# An Iterative Stemmer for Tamil Language

Vivek Anandan Ramachandran[1] and Ilango Krishnamurthi[2]

Department of Computer Science and Engineering, Sri Krishna College of Engineering and Technology, Coimbatore – 641 008, Tamilnadu, India
{rvivekanandan,ilango.krishnamurthi}@gmail.com

**Abstract.** Stemming algorithm is a procedure that attempts to map all the derived forms of a word to a single root, the stem. It is widely used in various applications with the main motive of enhancing the recall factor. Apart from English, researches on developing stemmers for both the native and the regional languages are also being carried out. In this paper, we present a stemmer for Tamil, a Dravidian language. Our stemmer effectiveness is 84.32%.

**Keywords:** Information Retrieval, Stemming, Tamil.

## 1 Introduction

A program that performs stemming is referred as a stemmer [1]. Stemmer attempts to map a derived form of a word to its root. For example, stemmer maps the word *creation* to the term *cre*. Resultant of a stemmer need not be a proper meaningful word. This could be understood from the example as *cre* is not a meaningful word.

Stemmers are widely used in the query based systems such as web search engine, question answering etc as a factor to retrieve more number of documents relevant to the input . This is mainly due to the reason that such systems treat the word and its derived forms as one and the same. For example, when a user wants to search the documents with the term *creating* they might also need the documents with the term *creates* or *created*.

In recent years web documents and other information are published in languages other than English too. These published information made researchers to focus on the need for developing computational supportive tools such as stemmer, lemmatizer, parts-of-speech tagger etc for languages other than English. In this paper, we discuss our experience in developing stemmer for Tamil[1][2]. In this paper, we propose a suffix stripping stemmer for Tamil.

This paper is organized as follows: In Section 2 we discuss the researches related to our stemmer. In Section 3 we brief about the Tamil suffixes. The difficulties in developing stemmer for Tamil language are highlighted in Section 4. In Section 5 we describe about our algorithm. In Sections 6 and 7, we present the evaluation analysis and the concluding remarks respectively.

[1] Tamil is a Dravidian language. It is spoken majorly by the Tamil people of the Southern India and Substantial minorities in Malaysia, Mauritius and Vietnam. Throughout the paper transliterated form of Tamil as quoted in the Appendix Section is used.

J.-S. Pan, S.-M. Chen, N.T. Nguyen (Eds.): ACIIDS 2012, Part III, LNAI 7198, pp. 197–205, 2012.

## 2 Related Work

Earlier, stemmers were primarily developed for English language[3][4]. But later due to the corpus growth of languages other than English, there was an increased demand from the research community to develop stemmers for other languages too. In the case of Indian languages, stemming was first reported for Hindi in 2003[5]. Slowly investigations for other languages such as Bengali[6], Urdu[7], Malayalam[8] and Punjabi [9]were also carried out. However, there is no readily available stemmer for Tamil. In this paper, we present our research experiences in developing a Tamil Stemmer.

To develop a stemmer for Tamil or any other languages, a basic approach to carry out the process is required. The most common approaches used for developing a stemmer are Brute force, Affix Stripping, N-Gram, Hidden Markov Model (HMM), Corpus based technique, Clustering method, Finite-State-Automata method, Morphological process, String distance measure, Hybrid approaches. Among all the existing approaches, we make use of affix stripping because of its inherent support to develop a stemming algorithm in an easier and faster way.

Most of the existing stemmers remove the suffixes based on the longest matching word. For example among the matching suffixes *ates*, *tes*, *es* and *s* existing in the word creates, the suffix *ates* will be removed by a stemmer. Similar to most of the existing stemmers, we remove the suffixes based on the longest matching word.

To develop a stemmer for a language, a preliminary study on the possible suffixes of a word in the corresponding language need to be taken. In the next Section, we explain about the possible suffixes for Tamil.

## 3 Tamil Suffixes

In Tamil, a word usually contains a root to which one or more affixes can be attached. The affixes can be either a prefix or a suffix. We have designed our algorithm to handle only Tamil suffixes, so we discuss only about it. Tamil word can have multiple suffixes there is no exact limit for attaching the number of suffixes to a Tamil word.

