Project Documentation

Root Extractor for Malayalam

Jincy Baby

International Centre for Free and Open Source Software (ICFOSS)

Project Overview

Extracting root of a word is vital in the preprocessing stage of most Language processing systems for Malayalam. Even though the requirement of developing a root extractor system is crucial, we hardly ever see the development of any such powerful system to serve the purpose. A hybrid method is presented for the development of such an extractor which can strip away as many suffixes attached with the root word and as such can produce root for compound words with any number of suffixes joined together. The root extractor may prove to be efficient in research areas such as Automatic document categorization, Automatic summarization, Text mining, Machine Translation and others. The use of roots will significantly increase the efficiency of such systems over using words or stems.The current tests have provided an average accuracy of approximately 98% when worked with the words having roots in the dictionary and 90% for Nouns whose roots are not included in the dictionary.

Background and Scope

Malayalam, with its rich inflections and agglutinative nature, further complicates the preprocessing tasks essential for all the language processing applications. The most striking trait exhibited by the language is the morpho-phonemic changes the words exhibit when joining with other words or suffixes. The trait itself complicates the identification of the words and the suffixes, making it absolutely difficult to derive the root of a word. Roots refers to a form which is not further analysable, either in terms of derivational or inflectional morphology. It is that part of word-form that remains when all inflectional and derivational affixes have been removed. In the form 'acting' the root is 'act', to which the suffix '-ing' have been added.

Root extraction should be treated as the one of the most critical pre-processing tasks as it may aid most of the language processing applications. As such, the development of a Malayalam root extraction algorithm will prove to be efficient. The first phase of the proposed extractor concentrates on suffix stripping method utilizing pattern matching at the end of the string while the second phase carefully strips suffixes exploiting sandhi rules and using dictionary look up method.

Major projects related to NLP in ICFOSS faces a serious impact in the absence of a tool to succesfully generate root from a compound word. They urgently required an effective tool powerful enough to extract the root words inorder to overcome the constraints of the language and rise the accuracy rates of the individual projects. The module is currently integrated with the Plagiarism Checker for Malayalam and Parser for Malayalam.

System Description

The root extraction algorithm discussed can derive the root of any given words regardless of the number of suffixes or words attached with the stem. Two dictionaries and two suffix lists are introduced in the extractor for proper validation. Five interrelated functions, namely *stem, split,*

matra, chillu and *dvithva*, follows sandhi rules for stripping apart the suffixes and two functions, *change* and *verb*, act as supporting functions with specific purposes. The functions *split, matra* and *chillu* utilize the suffix lists and dictionaries while *stem* and *dvithva* strips apart the suffixes based on the defined rules, making the system efficient in handling unknown words. The functions being the integral part of the system must be discussed in detail to acquire a clear picture of the whole system.

The function *stem* utilizes a method similar to pattern matching where it checks if the input word endswith any pattern listed in a database specifically introduced for the function. Here, "patterns",which are the strings stored in the database, are checked for at the end of the input word. Accordingly, the database consist of a column of "patterns" and another column of "replacement strings", where each entry is the string to be attached replacing the pattern listed in the first column. The database has approximately 116 entries and the pattern entries are mostly the suffixes for Nouns. This function makes it easier to handle unknown words. Let $x_1x_2...x_tX$ be the given input. The patterns stored in the database is stored as a list initially. The function checks whether the input ends with any pattern in the list and if it find such a pattern, it obtains the corresponding replacement string stored in the database. Assume tha X is the pattern found and y is the corresponding replacement string to be attached to the word after removing the pattern. The function strips away X and replace it by y thus forming $x_1x_2...x_ty$. $x_1x_2...x_ty$ is again returned to the function as input. The process is repeated until no such pattern is found and the word is return as the result.

Algorithm(*stem*)
Input: word
Output: word with no pattern left
Steps:
1. Take the input word
2. Check the list of stopwords and return the word if found any.
3. For pattern in list do
4. If word ends with pattern do
5. Replace the pattern with the corresponding replacement string from the database
6. Re-initialize the word and go to step 1
7. Continue the process until no such pattern is found

The following shows a simple example which displays the detailed process of the function *stem*

Example1

Input Word: മാധ്യമപ്രവർത്തകരും
Process:
Input: മാധ്യമപ്രവർത്തകരും
pattern found : "ും"
Corresponding Replacement String: "ർ"
word obtained: മാധ്യമപ്രവർത്തകർ
Re-initialized Input: മാധ്യമപ്രവർത്തകർ

pattern found : "ർ്"
Corresponding Replacement String: "ർ"
word obtained: മാധ്യമപ്രവർത്തകർ
Output Word: മാധ്യമപ്രവർത്തകർ

A similar method is used in the function *verb*, with a major difference that the function is not recursive and it uses a verb dictionary for verification of the output. It converts an input word into verb form considering the pattern introduced in a database for the function.

