Vriksh: A Tree Based Malayalam Lemmatizer Using Suffix Replacement Dictionary

Dhanya P.M
Dept of Computer Applications, CUSAT, Kochi, India.

Dr. Sreekumar A
Dept of Computer Applications, CUSAT, Kochi, India.

Dr. Jathavedan M
Dept of Computer Applications, CUSAT, Kochi, India.

Abstract – Morphology is a term in Natural Language Processing(NLP) which refers to the different forms a particular word can take. The process of removing the affixes from the word and extracting the stem is called stemming. The lemmatizer generates the root word from the given word rather than reducing it to the stem. Lemmatization is very important in any NLP project since only the root word can contribute to the word count. This explicatory paper presents two methods for extracting the root word from a Malayalam word .One is clustered indexed dictionary based , which uses a suffix replacement method by considering nearly thousand rules which are identified by several test cases done during the development of the project. The second method creates a tree from the set of rules . The permanently stored tree is then searched for a path which matches with the suffix and the corresponding leaf node , which is the replacement, is retrieved. Testing with online Malayalam documents helped in finding out and adding several rules to the dictionary. This is a pioneer lemmatizer which uses a tree based method.

Index Terms – Suffix, Replacement, Prefix, Lemma.

1. INTRODUCTION

Since Artificial Intelligence mainly deals with inducing intelligence in computers so that they can behave like human beings, programming them to understand natural languages like English, Hindi, Malayalam etc. is gaining importance. It is obviously due to the area of artificial intelligence, sub areas like Natural Language Processing, expert systems etc. develop. Interaction of the user with the system using natural language can be either through natural text or natural language speech. Thus text mining has been an important area of research under Natural Language Processing. So text document processing has many sub areas where their accuracies increase only if the words are reduced to their root form which we call as the process of lemmatization. This text mining in any language requires the word to be converted to the root form. Malayalam is a language which is spoken by over 33 million people in the state of Kerala, India. Malayalam words are subjected to morphology to a large extend. A particular word in Malayalam can take more than 100 affixes and same is discussed in section 3. Due to the complexity of the words, lemmatization is very much required and very difficult in the case of a language like Malayalam. Stemmers are already available in languages like English, Arabic, Persian etc. Many stemming algorithms have been developed in various Indian Languages also. A lemmatizer [1] is developed in Malayalam which is a three pass method.

2. RELATED WORK

Spanish version of Porter Stemmer is used as preprocessing tool for correctly extracting the text in each document image [2]. This helps in forming the bag-of-words vector. The performance of various page classifiers are being compared for all of which stemming is an important module which affects the accuracy of the system. The method [3] has considered stem and suffixes for word segmentation in Mongolian language. Application of predefined pattern to derive the stem words are discussed in the paper authored by [4]. The presence of affixes results in a large vocabulary in any language. They discussed rule based, direct, interlingua, transfer, statistical, example based, knowledge based and hybrid methods of translation where stem and affixes play a vital role. The document similarity calculations like tree based, time series, vector based and different text clustering methods like hierarchical, partitioning, combination and multilevel clustering mentioned in [5] requires the word to be stemmed. Stemming, the removal of affixes, is used as the preprocessing task for the calculation of term frequency and is used in Map Reduce framework for text summarization as in [6]. In [7], the authors mention about factorizing words in to morphological components which requires the stem to be extracted.

There are some stemmers developed in Asian languages like Arabic [8], which removed both prefix and suffix of given word. Since prefix words are very rare in Malayalam, the algorithm we developed has considered suffixes only. The Indonesian Stemmer [9], is based on finite state automata. A best word candidate is selected from a candidate list. A detailed

survey on stemming is done in [10] referring to many languages around the world like English, Czech, Bulgarian, Turkish, Greek, German, Dutch, Portuguese, French etc. Some methods discuss about integrating stemming rules [11] and have considered both suffixes and prefixes.

The stemmers developed in Indian languages are surveyed by [12]. The Gujarathi stemmer [13] and the one developed in Bengali [14] uses a rule based approach. Stemming has reduced the number of unique words and they have considered both understemming and overstemming. In [15], they use minimum stem set model of stemming which gives an accuracy of 80% - 88%. They have tested for languages like Hindi, Marathi, Malayalam and English. The method has an input word list and an input suffix list. There are methods which uses a probabilistic approach[16] using a suffix list. An Urdu stemmer is developed by [17].

A stemmer in Hindi language was developed by [18] .The stemmer developed by [19] , stems only nouns and proper names with the help of a dictionary. One among 19 conditions is applied for an input Punjabi word and the result is checked against a Punjabi Name list. The method suggested in [20], show how word sense disambiguation is benefited from stemming. Here testing is done with the help of a sense list created from Hindi Wordnet. In word stemming implemented with the help of hashing[21], the authors have considered nominal inflections, prenominal inflections and verb inflections. The rule based stemmer in Hindi [22] makes use of a Hindi WorldNet. The Tamil stemmer [23] is clustering stemmed words using k-means to improve the IR system and the one developed by [24], deals with Tamil suffixes only.

In Malayalam stemmer [25], stemming is done using Finite State Automata. FSA is modeled using all possible suffixes and have considered singular nouns, plural nouns and verbs. In [26] the authors used a method of recursive suffix stripping with an accuracy of 83.67%. If a word has three suffixes then it is processed thrice. The system is a sub module of a sentence parser and the accuracy rely on the dictionaries being used. In morphological analyzer developed by [27], rather than root extraction, morphological features like noun/verb, number: plural/singular and tense: past/present/future are also extracted.

In the light weight stemmer developed by [28], they developed a suffix dictionary and used the suffix stripping algorithm to get the stem. Here the word after stripping is taken as the output which is not the actual root word. In the method developed by [1], they used a three pass algorithm for a lemmatizer, while the proposed method uses a single pass algorithm. The system is used as a sub module of question answering system. In the method suggested by [29], a rule list is provided separately for nouns and verbs and gives an accuracy of only 60%. A comparison of stemming methods developed in Indian languages is shown in Table I.

