Project Documentation

Morpheme Generator for Malayalam

Jincy Baby

International Centre for Free and Open Source Software (ICFOSS)

Project Overview

The pre-processing stage which splits the input word into morphemes, is the major part in many language processing projects. The morpheme generator developed uses a rule-dictionary based approach, where sandhi rules and pattern matching methods are considered for the splitting of words. This part is an extension of root extractor and the output is a list containing the root of the word and its respective suffixes that are stripped from the word. The suffixes to be stripped was fixed as per the discussions with a language expert.

Introduction

A major way in which morphologists investigate words, their internal structure, and how they are formed is through the identification and study of morphemes, often defined as the smallest linguistic pieces with a grammatical function. Consider the word "കാണുകയാണെന്ന്". We can break it into three morphemes: "കാണുക", "ആണ്", and "എന്ന്". "കാണുക" is called the stem, which is a base morpheme to which another morphological piece is attached.

System Description

The functions, dictionaries and suffix lists introduced in the morpheme generator are almost similar to the root extractor. The difference being that the output produced by each functions are list of stripped morphemes rather than a word.

stem

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 that 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 $x_1x_2...x_ty$. An additional column is available in the database which stores the suffix corresponding to X to be inserted in the output list. $x_1x_2...x_ty$ is again returned to the function as input and the process is repeated until no such pattern is found. Finally the last word is inserted into the list with suffixes before returning it as the output.

```
Algorithm(stem)
input: word, slist=list of patterns
output: list of word with no patterns left and suffixes
Steps:
1. Take the input word
2. Check the list of stopwords and return the word if found any.
3. For pattern in slist do
4.   If word ends with pattern do
5.     Replace the pattern with the corresponding replacement string from the database
```

```
6.   Store the suffix in the output list
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: " ് "
  word obtained: മാധ്യമപ്രവർത്തകര്
  list obtained: ["ഉം"]
  Re-initialized Input: മാധ്യമപ്രവർത്തകര്
  pattern found : "ര്"
  Corresponding Replacement String: "ർ"
  word obtained: മാധ്യമപ്രവർത്തകർ
  list obtained: [" "]
Output list: [മാധ്യമപ്രവർത്തകർ , ഉം]

verb

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 and return the list of verb and suffixes.

```
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.     Insert the corresponding suffix into output list
6.     If the word obtained is in the verb dictionary, return the output list
```

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

Example2

Input Word: പകർന്നു
Process:
    Input: പകർന്നു
    pattern found: "ർന്നു"
    Corresponding Replacement String: "രുക"
    Corresponding suffix: "ന്നു"
    word obtained: പകരുക
Output list: [പകരുക, ന്നു]

change

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 "ഒ" & "േ◌ാ" to "ഓ".

dvithva

Another function *dvithva* checks the input word for any string in the list ["ക്ക","പ്പ","ത്ത", "ശ്ശ","ച്ച"]. It takes 'dvithva sandhi', which occurs when the first part in a morpheme doubles when joined to another word, into consideration. 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 $[x_1x_2...x_{t-1}, x_{t+1}...x_n]$.

Algorithm(*dvithva*)

```
Input: word, checklist=["ക്ക","പ്പ","ത്ത", "ശ്ശ","ച്ച"]
Output: list of splitted morphemes
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], word[N+2:]]
```

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 list: [എത്തി, കഴിഞ്ഞു]

sp

The function *sp* 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. *verb* is also employed to confirm that the function can efficiently handle verbs as well. The list of suffixes and final word is returned as output

```
Algorithm(sp)
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 "ല്ല" , 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.       insert change+$x_{t+1}...x_n$ into output list
10.        if reword in dictionary, insert it into output list and return it as output
10.        else do
11.          take stem(reword) and if non-empty, add suffixes obtained into output list
12.          take dvithva(reword) and if non-empty, add suffixes obtained into output list
13.         Re-initialize the word and go to step 1
14.   else do
15.     if word in dictionary, insert it into output list
16.     else take verb(word) and if non-empty, insert the obtained word and suffixes into output list
17.    return the output list
```

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

Example4

Input Word: ശേഖരിക്കുന്നതെന്ന

Process:
  Input: ശേഖരിക്കുന്നതെന്ന
  pattern found: "◌െ"
  output list: [എന്ന]
  word obtained after removing suffix: ശേഖരിക്കുന്നത്
  list obtained using *stem*: [ശേഖരിക്കുന്ന,ത്]
  output list: [ത്, എന്ന]
  Reinitialize word: ശേഖരിക്കുന്ന
  Input: ശേഖരിക്കുന്ന
  pattern found: "◌ു"
  output list: [ഉന്ന, ത്, എന്ന]
  word obtained after removing suffix: ശേഖരിക്ക്
  word obtained using *verb*: ശേഖരിക്കുക
Output list: [ശേഖരിക്കുക, ഉന്ന, ത്, എന്ന]

fn

Another function *fn* 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 *sp*. After validation of both, $[x_1x_2...x_t, x_{t+1}...x_n]$ or the output from *sp* after appending $x_{t+1}...x_n$ is returned as the output.