Algorithm(*verb*)
Input: word
Output: word after replacing the pattern
Steps:
1. Take the input word
2. For pattern in list do
4. If word ends with pattern do
5. Replace the pattern with the corresponding replacement string from the database
6. If the word obtained is in the verb dictionary, return it as the output

A simple example to show the process of the function *verb* is discussed below.

Example2

Input Word: പകർന്ന്
Process:
Input: പകർന്ന്
pattern found: "ർന്ന്"
Corresponding Replacement String: "രുക"
word obtained: പകരുക
Output Word: പകരുക

Another function *dvithva* checks the input word for any string in the list [“ക്ക”,“പ്പ”,“ത്ത”, “ശ്ശ”,“ച്ച”]. It is the only function that takes ‘dvithva sandhi’ into consideration. Dvithva sandhi occurs when the first part in a morpheme doubles when joined to another word. Let $x_1x_2...x_tx_{t+1}...x_n$ be the input where $x_t = x_{t+1}$. Then the function splits the word into $x_1x_2...x_{t-1}$ and $x_{t+1}...x_n$ by removing x_t. If it finds $x_{t+1}...x_n$ in suffix list for dvithva, it returns $x_1x_2...x_{t-1}$.

Algorithm(*dvithva*)
Input: word, checklist=[“ക്ക”,“പ്പ”,“ത്ത”, “ശ്ശ”,“ച്ച”]
Output: word with the suffix, joined using dvithva sandhi, removed
Steps:
1. Take the input word
2. If any string in checklist is in word do

3. for string in checklist do
4. if word endswith the string, continue
5. else do
6. find index of the string as N
7. if word[N+2:] in suffix list for dvithva do
8. return word[:N]

An example demonstrating how the function works is described below.

Example3

Input Word: എത്തിക്കഴിഞ്ഞു
Process:
word: എത്തിക്കഴിഞ്ഞു
string found at position N: "ക്ക"
word[N+2:]: "കഴിഞ്ഞു"
word[:N]: എത്തി
Output Word: എത്തി

change is a function which supports other functions mainly to form the suffixes. The function takes in the maatra and converts it into its appropriate Malayalam letter. It converts "ാ" to "ആ", "ി" to "ഇ", "ീ" to "ഈ","ു" to "ഉ", "ൂ" to "ഊ", "െ" to "എ", "േ" to "ഏ", "ൈ" to "ഐ", "ൊ" to "ഒ" & "ോ" to "ഓ".

The function *split* is the most used among the other functions. As such, it is the vital part of the extractor. It considers the list ["ാ", "ി", "ീ", "ു", "ൂ", "െ", "േ", "ൈ", "ൊ", "ോ"] and check whether the input word contains any of the strings in the list. Let $x_1x_2...x_{t-1}x_tx_{t+1}...x_n$ be the input. The word is searched for any string included in the defined list. Assume that first such string, from the right of the word, occurs at position t. *change* operates on the string and converts it to appropriate letter. Let x_t be changed to y_t. Then, the suffix $y_tx_{t+1}...x_n$ is searched in the suffix list and if found $x_1x_2...x_{t-1}$ is treated using *stem* and *dvithva*. The process is repeated until all the suffixes are removed and obtained an output. *verb* is also employed to confirm that the function can efficiently handle verbs as well.

Algorithm(*split*)
Input: word = $x_1x_2...x_{t-1}x_tx_{t+1}...x_n$, mlist=["ാ", "ി", "ീ", "ു", "ൂ", "െ", "േ", "ൈ", "ൊ", "ോ"]
Output: word stem
Steps:
1. Take the input word
2. If in stoplist, return word
3. For k in mlist do
4. if k is in word do

5. find the last position among all the positions of k, t
6. if $x_1x_2...x_{t-1}$ ends with "ല്" , reword=$x_1x_2...x_{t-1}$
7. else reword=$x_1x_2...x_{t-1}$+“ ്”
8. if *change*+$x_{t+1}...x_n$ in suffix list do
9. if reword in dictionary, return it as output
10. else do
11. take *stem*(reword) and if non-empty, replace the output as reword
12. take *dvithva*(reword)
13. Re-initialize the word and go to step 1
14. else do
15. if in dictionary, return the word
15. else return *verb*(word)

A simple example is shown below to display the detailed process of the function *split*