TABLE 1. Comparison of Stemmers in Indian Languages

AUTHOR, YEAR	LANGUAGE	METHOD	TESTING & ACCURACY
Prajitha U et.al,2013	Malayalam	One pass Suffix stripping Used Suffix Dictionary Scanning from right to left for the longest match .	86%
Prajisha K, 2013	Malayalam	Three pass Suffix Stripping Rule based system	97% Testing done with news artciles in the web
Vinod P.M et.al, 2012	Malayalam	Morphological analyzer Recursive suffix stripping Uses lexical dictionary, Monolingual dictionary and Bilingual dictionary .	Testing with words taken from Malayalam dictionary 83.67%
Jisha P .J et.al, 2011	Malayalam	Morphological analyzer Uses Bilingual dictionary and Root dictionary .	Not specified
Vijay S R et.al, 2010	Malayalam	Finite state automata Next state is determined using Morphotactic rules	Testing done with Online newspaper 94.76%
Jikitsha S,Bankim P, 2014	Gujarathi	Substitution rules	Evaluation based on EMILLE corpus 92.41%
Das S,Mitra.P, 2010	Bengali	Using hash table containing suffixes of nouns,verbs considers derivational inflectional words	Tested using FIRE 2010 dataset Recall of 96.27 %
Vasudevan N,Pushpak B, 2012	Hindi	Semi supervised Stemming by weighted minimum stem set	84%
Ramachandran V.A,Illango K 2012	Tamil	Iterative Suffix Stripping	84.79%
Kasthuri M ,Britto R.K 2014	Tamil	Prefix, Question, Conjunction, Case Plural, Imperative Tense suffixes are stripped	Testing with docs from net

The various langauges mentioned here are spoken in different states in India. They are developed considering the features of their language.

3. MORPHOLOGY IN MALAYALAM

Since Malayalam language is rich in morphology, the lemmatizer developed here employs a total of 1121 suffix replacement rules. Morphology in Malayalam language is so complex that a single word can take many different forms.

Difficulty in developing a lemmatizer is due to this language complexity. We can show this complexity by taking as example a simple word (mavu) which means mango tree in English. Some of the morphological forms of the mentioned word are shown in Fig. 1.

Figure 1 Morphology of word ‘mavu’

The word (mavu) which is a noun can be attached with one preposition as (mavil) , (mavulla), (mavinte), (mavo), (mavaya),(mavakkiya), (mavennal), (mavundu), (mavilla), (mavum), (mavanu), (mavinu), (mavalle) , (maville) etc. The same word can come with two suffixes as in the words (mavilninnu), (mavillengil) , (mavayathinal), (mavayirikkum), (mavanullathu), (mavupayogichu), (mavileykkanu), (mavundengil), (mavanithu) , (mavilekkayi), (mavillathayi), (mavilottu), (mavukalum) ,(mavinoduchernnu) etc. The word mavu can come with three affixes as the cases (mavilninnukondu), (mavukalumayi), (mavukalumanu) etc and also with four affixes as in the following cases (mavundayirunnappol), (mavilekkayirikkum). Similar inflections happens in the case of verbs also.For example the verb (varuka) can take one affix as in (vannal) , (vannilla), (vannittu) etc . The verb can take two affixes as in the case (vannennal), (vannittundu), (vannittanu), (vannittilla) etc.

The suffixes used in Malayalam language can be divided in to singular suffixes, binary suffixes, triple suffixes and suffixes with more than three parts. They can be categorized as following.

3.1 Singular Suffixes

- Suffixes which end with 'chillu' - 'ththil', 'kal', 'ral', 'mbol', 'ngalil', 'mengil', 'ppol', 'uppol' etc are examples which belong to this category.
- Suffixes which end with 'chandrakkala'. Some examples of such category include 'rathu' , 'manu', 'ru', 'rodu', 'rkku', 'rkkai', 'arundu', 'rnnu', 'mennu', 'nnathu', 'ttathu'.
- Suffixes which end with 'u , e, ea '- 'rathu' , 'chchu' , 'ththe' , 'ththinte' , 'kale' , 'ththode', 'loode' , 'katte' , 've', 'nude', 'yalle' etc belong to this category.
- Suffixes which end with 'i' – 'kumayi', 'koodi' , 'dakki', 'kkayi', 'ippoyi', 'raadi' etc are few in this category.
- Suffixes which end with 'anunasika' – 'kum', 'makaam' , 'ththinum', 'yolam', 'kalum', 'injnjum' .
- Suffixes which end with 'a, o' – Some of the cases are 'maya' , 'malla', 'killa' , 'ninna' , 'unna' , 'kulla' , 'mo' , 'yallo'.

3.2 Binary Suffixes

These suffixes consist of two singular suffixes joined together. They can be further classified as the following.

- Suffixes which end with 'chillu' – Some cases include 'yennal, 'unnappol', 'rumbol' etc.
- Suffixes which end with 'chandrakkala' – 'ththileykku', 'mbaththekku', 'yumanu', 'mbozhanu', 'ngalkku', 'yittullathu', 'ninnanu', 'kondathinu', 'mayundu', 'nullathu' , 'dunnathu', lekkulathu ', mayittu ', 'nathinu' nilekkanu', 'ilekkullathu', 'yilninnu', 'lanithu', 'ththilumundu'.
- Suffixes which end with 'u, e, ea' - 'rnnathinu', 'kkappedunnu', 'kalpole', 'lanivide', 'njnjathalle' .its, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Suffixes which end with 'i'- 'ththodukoodi', ' lumundayi', 'marundaayi' , 'ththilekkayi' , 'yallathaayi'.
- Suffixes which end 'a' – This include 'rumaayirunna' , 'mundaayirunna', 'ndndakkiya', 'ndndavunna', 'ndndakavunna', 'yanundavuka', 'naduththulla' , 'kalkkalla' , 'kallulla' , 'mallaththa' .

3.3 Triple Suffixes

These suffixes consist of three suffixes joined together. They can be further classified in to the following .

- Suffixes which end with 'chillu'. They include 'ndndakkiyennal', 'ndndavukayennal', 'ngngittillengil' , 'markadiyil' , 'markidayil', 'ndndayirunnappol' .

- Suffixes which end with 'i'. - include 'mundayirunnathayi', 'inodoppamayi'.
- Suffixes which end with 'chandrakkala' – 'kalilonnanu' , 'kalolottallaththathu', 'kalilottallaththathanu' , 'iyappozhanu' .
- Suffixes which end with 'ukaram' – include 'kkumaayirunnu', 'umundaayirunnu' , 'ththilekkayirunnu'

3.4 More Than Three Suffixes

Here more than three suffixes are joined together to form a single suffix. They include 'lokkeyundakarundengil', 'kalilottallaththathanennanu', 'ilekkayirikkum', 'millaththathukondulla' etc.