```
Algorithm(fn)
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 $x_{N+1}...x_n$ in suffix list or dictionary do
9.         if $x_1x_2...,x_{N+1}$ in dictionary, return [$x_1x_2...,x_{N+1},x_{N+1}...x_n$]
10.        else if $x_1x_2...,x_N$ in dictionary, return [$x_1x_2...,x_N,x_{N+1}...x_n$]
11.        else if sp($x_1x_2...,x_N$+ “ ◌്” ) not empty, return sp($x_1x_2...,x_N$+ “ ◌്” )+[$x_{N+1}...x_n$]
12.      if verbsuf[0] in suffix list or dictionary do
13.        if $x_1x_2...,x_{N+1}$ in dictionary, return [$x_1x_2...,x_{N+1}$]+verbsuf
14.        else if $x_1x_2...,x_N$ in dictionary, return [$x_1x_2...,x_{N+1}$]+verbsuf
15.        else if sp($x_1x_2...,x_N$+ “ ◌്” ) not empty, return sp($x_1x_2...,x_N$+ “ ◌്” )+verbsuf
```

```
16. if $x_1x_2...,x_tx_{t+1}...x_n$ ends with k do
17.   if $x_1x_2...,x_tx_{t+1}...x_{n-1}$ in dictionary, return [$x_1x_2...,x_tx_{t+1}...x_{n-1}$],$change$(k)
18.   else if $sp$($x_1x_2...,x_tx_{t+1}...x_{n-1}$+" ◌്") is not empty do
19.     return $sp$($x_1x_2...,x_tx_{t+1}...x_{n-1}$+" ◌്")+[$change$(k)]
```

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

Example5

Input Word: പകർന്നുതന്ന
Process:
  Input: പകർന്നുതന്ന
  pattern found: "◌ു"
  list: [പകർന്നു , തന്ന]
  conversion of suffix using *verb*: [തരുക]
  തരുക in dictionary
  conversion of പകർന്നു using *verb*: [പകരുക,ന്നു]
  പകരുക in dictionary
Output list: [പകരുക,ന്നു,തരുക]

ni

ni 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 *sp* and concatenated together 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, $sp(x_1x_2...x_t)$ is obtained and $sp(x_1x_2...x_t)$+[$x_{t+1}...x_n$] is returned as output.

```
Algorithm($ni$)
Input: word=$x_1x_2...x_{t-1}x_tx_{t+1}...x_n$, chlist=["ൽ", "ർ", "ൻ", "ൾ", "ൺ"]
Output: output list
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 $sp(x_1x_2...x_t)$+[$x_{t+1}...x_n$]
```

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

Example6

Input Word: മഴയിൽനിന്ന്
Process:

Input: മഴയിൽനിന്ന്
pattern found: "ൽ"
suffix: നിന്ന്
word obtained after removing suffix: മഴയിൽ
word obtained using *sp*: [മഴ, ഇൽ]
Output list: [മഴ, ഇൽ, നിന്ന്]

finallist

The function obtains two lists. It removes the first element in the second list and add it with the first list to obtain the outputs to be returned.

root

The root module uses all the suffix stripping module to create a final list to be provided to the morphological analyser.
Algorithm
Steps:

```
1. Take the input word
2. Remove special characters and re-initialize the word
2. Stemlist = stem(word)
3. Store the first element in the stemlist in a list split.
4. Derive ni(split[0]) and if its not [split[0]], split=ni(split[0])
5. Derive dvithva(split[0]), split is set as finallist(dvithva(split[0]),split))
6. Derive sp(split[0]), split is set as finallist(sp(split[0]),split))
7. Derive fn(word), split is set as finallist(fn(split[0]),split))
8. Derive verb(word), split is set as finallist(verb(split[0]),split))
9. return finallist(split,stemlist))
```

Summary

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 tests produced an average 92% accuracy for the morpheme generator. A total of 7000 words were tested out of which 6453 results produced accurate results. The updation of dictionary and suffix list seems to increase the overall accuracy rate of the system.

Installation

You may create a virtual environment for installing the package.