To know about the possible suffixes that a word in Tamil language one can refer to the flow charts [10] and [11]. Besides considering the possible suffixes forms of a word, a stemming algorithm has to consider certain standard computing issues. In the next Section, we explain the computing issues considered for developing our stemmer.

## 4 Computing Issues

Developing a Tamil stemmer is not a straightforward task. In this Section, we explain the major difficulties faced by us while designing the algorithm. They are briefed as follows:

### 4.1 Homographs

Homographs are the words that have identical pronunciations but different meanings. In Tamil, there are numerous instances of homographs. For example, the word *aaNTavan* denotes either the noun *God* or the verb *ruled by a male* formed from the root *aaL*. It is difficult for a rule-based stemmer to map such terms to their root. Hence, we decided to frame our algorithm by do not considering the homograph issues.

### 4.2 Irregular Verbs

Irregular verbs do not follow standard patterns in their tense form. For example, the past tense of the verb *say* is *said* and not *sayed*. Mapping such forms of word to a single root is a difficult task. Such cases exist in Tamil also. For example, the past, present and future tense of the verb *sol* (*say*) are *sonneen* (*I said*), *solkiReen* (*I am saying*) and *solveen* (*I will say*) respectively. Following the standard patterns the past tense for *sol* should be *solneen*. However, this is not correct. Devising rules to handle such case is arduous as it needs a deep look up dictionary. Hence, we decided to consider this issue in the future version.

### 4.3 Proper Noun Derivations

A proper noun usually indicates a particular thing. In certain cases, proper noun end letters match with the normal suffixes. Assuming those patterns to be suffixes, most of the stemmers remove them from the proper noun. For example, Porter stemmer maps the proper noun *creator* to *cre* due to the assumption that *ator* is a suffix. To overcome such cases, it is very difficult to realize a proper noun by framing hand-crafted rules. Therefore, similar to most of the algorithms, if the common suffix patterns exist in a proper noun we decide to stem it. For example, our approach maps *iyakkiyavan* (*A person who operated*) to *iyakki* (*operated*).

### 4.4 Handling Non Derived Words Ending with Usual Suffix Pattern

In some cases, word ends with few patterns that match with a suffix but that pattern does not denote a suffix. For example:

- In English, the word *ring* contains *ing* which usually denotes a suffix but not in this case.
- In Tamil, the word *kathai* (*story*) contains *ai* which usually denotes a suffix.

It could be inferred that to handle this case a stemmer needs a heavy look-up table and this table cannot be constructed easily. So we decided to handle this case in the future versions.

### 4.5 Study on the Number of Iterations Needed to Remove Suffixes

We have already mentioned that Tamil words can have multiple suffixes. It is discussed in the previous Sections that the best way for removing multiple suffixes is iterating the suffixes in the descending order of their length and removing the suffixes in the input. But an interrogation arises on the following issue:

- For a stemmer to remove n suffixes in an input whether n iterations are needed or it could be done with less than n iterations?

For addressing this issue, we discuss three cases:

#### 4.5.1 For Removing n Suffixes Less than n-1 Iterations Are Needed

Consider the word *nuulkaLiliruntu* (*from the book*). It has three suffixes to be removed viz. *kaL*, *il* and *iruntu*. In our rule-base three suffixes will be in the order *iruntu*, *kaL* and *il*. However, in the example they are in the removal order *iruntu*, *il* and *kaL*. So if our approach follows linear methodology the suffix *iruntu* and *il* matches with the example. Therefore, they are stripped from *nuulkaLiliruntu*. So, the resultant output is *nuulkaL*. However, the suffix *kaL* is not removed from the input. During the next iteration the suffix *kaL* will be removed. So for removing 3 suffixes 2 iterations are needed.

#### 4.5.2 For Removing n Suffixes n Iterations Are Needed

Consider the word *nuulkaLil* (*in the book*). It has two suffixes to be removed *kaL* and. In our rule-base two suffixes will be in the order *kaL* and *il*. However, in the example they are in the removal order *il* and *kaL*. The first iteration removes the suffix *il* and the second iteration removes the suffix *kaL*. So for removing 2 suffixes 2 iterations are needed.