Example4

Input Word: ശേഖരിക്കുന്നതെന്ന
Process:
Input: ശേഖരിക്കുന്നതെന്ന
pattern found: "ത"
suffix: എന്ന
word obtained after removing suffix: ശേഖരിക്കുന്നത്
word obtained using *stem*: ശേഖരിക്കുന്ന
Reinitialize word: ശേഖരിക്കുന്ന
Input: ശേഖരിക്കുന്ന
pattern found: "ു"
suffix: ഉന്ന
word obtained after removing suffix: ശേഖരിക്ക്
word obtained using *verb*: ശേഖരിക്കുക
Output Word: ശേഖരിക്കുക

Another function *matra* checks the input word for any string in the list [“ാ”, “ി”, “ീ”, “ു”, “ൂ”, “െ”, “േ”, “ൈ”, “ൊ”, “ോ”]. The word upto the particular string is cut off and thus split into two. Let $x_1x_2...,x_tx_{t+1}...x_n$ be the input word where x_t is the string found in the list. The word is then split into $x_1x_2...x_t$ and $x_{t+1}...x_n$. If $x_{t+1}...x_n$ is found in the suffix list or the dictionary, the system examines if $x_1x_2...x_t$ is found in the dictionary else it passes $x_1x_2...x_t$ through the function *split*. After validation of both, $x_1x_2...x_t$ or the output from *split* is returned as the output.

Algorithm(*matra*)
Input: word, mlist=[“ാ”, “ി”, “ീ”, “ു”, “ൂ”, “െ”, “േ”, “ൈ”, “ൊ”, “ോ”]
Output: word stem
Steps:

1. Take the input word
2. If word is in dictionary, then return the word
3. For k in mlist do
4. if k is in word do
5. find all indices of k and store in a list
6. For N in list of indices do
7. verbsuf=*verb*($x_{N+1}...x_n$)
8. if $x_{N+1}...x_n$ or verbsuf in suffix list or dictionary do
9. if $x_1x_2...,x_{N+1}$ in dictionary, return $x_1x_2...,x_{N+1}$
10. else if $x_1x_2...,x_N$ in dictionary, return $x_1x_2...,x_N$
11. else if *split*($x_1x_2...,x_N$+ " ◌ു") not empty, return its output
12. if $x_1x_2...,x_tx_{t+1}...x_n$ ends with k do
13. if $x_1x_2...,x_tx_{t+1}...x_{n-1}$ in dictionary, return it as the output
14. else if *split*($x_1x_2...,x_tx_{t+1}...x_{n-1}$+" ◌ു") is not empty, return its output

The following shows a simple example which displays the detailed process of the function *matra*

Example5

Input Word: പകർന്നുതന്ന
Process:
Input: പകർന്നുതന്ന
pattern found: "ു"
word splits: പകർന്ന , തന്ന
conversion of suffix using *verb*: തരുക
തരുക in dictionary
conversion of പകർന്ന് using *verb*: പകരുക
പകരുക in dictionary
Output Word: പകരുക

chillu searches the input for any string in the list ["ൽ","ർ", "ൻ", "ൾ","ൺ"]. If found, it splits the word upto the string and checks if the other half is in the suffix list or dictionary. The word derived is passed through *split* to obtain the output. Let $x_1x_2...x_{t-1}x_tx_{t+1}...x_n$ be the input. Let x_t be the string in the list. $x_{t+1}...x_n$ is searched in the suffix list or dictionary and if found, *split*($x_1x_2...x_t$) produces the output.

Algorithm(*chillu*)
Input: word=$x_1x_2...x_{t-1}x_tx_{t+1}...x_n$, chlist=["ൽ", "ർ", "ൻ", "ൾ", "ൺ"]
Output: word stem
Steps:
1. Take the input word
2. for pattern in chlist:
3. if $x_{t+1}...x_n$ in suffixlist or dictionary do

4. return *split*($x_1x_2...x_t$)

The below example depicts the working of the function *chillu*

Example6

Input Word: മഴയിൽനിന്ന്
Process:
Input: മഴയിൽനിന്ന്
pattern found: "ൽ"
suffix: നിന്ന്
word obtained after removing suffix: മഴയിൽ
word obtained using *split*: മഴ
Output Word: മഴ

The extractor exploits these functions effectively to derive the root of the words. The algorithm for the extractor gives a proper insight on the way the extractor works.
Algorithm
Input: word
Output: root
Steps:
1. Take the input word
2. Use *stem* to remove suffixes
2. Derive *chillu*(word) and re-initialize word, if non-empty
3. Take *dvithva*(word) and re-initialize word, if non-empty
4. Find *split*(word) and re-initialize word, if non-empty
5. Derive *matra*(word) and re-initialize word, if non-empty
6. Return *verb*(word) if non-empty
7. else return the word

Results

Series of experiments has been conducted to improve and test the performance of the proposed algorithm. The tests were carried out with testing corpus from various sources. The system has shown an accuracy ranging from 80% to 99% while working with the testing corpus.The downfall in the accuracy was observed incase of the poem Anushochanam by Kumaran Asan since the suffix list and dictionary lacked the type of morphemes that may appear in words used in poetic language. The extractor showed an increase in accuracy with updation of dictionary.