```
python -m venv ENV_DIR
source ENV_DIR/bin/activate
pip install morph-gen
```

Otherwise, use

```
pip install --user morph-gen
```

Implementation

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

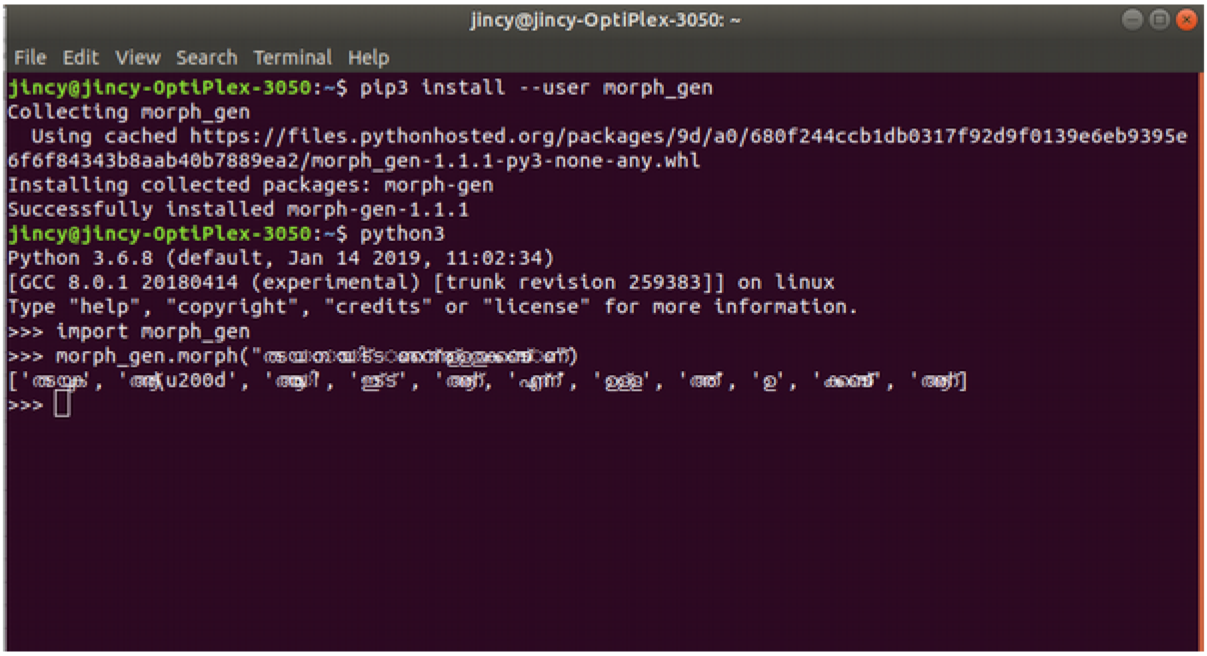
```
import morph_gen
morph_gen.morph(wordi)
```

For example,

```
import morph_gen
morph_gen.morph("മകന്റെയുമാണെന്നാണ്")
```

output:

```
['മകൻ ', 'ന്റെ', 'ഉം', 'ആണ്', 'എന്ന്', 'ആണ്']
```



jincy@jincy-OptiPlex-3050: ~
File Edit View Search Terminal Help

```
jincy@jincy-OptiPlex-3050:~$ pip3 install --user morph_gen
Collecting morph_gen
  Using cached https://files.pythonhosted.org/packages/9d/a0/680f244ccb1db0317f92d9f0139e6eb9395e
6f6f84343b8aab40b7889ea2/morph_gen-1.1.1-py3-none-any.whl
Installing collected packages: morph-gen
Successfully installed morph-gen-1.1.1
jincy@jincy-OptiPlex-3050:~$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import morph_gen
>>> morph_gen.morph("[illegible])
['[illegible]', '[illegible]\u200d', '[illegible]', 'ഇട്ട്', '[illegible]', '[illegible]', 'ഉള്ള', '[illegible]', 'ഉ', '[illegible]', '[illegible]]
>>> 
```