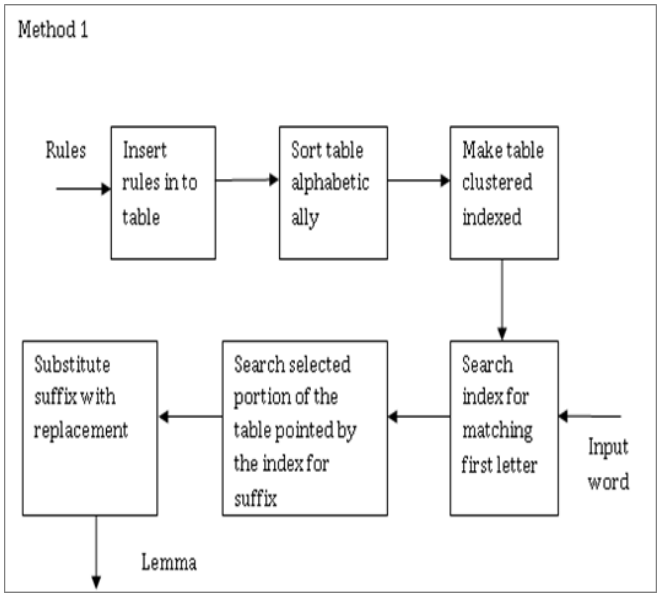

Figure 2 Architecture of Dictionary based method

4. SYSTEM ARCHITECTURE

Here two methods are proposed, method1 - which is a dictionary based and method2 - which is suffix tree based. The suffix - replacement dictionary which is developed in method1 is used to create the suffix tree. Both the methods are illustrated in the Fig. 2 and Fig. 3. The concept of suffix tree is slightly modified. In actual definition of suffix tree, except for the root, every internal node has at least two children and each edge is labeled with a non-empty substring of the search word S.

The tree implemented here does not pose any such restrictions. Here a node can have empty substring of S. The tree is being pickled in order to make it a permanent storage. Tree pickling is a concept in python programming where by permanent data structures can be created.

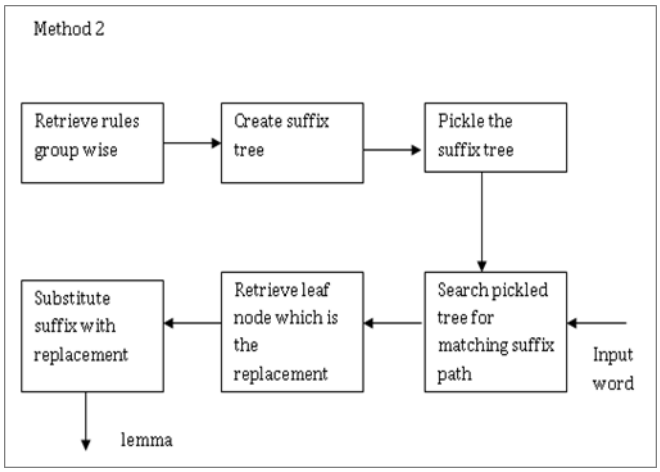

Figure 3 Architecture of Tree based method

5. METHODOLOGY

5.1 Dictionary Based Method

The method proposed here uses a suffix replacement methodology where it creates a Malayalam dictionary of suffixes and replacements .This method follows a lookup table approach as in English stemmers, but the difference is that, in latter the lookup table is maintained only for some exceptional cases. Here the suffix is not stripped, but the suffix tself get replaced with another. Eg:- (avalude - aval), (vannupoyi-varuka), (kazhukum-kazhukuka), (pokanayirikkum-pokuka), (thengilninnanu-thengu). The Malayalam suffix-replacement dictionary is being developed using MYSQL. The table is permanently sorted based on the alphabetical order of the suffix as we can see in Fig. 5.

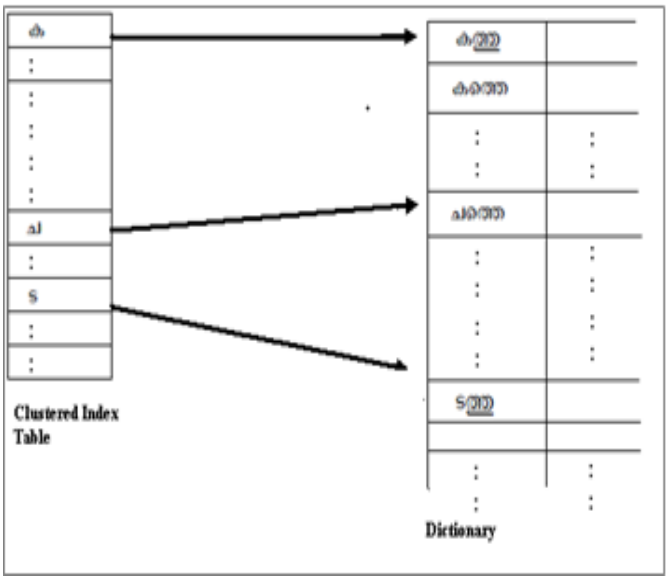

Figure 4 Indexed Table

Since the table is a large one, in order to reduce the searching time, it is clustered indexed as shown in Fig. 4. The clustered index is created based on the first letter of suffixes. The search is first directed to the index table which consists of the unique first letter of the suffixes and pointers to the first occurrence of the suffix starting with that unique first letter in the dictionary. The largest suffix from the word is found out and is substituted with the replacement in the dictionary.

1) Algorithm 1

Input: Input document and suffix -replacement dictionary which is clustered indexed.

Output: lemma

Read the text and tokenize the sentences.

Remove stop words. i = 0.

For each token tk **do**

Repeat

ch = tk[i]. Search the clustered index for the entry ch.

If found then

Search the portion of the table pointed by the pointer for the suffix starting with ch.

If found **then**

Retrieve the replacement from the table corresponding to that suffix.

Replace the suffix with the replacement.

Display the lemma and exit from loop.

Else

i = i + 1

Else

Display the token unchanged.

Until i<n

For a given input word (token) the identification of the suffix starts from the first letter itself and only the portion of the table consisting of the suffixes starting with that letter is being searched. This limitation is imposed due to the presence of the clustered index. If a suffix could not be found in that area, the algorithm takes the next letter of the token and searches the table in the limited area containing suffixes starting with that letter. Once a proper suffix is found, it is replaced with the corresponding replacement.

Suffix	Replacement	Suffix	Replacement
*കാവൽ	*കക	കട്ടുള്ള	NULL
*വരം	*വര്	കപ്പെടുന്ന	കക
കണ്ണും	കാണക	കള്ളം	NULL
കറ്റ	കം	കള്ളമല്ലാം	NULL
കരണ	കം	കാകട്ട	ക്ക്
കരണക്ക്	കം	കാണുള്ളത്	ക്ക്
കരണത്ത്	കം	കാനം	കക
കത്ത്	കം	കാൻ	കക
കന്നാ	കുക	കായി	NULL
കന്റ്	കൻ	കായുണ്	NULL
കളാണ്	ആണ്	കായ്	NULL
കളിലൊന്നാണ്	NULL	കിനം	ക്ക്
കളിലേക്കാണ്	NULL	കിനെ	ക്ക്
കളിലേക്ക്	NULL	കിലുണ്ടാകാനുള്ള	NULL
കളിലൊട്ടു	NULL	കുണ്ടായിണ്ട	ക്ക്
കളിലൊട്ടുണ്ടാതാണെന്നാണ്	NULL	കുണ്ടായിണ്ടായി	ക്ക്
കളിലൊട്ടുണ്ടാതാണ്	NULL	കുന	കക
കളിലൊട്ടുണ്ടാതത്	NULL	കുനാണ്	കക
കളിൽ	NULL	കുപയോഗിച്ച്	ക്ക്
കളും	NULL	കുമായി	ക്ക്
കളുണ്	NULL	കുമുണ്ടാകില്ല	NULL
കളുമല്ലാം	null	കുമുണ്ടായിണ്ട	ക്ക്
കളെ	null	കുള്ള	ക്ക്
കൾ	NULL	കുള്ള	null
കൾക്കടിയിൽ	NULL	കൊ	ക്ക്
കൾക്കല്ല	NULL	കൊടെ	ക്ക്
കൾക്കിടയിൽ	NULL	കൊള്ളം	ക്ക്
കൾക്കം	NULL	കൂടാതെക്	കൂം
കൾപോലെ	NULL	കൂടാത്തത്	കൂം
കുണ്ടില്ല	കുണ്	ക്കുറ	ക്കം
കാകട്ട	ക്	ക്കാന	ക്കം
കാട്ടിലൂടെ	കാട്	ക്കാത്	ക്കം
കാട്ടിലേക്കാണ്	കാട്	ഗതിനെ	ഗം
കാട്ടിലേക്ക്	ട്	ഗുറ	ഗം
കായ	ക്	ഗത്	ഗം
കായി	ക്	ഗാകട്ട	ഗ
കാറായ	കാർ	ഗിനം	ഗ
കാവുണ്	കക	ഗിലേക്കായി	ഗ
കിനം	ക്	ഗിലേക്കായിണ്ട	ഗ
കിന്റെ	ക്	ഗുമാണ്	ഗ
കിപ്പോയി	കക	ഗൊള്ളം	ഗ
കിലൂടെ	ക്	ഗാകട്ട	ഗ
കിലെ	ക്	ഗാക്കിയ	ഗ
കിലേക്കാണ്	ക്	ഗാണിത്	ഗ
കിലേക്കായി	ക്	ഗായ	ഗ
കിലേക്കായിണ്ട	ക്	ഗായി	ഗ
കിലേക്ക്	ക്	ഗിനട്ടുള്ള	ഗ
കിലേല്ല്	ക്	ഗിനം	ഗ
കിൽ	ക്	ഗിനെ	ഗ
കിൽനിന്നാണ്	ക്	ഗിന്റെ	ഗ
കിൽനിന്നാകൊണ്ട്	ക്	ഗിലൂടെ	ഗ
കിൽനിന്ന്	ക്	ഗിലെ	ഗ
കില്ല	കക	ഗിലേക്കാണ്	ഗ
കം	ക്	ഗിലേക്കായിണ്ട	ഗ
കാണ്	ക്	ഗിലേക്ക്	ഗ
കായി	ക്	ഗിലേല്ല്	ഗ
കായിണ്ട	കക	ഗിൽ	ഗ
കള്ള	ക്	ഗിൽനിന്നാണ്	ഗ
ങ്കടി	ംം	ഗിൽനിന്നാണ്	ഗ
കേട്ടും	കേൾക്കക	ഗിൽനിന്നാകൊണ്ട്	ഗ
കേട്ടപ്പോഴും	കേൾക്കക	ഗിൽനിന്ന്	ഗ
കൊട്	ക്	ഗില്ല	ഗ

Figure 5 Dictionary

5.2 Tree Based Method

In this method a tree is constructed from the set of rules available in the database. Here the rules are retrieved group wise, where one group consists of rules which start with the same first letter. The tree is being pickled using the pickling technique of python which makes the tree a permanent structure. This permanent tree is being queried in search of the

suffix. The root node of the tree is head node without storing any identifier.

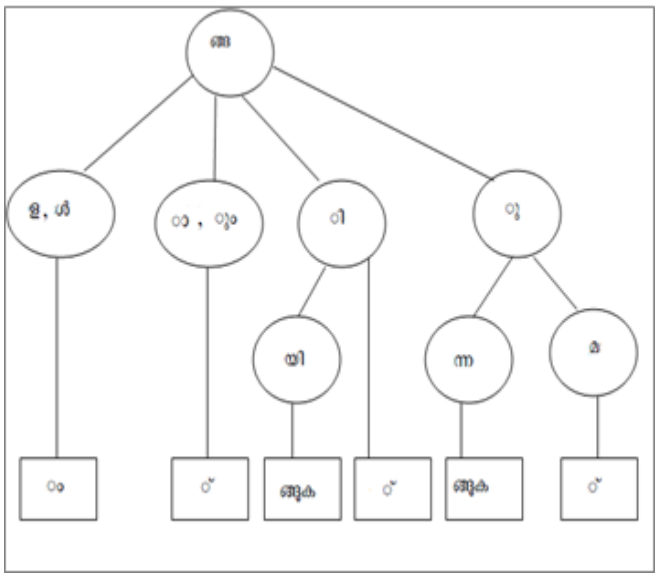

Figure 6 Tree created

The first level children are unique first letter of the suffixes. The second level nodes are formed by taking the unique prefixes of the suffixes in the dictionary after removing the first letter. The third level nodes are constructed by taking the rest of the suffix after removing the prefix from it. The leaf nodes of the tree are the replacements. The method passes through two phases where the first phase is dealing with tree creation and tree pickling. The second phase is dealing with searching the tree for a matching suffix path and replacing the matched string in the word with the leaf node of that path. The search starts with the first letter of the word. The letter is compared with the identifier of the nodes in the first level children. Once a match is found, the letter is removed from the word and unique prefixes of the resultant string are taken. The prefixes are now compared with the second level children of the identified path. Once a match is found up to third level, the leaf node is taken as the replacement of the suffix. If at any level there is a mismatch, we do not backtrack to the previous level, but advance the pointer in the word to the second letter and do the above process. The back tracking is avoided by the careful selection of prefixes in the second level node formation. All the prefixes selected are unique and this reduces the chance of backtracking to nil. Execution time of both tree based method and dictionary based method is calculated for 100 Malayalam words from Online documents and is shown in the Fig. 15. Tree based method shows an average execution time of 0.00073s and the dictionary based shows an average execution time of 0.0084s. Tree has reduced the searching time very much since we need to search only a portion of the tree under a particular start letter of the suffix. A portion of the tree created is shown in the Fig. 6.

5.3 Algorithm 2: Tree Creation

Input -Rules from the database.

Output: Tree

Read the rules of the form S-R group wise from the suffix replacement dictionary

SF_i = unique first letter of suffixes

where i = 1 to n ; n is the number of unique prefixes

Create root node R_n

Create first level child nodes of the tree with SF_i

For each group i do take the suffixes S_j

where m is the no of suffixes in a group $(S- R)_i$

$RS_j = S_j - SF_i$

PR_k = Unique prefixes of RS_j

where k = 1 to p and p is the no of unique prefixes in that group

Create second level child nodes with PR_k

$RPR = RS_j - PR_k$

Create third level child nodes with RPR

Create leaf nodes R_l which corresponds to the replacement in S-R rule, where l = 1 to r and r is the no of replacements

5.4 Algorithm3: Tree Search (tree, word)

Input – Word to be searched

Output – lemma

C = first letter of W_i where W_i is the i^{th} word of the document.

children = level1nodes of the tree.

For each child in children **do**

Compare child.identifier with C.

If not match

Break

Else

nextnode = child

$CW_i = W_i - C$

children = childnodes of nextnode

find prefixes of CW_i

For child in children **do**

Compare prefixes with child.identifier

If not match

Break

Else

nextnode = child

$RW_i = CW_i$ - child.identifier

children = childnodes of nextnode

For child in children do

Compare RW_i with child.identifier

If not match **do**

Call Tree search (tree, CW_i)

Else

Return leafnode

5.5 N- Gram Rules

5.5.1 bi-gram rules

As in Fig. 14, we can see that, 'thinu' can't be taken as a suffix, as it can come with many other affixes and can create specific rules. It can be converted to a bigram rule by considering one more letter on the left side and the same is shown in the entries one and two in Fig. 14. 'thinu' along with the letter 'na' and 'nja' form two different suffixes 'nathinu' and 'njathinu' and get replaced with two different replacements 'zhuka' and 'yuka'. Similarly the 'ttathinu' , 'yathinu', 'thathinu' are also bigram suffixes which are formed by considering one more letter('tta', 'ya', 'tha' respectively) on the left side of 'thinu'. They get converted to the replacements like 'kkuka', 'vuka', 'um' respectively.

5.5.2. Tri-gram rules

Suffix	Replacement
ാനോളം	നം
നോളം	ങ്
വരോളം	ര്
ളോളം	ശ്
ക്കോളം	ക്ക്
പ്പോളം	പ്പ്
രോളം	ര്
ഴോളം	ഴ്
ഗോളം	ഗ്
ച്ചോളം	ച്ച്
ങ്ങോളം	ങ്ങ്
ത്തോളം	ത്ത്
മ്പോളം	മ്പ്
ടോളം	ട്
തോളം	ത്
ശോളം	ശ്
സ്സോളം	സ്സ്

Figure 7 Suffixes which ends with 'anunasika'

Trigram rules are formed by considering two more letters to the left of the actual suffix word. The entries 3, 4 , 6, 8 and 11 in Fig .14 are examples of such trigram rules. As sample rule we can take (ttiyathinu - ttuka) where two letters 'tti' and 'ya' to the left of 'thinu' is also considered for creating the rule. The third entry in Fig. 7 (varolam - r) is also an example of trigram rule where the suffix is formed by adding two letters 'va' and 'r' along with the actual suffix 'olam'. All the entries in Fig .8 and Fig .9 are examples of n-gram rules where more than 3 letters to the left of actual suffix is considered for the creation of the L.H.S of the rules.

Suffix	Replacement
ളോടുള്ളത്	ൾ
രോടുള്ളത്	ർ
യോടുള്ളത്	NULL
നോടുള്ളത്	ൻ
ോടുള്ളത്	്

Figure 8 Suffixes which ends with 'chandrakkala'

Suffix	Replacement
കളിലോട്ടല്ലാത്തതാണ്	NULL
ങ്ങളിലോട്ടല്ലാത്തതാണ്	ാം
ുകളിലോട്ടല്ലാത്തതാണ്	്
കളിലോട്ടല്ലാത്തതാണെന്നാണ്	NULL
ങ്ങളിലോട്ടല്ലാത്തതാണെന്നാണ്	ാം
ുകളിലോട്ടല്ലാത്തതാണെന്നാണ്	്

Figure 9 Suffixes which ends with ‘anu’

6. COMPARISON WITH EXISTING METHODS

Suffix tree method is the one used in human genome project, where a tree of suffixes is created in which every node has a single letter of the substring to be searched, but here the nodes in level two and level three can have more than a single letter as the node identifier. The node in level one has only single letter as the node identifier, which in this case is the starting letter of the suffix. As mentioned earlier, the tree created in the proposed method avoids backtracking with a careful selection of prefixes. Initially common prefix with maximum length is found out and the words with these prefixes are removed from the list. From the remaining list , again common prefix with maximum length is found out. This process continues until there are no common prefixes.

The method [28] has mentioned about the suffix 'yolam', the proposed method has a related similar set which consists of the rules given in Fig. 7. While the lalitha stemmer has given the suffix 'kalodullathu', few similar suffixes identified by the lemmatizer are given in Fig. 8. While the lalitha stemmer has the implemented the suffix, 'kalilottallathathanennanu' the proposed method identified some related rules as given in Fig. 9. Lalitha stemmer uses a suffix dictionary and the proposed method used a suffix - replacement dictionary which is permanently alphabetically sorted and clustered indexed, followed by a tree created out of it. The method suggested by [28] searches for the suffix from right to left and as mentioned in their paper considers only stemming where word 'angangal' get converted to 'anga' , but our method searches from left to right and generates the output 'anagam' which is lemma, the real meaningful word. The suffixes identified in the words 'odunnu', 'odum', 'odumbhol', 'odarundu', 'odan' , 'odiyappol' , 'odippoyi' , 'odimari' , 'odivannu', 'odikkondu', 'odikkondirunnu' etc get substituted with the replacement 'duka' and result in the word 'oduka' which is the correct lemma, but most of the Malayalam stemmers give the output 'odu' which is only the stem. The morphological analyzer developed by [27] convert the input word 'varum' to the output 'varu', but our method gives the correct output 'varuka'. The method suggested by [26] gives the output 'odu' for the input word 'odikkondirikkukayayirunnu', but our method gives the output 'oduka' which is the correct meaningful word. Comparison with Indic stemmer can be done with the following cases and is shown in Table II.

TABLE 2. Few Results of Vriksh Stemmer

Cases	Results of Vriksh Stemmer
Case 1	The rule (rum – r) in [29], stem the words (avarum -avar), (palarum - palar) correctly but stem the words wrongly as (varum - var), (theerum – theer), (chaarum - chaar) , (tharum – thar). The results correctly obtained in the new method and the addition of the extra rules is shown in Fig. 10. This will take the largest matching suffix and substitute with its replacement.
Case 2	Online stemmer has given only the rule (thilekku-um) , but the new lemmatizer has considered additional rules as in Fig. 11.
Case 3	While the indic stemmer has considered only the rule (mundayirunnathayi - um + undu), the new method has identified many additional rules as shown in Fig. 12.
Case 4	Many stemmers developed so far generalise the rules which lead to many stemming errors. For eg:- (akaam – um), but is true only for certain words like (pakamakaam – pakam). We can see that the rules fail in the case of the words like (avaralakaam), (mazhayakaam), (avalakaam), (vayasakaam) etc. Some of the new suffixes identified by this method to sort out this issue are given in Fig. 13.
Case 5	Let us consider some words like (veenathinu), (marinjathinu), (thottathinu), (pettathinu), (kattathinu), (njettiyathinu), (poyathinu), (pottiyathinu), (charithrathinu), (ponnathinu). A generalised rule can't be used here as they get converted to their root form in different ways. The solution is given in Fig. 14. The fourth, eighth, last two entries in Fig. 10 and last four entries in Fig. 11 are exceptions where the suffix and replacement is the entire word.
Case 6	The rules (nanu - n + anu), (nalla - n + alla), (nilla - n + illa) are some of the rules in indic stemmer which again creates the stop word 'alla', which need to be removed again.The present method modified the rules as (nanu - n), (nalla - n), (nilla – n), where by it eliminates the creation of stop words.Similar is the case with the (lanu - l + anu), (lalla = l + alla), (lilla - l + illa). Since the above rules create stop words 'anu', 'alla', 'illa', they are modified as (lanu – l), (lalla - l), (lilla - l) .
Case 7	The rules (ykanayi - ykan + avuka) is actually creating a word which need to be stemmed again, so we have modified the replacement as (ykuka) and the rule (kanayi - kan + avuka) is also modified as the (kanayi – kuka) which gives the real meaningful lemma.
Case 8	Unwanted stop words are again being created by the rules mentioned in online stemmer which can be seen in the following ones (yaaniva = anu + iva), (yullava - ulla + ava), (yullathu - ulla + athu), (yallo – none + allo). The proposed method has created new rules which replaces all these rules with null as R.H.S .
Case 9	The indic stemmer developed by [29] has applied stemming rules separately for noun/verb. He has given the rule (rilla - r + illa). This is true only for nouns ie, (avarilla - avar + illa). In case of verb (kavarilla), the stemmed output should be (kavaruka + illa). The new method takes in to consideration a lot of such stemming errors. Similarly the indic stemmer has given the rules (nil – n), (ril-r), (yil- null), (lil-l), but the new method has considered a lot of relevant rules like (kil - k), (ngil - ng), (chil - ch), (dil-d), (nnil – nn), (thil - th) and specifically (mil - m)(Eg:- assamil - assam) which is not addressed in indic stemmer.

Cases were developed from the incorrect results shown by indic stemmer and the new vriksh stemmer has corrected it with bigram, trigram and ngram rules.

Input word	lemma	rules
അവരും	അവർ	*വരും -->*വർ
വരും	വരുക	രും --> രുക
തീരും	തീരുക	രും --> രുക
പലരും	പലർ	പലരും --> പലർ
പോരും	പോരുക	രും --> രുക
ചാരും	ചാരുക	രും --> രുക
തരും	തരുക	രും --> രുക
ചിലരും	ചിലർ	ചിലരും --> ചിലർ
കലരും	കലരുക	രും --> രുക
കവരും	കവരുക	കവരും -->കവരുക
മലരും	മലർ	മലരും --> മലർ

Figure 10 Rules

Suffix	Replacement
യിലേയ്ക്ക്	NULL
നിലേയ്ക്ക്	ൻ
കിലേയ്ക്ക്	ക്
ഗിലേയ്ക്ക്	ഗ്
രിലേയ്ക്ക്	ർ
ളിലേയ്ക്ക്	ൾ
മിലേയ്ക്ക്	മ്
ചിലേയ്ക്ക്	ച്
ങ്ങിലേയ്ക്ക്	ങ്
പാട്ടിലേയ്ക്ക്	പാട്
കാട്ടിലേയ്ക്ക്	കാട്
വീട്ടിലേയ്ക്ക്	വീട്
കൂട്ടിലേയ്ക്ക്	കൂട്

Figure 11 Suffixes which ends with 'leykku'

Suffix	Replacement
ലുണ്ടായിരുന്നതായി	ൽ
രുണ്ടായിരുന്നതായി	ർ
ക്കുണ്ടായിരുന്നതായി	ക്ക്
ച്ചുണ്ടായിരുന്നതായി	ച്
ടുണ്ടായിരുന്നതായി	ട്
റ്റുണ്ടായിരുന്നതായി	റ്റ്
ണ്ണുണ്ടായിരുന്നതായി	ണ്ണ്
തുണ്ടായിരുന്നതായി	ത്
നുണ്ടായിരുന്നതായി	ൻ
റുണ്ടായിരുന്നതായി	ർ
ളുണ്ടായിരുന്നതായി	ൾ

Figure 12 Suffixes which ends with 'ndaayirunnathaayi'

Figure 13 List of suffixes

Input word	lemma	Rule
വീണതിന്	വീഴുക	ണതിന് --> ഴുക
മറിഞ്ഞതിന്	മറിയുക	ഞ്ഞതിന് --> യുക
തൊട്ടതിന്	തൊടുക	ൊട്ടതിന് --> ൊടുക
പെട്ടതിന്	പെടുക	ൊട്ടതിന് --> ൊടുക
കട്ടതിന്	കക്കുക	ട്ടതിന് --> ക്കുക
ഞെട്ടിയതിന്	ഞെട്ടുക	ട്ടിയതിന് --> ട്ടുക
പോയതിന്	പോവുക	യതിന് --> വുക
പൊട്ടിയതിന്	പൊട്ടുക	ട്ടിയതിന് --> ട്ടുക
ചരിത്രത്തിന്	ചരിത്രം	ത്തിന് --> ം
പോന്നതിന്	പോരുക	ന്നതിന് --> രുക
പോണതിന്	പോവുക	ോണതിന് --> ോവുക

Figure 14 lemma

7. DATA SET

Online Malayalam documents from various domains are considered for the testing the performance of the system developed . The documents selected are of varying sizes as mentioned in section 8. The results obtained from the vriksh stemmer and the target system is being manually checked by Malayalam speaking natives . Repeated study of the Malayalam documents revealed that , only 50 % of the words come with suffixes and so need to be lemmatized. Out of that , majority of the words end with 'chandrakkala'. The words which end with 'anunasika' are less compared to the total number of words to be stemmed. Statistics of distribution of various category of words are shown in the Table III.

TABLE 3. Distribution of words in Malayalam documents

Malayalam words Categories	Average distribution of words
Words to be lemmatized	50 % of N
Words which ends with 'chandrakkala '	30 % of L
Words which ends with ' aekaaram'	20 % of L
Words which ends with ' akaaram'	20 % of L
Words which ends with ' ikaaram'	13 % of L
Words which ends with ' ukaaram'	6 % of L
Words which ends with ' chillu'	6 % of L
Words which ends with 'anunasika'	6 % of L

N = Total number of words in the document

L = Total number of words to be lemmatized

8. RESULTS AND DISCUSSIONS

Comparison with other Malayalam stemmers can be done by considering the measures like precision, recall and accuracy. According to this project true positives, false positives , true negatives and false negatives are defined as follows.

t_p = no of words correctly lemmatized (1)

f_p = no of words mistakenly lemmatized (2)

t_n = no of words correctly not lemmatized (3)

f_n = no of words mistakenly not lemmatized (4)

For comparison online Malayalam documents were taken as the dataset . All together documents of varying lengths are considered. Here $t_p + t_n$ forms the correct result and $f_p + f_n$ forms the incorrect result. The results obtained can be tabulated as in Table IV. The documents are tested with dictionary based, Tree based and Indic stemmer.The dictionary based and tree based gives the same result as the second is developed from the first. The difference lie in the execution time which is shown in Fig 15. The testing was done on a system with memory – 1.9 Gb, Processor – Intel Core 2 Duo CPU T6570@2.10 Ghz x 2 , OS type 32 bit Ubuntu 11.10 . Accuracy , precision and recall, F1 score measures are calculated and tabulated in Table V.

There are some words which gave incorrect results in the proposed method .The addition of rule for correcting those words may result in the incorrect result of other words. So such words can be considered as the exceptional cases and should be dealt separately. The only solution is to consider the entire word as the suffix and the expected lemma as the replacement. Some of the examples are (doore - door), (mylilere – myliler). Here the false positives and false negatives shows the incorrect results which are given by both the methods.

TABLE 4 Comparison of Results with Indic stemmer

	Vriksh Stemmer		Indic Stemmer	
No of words in the dataset	Correct words	%	Correct words	%
500	475	95	241	48
1000	821	82	563	56
2000	1706	85	782	39
2500	2025	81	1725	69
3000	2582	86	2040	68
5000	4102	82	3550	71
7500	6154	82	4651	63
10000	8608	86	4659	63
15000	14,793	98	6301	68
20000	17609	88	12000	60

Source: Indic stemmer is available online for testing. The correct results produced by both the stemmers are manually checked with the help of Malayalam speaking natives.

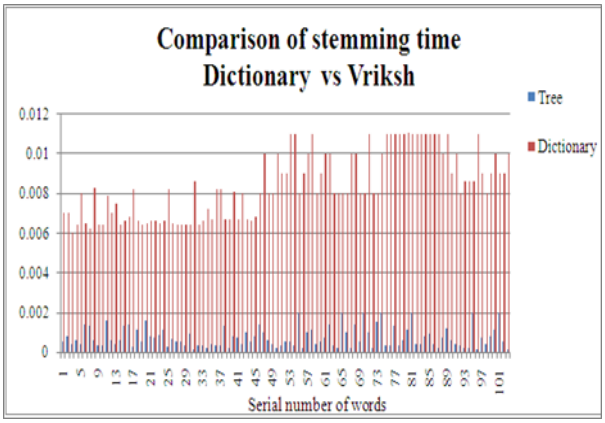

Figure 15 Comparison of Execution time

TABLE 5. Result Analysis

	Vriksh Stemmer				Indic Stemmer			
No of words in the data set	Accuracy	Precision	Recall	F1 score	Accuracy	Precision	Recall	F1 score
500	0.95	0.96	0.98	0.97	0.48	0.68	0.61	0.64
1000	0.82	0.78	0.94	0.85	0.56	0.43	0.47	0.45
2000	0.85	0.90	0.77	0.83	0.39	0.48	0.26	0.34
2500	0.81	0.75	0.80	0.77	0.69	0.88	0.44	0.59
3000	0.86	0.88	0.79	0.83	0.68	0.80	0.42	0.55
5000	0.82	0.85	0.87	0.86	0.71	0.71	0.52	0.59
7500	0.82	0.93	0.88	0.90	0.63	0.87	0.58	0.70
10000	0.86	0.79	0.98	0.86	0.63	0.78	0.61	0.68
15000	0.98	0.98	0.98	0.98	0.68	0.77	0.61	0.68
20000	0.88	0.84	0.98	0.90	0.60	0.77	0.40	0.53

Source: Since online documents from various domains are considered, it may effect the results in different cases.

9. CONCLUSIONS

There has been a lot of work in Indian languages for developing stemmers, taggers, translators etc. These types of applications help the people to communicate and work in their own mother tongue rather than depending on other languages. Now a lot of softwares dealing with native languages are available as mobile applications. There lies the importance of this project. The implementation of the tree based method makes the retrieval faster. The paper has discussed both the positive and negative sides of the proposed method. There is still room for improvement and the accuracy can be increased by adding more and more rules. Moreover the number of nodes can be reduced by making the method graph based. As no lemmatizer is available online for performance comparison and since Indic stemmer is the only link available online, the performance could be directly compared with it. On an average the proposed method has shown an accuracy of 87%.

REFERENCES

[1] Prejisha K,Dr Reghuraj P.C , "Stemmer for Malayalam Using Three Pass Algorithm", International Conference on Control Communication and Computing (ICCC), 2013.pp. 149-152.
[2] Marçal Rusiñol , Volkmar Frinken , Dimosthenis Karatzas , Andrew D. Bagdanov, Josep Lladós, "Multimodal page classification in administrative document image streams", International Journal of Document Analysis and Recognition, IJDAR 17: 2014.pp. 331–341
[3] Hongxi Wei · Guanglai Gao, "A keyword retrieval system for historical Mongolian document images", International Journal of Document Analysis and Recognition , IJDAR, 2014.Vol 17:pp.33–45.
[4] Arwa Alqudsi, Nazlia Omar, Khalid Shaker,"Arabic Machine Translation:A survey" , Artificial Intelligence Review, 2014. Vol. 42, pp. 549-572.
[5] Elaheh Asghari , MohammadReza KeyvanPour, "XML document clustering: techniques and challenges", Artificial Intelligence Review , 2015. Vol. 43, pp.417-436.
[6] Nagwani N K, "Summarizing large text collection using topic modeling and clustering based on MapReduce framework", Nagwani Journal of Big Data 2:6, 2015.
[7] Gheith A, Abandah, Alex Graves , Balkees Al-Shagoor, "Automatic diacritization of Arabic text using recurrent neural networks", International Journal on Document Analysis and Recognition, 2015, Vol 18, pp. 183-197.
[8] Osama Mohamed Elrajubi,"An Improved Arabic Light Stemmer" , Third International Conference on Research and Innovation in Information Systems, 2013, pp. 33-38.
[9] Ayu Purwarianti, "A Non Deterministic Indonesian Stemmer", International Conference on Electrical Engineering and Informatics,2011.
[10] Cristian Moral, Angelica de Antonio, Ricardo Imbert, "A survey of stemming algorithms in Information Retrieval", Information Research, 2014, Vol.19, No.1.
[11] Walid Cherif , Abdellah Madani, Mohamed Kissi, "Integrated effective rules to Improve Arabic Text Stemming", IEEE , 2014. pp. 1077 – 1081.
[12] Vishal Gupta,Gurpreet Singh Lehal , "A survey of Common Stemming Techniques and Existing Stemmers for Indian Languages", Journal of Emerging Technologies in WebIntelligence ,2013, Vol 5, No. 2 , 157-161.
[13] Jikitsha Sheth, Bankim Patel, Dhiya : "A Stemmer for morphological level analysis of Gujarati language" , International Conference on Issues and Challenges in Intelligent Computing Techniques,2014,pp. 151-154.
[14] Redowan Mahmud Md, Mahbuba Afrin, Md. Abdur Razzaque Ellis Miller, Joel Iwashige, "A rule based Bengali stemmer", ICACCI,2014, pp.2750 – 2756 .
[15] Vasudevan N, Pushpak Bhattacharya, Little by little Semi Supervised Stemming through Stem Set Minimization, ijcnlp , 2013.
[16] Vasudevan N.,Pushpak Bhattacharyya, Optimal Stem Identification in Presence of Suffix List, Computational linguistic and Intelligent Text Processing , 13th International Conference CICLing, ed . Alexander Gelbukh, New Delhi, Springer Berlin Heidelbeg, ,2012, pp.92- 103 .
[17] Rohit Kansal, Vishal Goyal and Lehal G.S, Rule Based Urdu Stemmer, Proceedings of COLING,2012, pp. 267 - 276.
[18] Upendra Mishra et al, MAULIK: An Effective Stemmer for Hindi Language, International Journal on Computer Science and Engineering ,2012,Vol. 4 , No. 05 .pp. 711 – 717.
[19] Gupta V and Lehal G. S, Punjabi Language Stemmer for Nouns and Proper Names, Proceedings of the 2nd Workshop on South and Southeast Asian Natural Language Processing , 2011,pp.35-39.
[20] Satyendr Singh and Tanveer J. Siddiqui,Evaluating Effect of Context Window Size, Stemming and Stop Word Removal on Hindi Word Sense Disambiguation, IEEE , 2012.
[21] Das .S and Mitra . P, A Rule-based Approach of Stemming for Inflectional and Derivational Words in Bengali, Proceeding of the 2011 IEEE Students' Technology Symposium , IIT Kharagpur, 2011,pp.134 – 136.
[22] Vishal Gupta,Hindi Rule Based Stemmer for Nouns, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 4, Issue 1, 62-65, 2014.
[23] Kasthuri. M, Britto Ramesh Kumar S,An Improved Rule based Iterative Affix Stripping Stemmer for Tamil Language using K-Mean Clustering, International Journal of Computer Applications , 2014, Vol.94 - No. 13, 36 – 41.
[24] Ramachandran V.A, Illango Krishnamurthi,An Iterative Stemmer for Tamil language, ACIIDS, Springer , 2012.
[25] Vijay Sundar Ram and Sobha Lalitha Devi ,Malayalam Stemmer, Morphological Analysers and Generators, ed. Mona Parakh, LDC-IL, Mysore, 2010,pp. 105 –113.
[26] Vinod P.M, jayan V, Bhadran V.K, Implementation of Malayalam Morphological Analyzer Based on Hybrid Approach, Proceedings of the Twenty-Fourth Conference on Computational Linguistics and Speech Processing, ROCLING, 2012, pp. 307 – 317.
[27] Jisha P. jayan , Rajeev R.R, Dr S. Rajendran, Morphological Analyser and Morphological Generator for Malayalam - Tamil Machine Translation, International Journal of Computer Applications,2011, Vol.13-No.8, pp 15 - 18.
[28] Prejitha U, Sreejith C, Reghu Raj P.C,Lalitha : A light weight Malayalam Stemmer using Suffix Stripping Method, International Conference on Control Communication and Computing (ICCC),2013,pp. 244 – 248 .
[29] Santhosh Thottingal , www.silpa.org/stemmer , SILPA project ,2014.

Authors

Dhanya P.M is currently working as Assistant professor in the Department of Computer Science and Engineering, Rajagiri School of Engineering and Technology, Rajagiri Valley, Kakkanad. She is also a research scholar in the Department of Computer Applications, Cochin University of Science and Technology, Kalamassery. She obtained her B.Tech and M.Tech degree from Cochin University of Science and Technology. Her areas of specialization include Natural language processing, Data mining, Soft Computing.

Dr.Sreekumar A is working as Associate Professor in the Department of Computer Applications, CUSAT. He took his Masters degree in M.Sc.

Mathematics, M.Tech. Computer Science and Ph.D in Cryptography. He has 24 publications to his credit.

Dr.Jathavedan M is working as Professor Emeritus in the Department of Computer Applications, CUSAT. He took his Masters degree in M.Sc Mathematics and Ph.D in Fluid Mechanics. His area of specialization include Fluid Mechanics, Differential Equations and Language Computing.