Corpus	words	correct roots	Percentage
Newspaper	5649	5597	99.07
Malayalam Bible	520	502	96.54
Poem	56	45	80.35

```
5    print (i,root(i))
```

ഇറങ്ങിയത് ഇറങ്ങുക
നീങ്ങുന്നതിനിടെ നീങ്ങുക
കാണുന്ന കാണുക
കണ്ടു കാണുക
കണ്ടിരുന്ന കാണുക
കണ്ടിരിക്കുന്ന കണ്ടിരിക്കുക
പാലിക്കുന്നതിന് പാലിക്കുക
കരയുകയാണെന്ന് കരയുക
പാടുകയാണെങ്കിൽ പാടുക
കണ്ടിരിക്കും കാണുക
കണ്ടിട്ടുണ്ടാകും കാണുക
കണ്ടേക്കുന്ന കാണുക
കണ്ടേക്കും കാണുക
കാണുകയാണ് കാണുക
കാണുകയാകുന്ന കാണുക
കാണുകയാകും കാണുക
കാണിക്കുകയാണ് കാണിക്കുക
കാണിക്കുകയാകും കാണിക്കുക
കാണാം കാണുക
പറയുന്നുണ്ടായിരുന്ന പറയുക
പരാജയപ്പെടുത്തിയത് പരാജയപ്പെടുത്തുക
നടന്നിരുന്നത്. നടക്കുക
നിലനിൽക്കുന്നത് നിലനിൽക്കുക

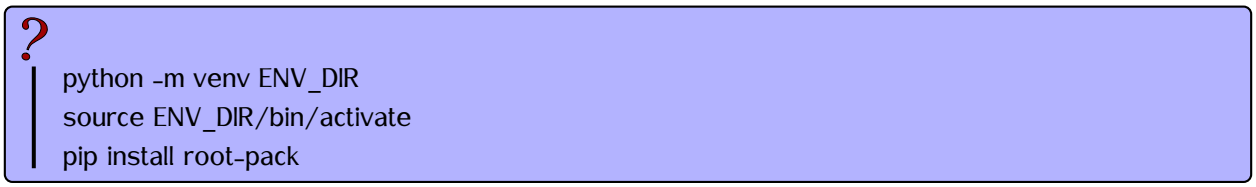
വിമാനത്താവളത്തിൽ വിമാനത്താവളം
റൺവേയിൽ റൺവേ
കൊച്ചിയിലെത്തിയ കൊച്ചി
എയർവെയ്സിന്റെ എയർവെയ്സ്
വിമാനമാണ് വിമാനം
അപകടത്തിൽപ്പെട്ടത് അപകടം
വിമാനത്തിൽ വിമാനം
യാത്രക്കാരുണ്ടായിരുന്ന യാത്രക്കാർ
ആളപായമില്ല ആളപായം
മഴയെ മഴ
ഉടർന്ന് ഉടരുക
റൺവേയിലെ റൺവേ
ലൈറ്റുകൾക്ക് ലൈറ്റ്
കേടുപറ്റി കേട്
നാശനഷ്ടങ്ങളില്ല നാശനഷ്ടം
അടിയന്തിരമായി അടിയന്തിരം
ശക്തമായ ശക്തം
കാറ്റിലും കാറ്റ്
മഴയിലും മഴ
മധ്യരേഖയിൽനിന്ന് മധ്യരേഖ
വലത്തോട്ട വലം
മാറിയാണ മാറുക
നിയന്ത്രണത്തിലാക്കാൻ നിയന്ത്രണം
പൈലറ്റിനെ പൈലറ്റ്

Summary and Conclusions

The first phase of the extractor concentrates on suffix stripping methods utilizing pattern matching and sandhi rules and the second phase carefully strips suffixes exploiting sandhi rules while using dictionary look up method. The extractor can strip away as many suffixes attached with the root word and as such can produce root for compound words with any number of suffixes joined together. It stems the suffixes adjoined with the root word one by one, eventually producing the root. Before roots are extracted, they are checked against a dictionary with 85000 entries and verbs are identified by using a list of 3956 entries. The accuracy of the proposed system improves with the updation of the dictionary.The extractor is made into a function which can aid any project in extracting the roots of the words for further processing. The user needs to tokenize the sentence and call extractor the inorder to acquire the root. The extractor faces issues when unknown suffixes are included in the word for which the root is to be found.

Installation

You may create a virtual environment for installing the package.



Otherwise, use

Implementation

After installation, you can import the module to utilize the morph() function.

For example,

output: