency value =1 and probability calculated by dividing frequency value (i.e. 1) with total number of words present in dictionary.

3. Result Anaysis

Our method of Lexical analysis correctly detected lexical error 1769 times when we tested it with 2000 Hindi words, showing an accuracy of approximately 88%.Further, Table 1 shows number of words that error recovery algorithms described in section 2.4 were able to correct when we took a sample size of 500 Hindi words for each case. Results shown in Table 1 highlights the fact that error recovery algorithms described by us gave uneven results and we cannot point that particular algorithm of error recovery is best since it totally depends on the user's habit of typing words in the text editor and the type of mistakes person usually commits.

Error recovery strategy	Number of Hindi words tested	Number of times error recovery strategy found correct word
Recovery by swapping	500	447
Recovery by deletion	500	376
Recovery by insertion	500	423
Recovery by replacement	500	434

Table 1

The most crucial advantage of the system proposed by us is that percentage of accuracy of system to detect Lexical errors and recover from them is not constrained to a fixed value and as our system keeps on learning new Hindi words (By using system again and again) percentage of accuracy keeps on increasing.

Our only assumption throughout the process was, we considered dictionary as the collection of meaningful/correct Hindi words.

4. Related Research Works

Pradipta Ranjan Ray, Harish V., Sudeshna Sarkar and Anupam Basu[1] designed an algorithm for local word grouping to extricate fixed word order dependencies in Hindi sentences. Local word grouping in their algorithm was achieved by defining regular expressions for the word.
Seema Mahato and Dr. Ani Thomasproposed[2] proposed an automated essay grading system to overcome the issues involved evaluating grammatical and semantic error and to overcome influence of local and regional languages in Hindi essays.
K.Panchapagesan, Partha Pratim Talukdar, N. Sridhar Krishna, Kalika Bali and A.G. Ramakrishnan [3] reported an ongoing effort on Hindi Text Normalization by using novel approach where tokenization and initial token classification were performed using a lexical analyzer that was derived from various token definitions in the form of regular expressions.

Akshat Bakliwal, PiyushArora andVasudeva[4] presented a graph based Word Net expansion method to generate adjective and adverbs lexicon by using synonyms and antonyms relation to expand the initial Hindi lexicon.

Veena Dixit, Satish Dethe and Rushikesh K. Joshi[5] designed a spellchecker for Marathi (an Indian Language) by rules of morphology of Marathi.

Muhammad Humayoun and Aarne Ranta[6] developed Punjabi Morphology, Corpus and Lexicon by using GF (Grammatical Framework, Ranta 2004). It is a framework for developing multilingual grammar applications which provides a built-in morphological analyzer and generator for the input languages.

5. Conclusion and Future Works

Lexical analysis of Hindi language is first task involved in building artificially intelligent and robust machines that take Hindi as input natural language. Correctness of these machines is measured on the basis of proportion of ambiguities that machine is able to resolve and this entirely depends on number of errors successfully detected and corrected by all phases of processing Hindi language. Since Lexical analysis is a starting step in the process therefore it becomes very important task to detect and recover from as many errors as possible in this starting phase itself. We can proudly say that our system is capable of detecting such lexical errors with high accuracy.
In order to have faster processing we used Binary search and since after a significant period of time, amount of words present in Hindi dictionary would adversely increase therefore work can be done to implement other methods of searching like hashing, B-tree traversal etc. in order to save time spent by error recovery algorithms in searching possible suggestions from dictionary. Once a sentence typed in the text area of our text editor completes Lexical analysis, work can be done to implement strategies that would suggest best possible parse tree for the given sentence(i.e. work can be done in syntax processing stage) moreover sentence's semantically correctness can also be checked provided it is syntactically correct.

6. References

[1] Pradipta Ranjan Ray, Harish V., Sudeshna Sarkar, Anupam Basu. Part of Speech Tagging and Local Word Grouping Techniques for Natural Language Parsing in Hindi.

[2] Seema Mahato, Dr. (Mrs.) Ani Thomas. Lexico-Semantic analysis of essays in Hindi language.

[3] K. Panchapagesan, Partha Pratim Talukdar, N. Sridhar Krishna, Kalika Bali, A.G.Ramakrishnan. Hindi Text Normalization.

[4] Akshat Bakliwal, Piyush Arora, VasudevaVarma. Hindi Subjective Lexicon: A Lexical Resource for Hindi Polarity Classification.

[5] Veena Dixit, Satish Dethe, Rushikesh K. Joshi. Design and Implementation of a Morphology-based Spellchecker for Marathi, an Indian Language.

[6] Muhammad Humayoun, Aarne Ranta.Developing Punjabi Morphology, Corpus and Lexicon.

[7] Niladri Dash, Pushpak Bhattacharyya, Jyoti Pawar (eds.), Wordnets of Indian Languages, Springer, ISBN 978-981-10-1909-8, 2016.

[8] Richard Sproat, AlanBlack, Stanley Chen, Shankar Kumar, Mari O stendorfand, Christopher Richards. Normalization of Non-Standard Words.

[9] Kapadia Utkarsh N., Desai Apurva A.Morphological Rule Set and Lexicon of Gujarati Grammar:A Linguistics Approach.

[10] Christopher Olston and Marc Najork. 2010.Web Crawling.

[11] Mini Singh Ahuja, Dr Jatinder Singh Bal, Varnica. 2014.Web Crawler: Extracting the Web Data.