#### 4.5.3 Ascending and Descending Order of Removal of Suffix Affecting the Number of Iterations

Consider the word *nuulkaLukkupatil* (*from the book*). It has three suffixes to be removed viz. *kaL*, *ukku* and *patil*. Our rule-base apart from the three suffixes *kaL*, *ukku* and *patil* will also contain the suffixes *il* and *kku* which matches with the input. They will be in the order *patil*, *ukku*, *kku*, *kaL* and *il*. Let us see what happens when suffixes are iterated in ascending and descending order of the length.

- If suffixes are iterated in descending order of the length the suffix *patil*, *ukku* and *kaL* will be removed in subsequent iterations and the final output will be *nuul* which is the expected one.
- If suffixes are iterated in ascending order of the length only the suffix *il* will be removed and the final output will be *nuulkaLukkupat.* But the desired output is *nuul.*

After analyzing the above three cases for an effective stemmer output we decided to perform the following functionalities in our stemmer:

- Remove *n* suffixes in *n* iterations.
- Iterate the suffixes in the descending order of their length.

We also propose a novel single iteration approach for removing n suffixes in a single iteration.

### 4.6 Handling Agglutinative Case

Tamil is an agglutinative language; a compound word can be formed from two or more simple words without changing the meaning of the simple words. For example consider the word *maJainiir* (Rain Water) formed from two simple words *maJai* (Rain) and *niir* (Water). Consider the word *maJainiiri_n* (of the rain water). The word is a derived form of *maJai*. Mapping the word *maJainiiri_n* to the simple word *maJai* is a laborious task. So we decided to neglect mapping compound word to simple word.

Apart from the above discussed computational issues, proper computational steps should also be designed to develop a good stemmer. In the next Section, we explain the design portion of our stemmers.

**Table 1.** Stemming Algorithm

| Input | Tamil String (*Input*), Suffix List (*SL*) |
|---|---|
| Output | Root of the *Input* (*Output*) |
| Prerequisite | The *SL* should be stored in descending order of suffixes length |

```
`Function String stem (Input)
Begin
          1. String Output,
          2. String Temp-Output= ruleBase(Input)
          3. While Input != Temp-Output
                a. Temp-Output = ruleBase(Input)
                b. Input = Temp-Output
          4. return Output
End
Function String ruleBase (temp)
Begin
        1.    Flag = true
        2.    While (Flag)
                a.    Iterate all the suffix one by one
                      i.     If the temp ends with any suffix (say Sf )
                                A.   temp = temp - Sf
                                B.   break
                b.   return temp
End
```

## 5 Design

Generally, for removing multiple suffixes existing in a word of any language iterative stemmer is used. An iterative stemmer starting from the end of the inflected input word will remove a longest matching suffix at a time and progress towards the root. As discussed in our design section we have designed our stemmer to remove *n* suffixes in *n* iterations. The algorithm pseudo-code is presented in Table 1. The list of suffixes that our stemmer can handle is listed in Table 2.

Consider the input derived word *choRkaLi_nil* (*in the words*) derived from the word *chol*(*word*). During the first , second and third iterations suffixes *il* , *i_n* and *kaL* will be removed respectively. The output will be *choR*. Although the algorithm for stemming Tamil words is designed successfully, it has to be evaluated. In the following Section, we present the analysis carried out by us to study the algorithm's effectiveness.

**Table 2.** Tamil Suffixes

| | | | | | | |
|---|---|---|---|---|---|---|
| *etirttaaRpool* | *appuRam* | *tavira* | *teRku* | *uTa_n* | *oTTi* | *kiR* |
| *aTuttaaRpool* | *koNTiru* | *muulam* | *aa_na* | *iyal* | *iTam* | *een* |
| *veeNTiyiru* | *veeNTum* | *piRaku* | *tolai* | *avaL* | *kiiJ* | *aaL* |
| *uNkaLee_n* | *etirkku* | *pi_npu* | *ava_n* | *mu_n* | *ttal* | *uL* |
| *vaJiyaaka* | *aayiRRu* | *pakkam* | *illai* | *avai* | *tiir* | *um* |
| *varaikkum* | *veLiyil* | *umee_n* | *poola* | *avar* | *paar* | *tt* |
| *veeNTivaa* | *kuRittu* | *meelee* | *aakum* | *a_na* | *atu* | *il* |
| *mu_n_naal* | *maatiri* | *kki_nR* | *kiTTa* | *paTi* | *aam* | *pp* |
| *patilaaka* | *paarttu* | *taaNTi* | *ki_nR* | *viTa* | *aar* | *ai* |
| *tavirttu* | *naTuvil* | *uTaiya* | *pooTu* | *meel* | *aay* | *al* |
| *illaamal* | *vaTakku* | *etiree* | *oJiya* | *kiJi* | *kka* | *ya* |
| *veeNTaam* | *meeRku* | *uNkaL* | *paNNu* | *ukku* | *iir* | *nt* |
| *kuRukkee* | *kuuTum* | *aarkaL* | *koNTu* | *viTu* | *kaL* | *a* |
| *allaamal* | *aTiyil* | *vaittu* | *aNTai* | *chey* | *oom* | *p* |
| *varaiyil* | *etiril* | *kiiJee* | *uLLee* | *kiTa* | *poo* | *t* |
| *kuuTaatu* | *aaTTam* | *arukee* | *taLLu* | *muTi* | *vaa* | *u* |
| *veLiyee* | *appaal* | *iirkaL* | *paRRi* | *ooTu* | *i_n* | *v* |
| *iTaiyil* | *iIruntu* | *kaaTTu* | *pinti* | *koTu* | *tal* | |
| *aakaatu* | *chuRRi* | *nookki* | *patil* | *kkiR* | *vai* | |
| *kiJakku* | *arukil* | *viTTu* | *mutal* | *aa_n* | *iru* | |

## 6 Evaluation

In general, evaluation signifies the act of assessing something. To evaluate our stemmer we implemented our algorithm in Java. A sample screen shot of our stemmer is shown in Figure 1.

After developing any stemmer it is important to analyze its capability, i.e., to assess the range of the words that the system is able to stem properly. We requested two students, none of whom are directly or indirectly involved with our project to generate the corpus. They developed a Tamil corpus containing 36720 words derived from 765 roots. The corpus is framed from different portions of Tamil newspapers confining to various domains such as Business, Classifieds, Entertainment, Politics and Sport.

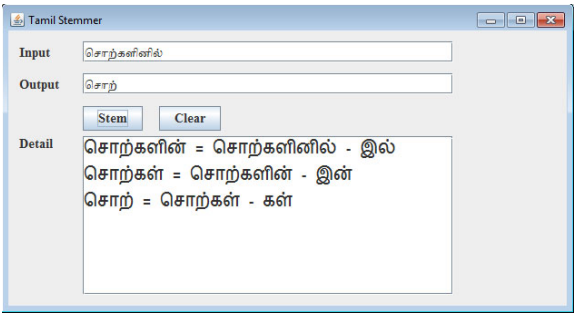


**Fig. 1.** Sample screen shot of the Tamil Stemmer

It is important to remember as stated in the Introduction that the output of a stemmer need not be a proper linguistic word. Therefore, correctness of a stemmer does not denote the linguistic correctness. A stemmer is said to be accurate if it conforms to the following conditions:

- If it maps all the derived forms of a word to a single root.
- The words mapped by it to a single stem are genuine linguistic variants.
- If it does not stem a non-suffix from a word.

If a stemmer does not map all the considered derived forms of word to a single stem then the phenomenon is called Understemming. An instance of Understemming is a stemmer conflating *tried* to *tri* and *try* to *try* instead of mapping both to *try*. Our stemmers map *cholvava_n* (*A man who is saying*) to *chol* (*say*) and *cho_n_nava_n* (*A man who said*) to *cho_n_n* instead of mapping both to *chol*.

If a stemmer maps the words to a single stem that are genuinely linguistic invariants then the phenomenon is called Overstemming. An instance of Overstemming is a stemmer conflating both the words *cares* and *cars* to *car*, instead of mapping *cares* to *care* and *cars* to *car*. An example for Overstemming in our case is our stemmers map both *cheluttuki_nRava_n* (*A man who is riding*) and

*chelki_nRava_n* (*A man who is going*) to *chel* (*go*) instead of mapping *cheluttuki_nRava_n* to *cheluttu* (*ride)* and *chelki_nRava_n* to *chel*.

If a stemmer removes a nonsuffix from a word, it is called *Mis-stemming*. For example, conflating the words *reply* to *rep* instead of conflating it to *reply* is called *Mis-stemming*. Most of the stemmers do not give importance to *Mis-stemming*. This is because it does not spoil the recall factor in an IR application. Due to the same reason, we evaluate our stemmer using only *Understemming* and *Overstemming*. They are calculated using the following formulas (1) and (2) respectively.

Understemming = (Number of variants understemmed/Total variants)*100% (1)

Overstemming = (Number of variants overstemmed/Total variants)*100% (2)

Evaluating our approach using the above-mentioned corpus containing 765 root variants, we found that 84 and 36 were *understemmed* and *overstemmed* respectively. Hence, The *Understemming* and *overstemming* values are 10.98 % and 4.70 % respectively. Our stemmer effectiveness is calculated using the formula (3).

Stemmer Effectiveness = 100% - [ *Overstemming %* + *Understemming %*] (3)

Effectiveness for our approach is 84.32 %. This is considerably a good value. Yet the reason behind achieving a moderate effectiveness value is due to the factors discussed in the Section 4.

## 7 Conclusion

In this paper, we hypothesize an Iterative Tamil Stemmer. Further, it should be noted that we have evaluated the performance of our stemmer in terms of understemming and overstmming as of now. The proposed stemmers need to be further evaluated with Tamil IR system using factors such as precision, recall etc. Such evaluations will provide the best trade-off between understemming and overstemming that can be obtained by removing or adding a few suffixes in the list. This leads us to believe that our stemmers will prove to be beneficial for Tamil Information Retrieval applications. The limitation with the current version of our stemmers is their ability to handle only suffixes. Investigation on handling prefixes is a part of our future work. It would also be interesting to apply our algorithms to Tamil's Sister languages such as Malayalam, Telugu etc.

## References

1. Kraaij, W., Pohlman, R.: Viewing Stemming as Recall Enhancement. In: The Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 40–48 (1996)
2. Germann, U.: Building a Statistical Machine Translation System from Scratch: How Much Bang for the Buck Can We Expect? In: ACL 2001 Workshop on Data-Driven Machine Translation, Toulouse, France (July 7, 2001)

3. Lovins, J.B.: Development of a stemming algorithm. Mechanical Translation and Computational Linguistics 11(1), 22–31 (1968)
4. Porter, M.F.: An Algorithm for Suffix Stripping. Program 14(3), 130–137 (1980)
5. Ramanathan, A., Rao, D.: A lightweight stemmer for Hindi. In: The Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL) for South Asian Languages Workshop (April 2003)
6. Zahurul Islam, M., Nizam Uddin, M., Khan, M.: A Light Weight Stemmer for Bengali and Its Use in Spelling Checker. In: Proceedings of 1st International Conference on Digital Communications and Computer Applications (DCCA 2007), Irbid, Jordan, pp. 87–93 (2007)
7. Q.-A. Akram, Naseer, A., Hussain, S.: Assas-Band, an affix-exception-list based Urdu stemmer. In: Proceedings of the 7th Workshop on Asian Language Resources (2009)
8. Malayalam Stemmer, http://nlp.au-kbc.org/Malayalam_Stemmer_Final.pdf
9. Kumar, D., Rana, P.: Design and Development of a Stemmer for Punjabi. International Journal of Computer Applications (0975–8887) 11(12) (December 2010)
10. Tamil Noun Flow Chart, http://www.au-kbc.org/research_areas/nlp/projects/morph/NounFlowChart.pdf
11. Tamil Verb Flow Chart, http://www.au-kbc.org/research_areas/nlp/projects/morph/VerbFlowChart.pdf

## Appendix: Tamil Transliteration Scheme Used in This Paper

| a | aa | i | ii | u | uu | e | ee | ai | o | oo | au | q | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| அ | ஆ | இ | ஈ | உ | ஊ | எ | ஏ | ஐ | ஒ | ஓ | ஔ | ஃ | | | | | |
| k | -N | ch | -n | _n | T | N | t | n | p | m | y | r | l | v | J | L | R |
| க் | ங் | ச் | ஞ் | ன் | ட் | ண் | த் | ந் | ப் | ம் | ய் | ர் | ல் | வ் | ழ் | ள் | ற் |