ISSN: 2394-3122 (Online)
e-ISJN: A4372-3088
Impact Factor: 5.045

Volume 5, Issue 12, December 2018

SK International Journal of Multidisciplinary Research Hub

Journal for all Subjects

Research Article / Survey Paper / Case Study

Published By: SK Publisher *(www.skijmr.com)*

Prototype Development for Stemmer of Wolaita Language

Girma Yohannis Bade
Department of Computer Science, Informatics School
Wolaita Sodo University,WSU
Wolaita Sodo, Ethiopia.

Abstract: ***Stemming algorithms (stemmer) are commonly known in a domain of Natural Language Processing (NLP) and which has positive impact on Information Retrieval (IR) system and morphological analysis. Most stemmers that has been developed so far are either in the design level or in interact way (using C ++). However, this paper aims to develop a graphical interface-based prototype stemmer for Wolaita texts using longest-match based approach. How to compile the possible combination of all the needed suffixes, researcher deeply analysed the morphology of Wolaita language. To pre-process (to tokenize) the texts by white space I used another C# application which put the actual texts after filtering unwanted characters and the prototype stemmer also written by C# programming language. The developed prototype stemmer was tested using a method proposed by Paice. On the test dataset the performance of the prototype was evaluated to 91.84% accuracy over same data was stemmed by manually.***

Keywords: Prototype; Stemmer; Morphology; Wolaita; Language.

I. INTRODUCTION

As an information technology has contributed a great availability of recorded information, there was a mass production of electronic information, and digitalized library collections. This increased the awareness of society for storing, maintaining and retrieving information in a systematic way [1]. Information retrieval (IR) system one of such a systematic ways is designed to facilitate the access to stored information [2]. To enhance the effectiveness of IR performance, the suffix stripping process (stemmer) helps by reducing the total number of terms in the IR system into common forms as conflating the variants of words which increases the success of matching of documents to a query. In another hand, natural language text processing system may begin with morphological analyses known as stemming. Again stemmer can be also used as computational linguistic to compress all morphological word variants which will help non experts the right root (stem) of certain word class. Having this consideration this study targets to develop a prototype stemmer for a Wolaita language. Wolaita is one of the Northern Omotic languages that spoken in the Wolaita Zone and some other parts of the Southern Nations, Nationalities, and People's Region of Ethiopia. The Latin script is being used since 1993 to write Wolaita texts and which is one of a morphologically rich language [1].

As each natural language has its own characteristics and features. So, it is quite difficult to follow the same stemming pattern and apply the same stemming rules for other languages. This is because of different prefixes and suffixes, and exceptions needs a special handling and a careful formation of frame with specific norms. Even though the iterative-based Wolaita stemmer was developed so far, the performance of the stemmer was poor. This is because, since Wolaita words' formation only depends on suffixation which led iterative-based Stemmer to face computational complexity. In addition, the previous research has not used the graphical interface but interactive which is not suitable for non-computer person [1]. Concerning these issues, it is important to develop prototype stemmer with better accuracy using longest-match based approach.

II. METHODOLOGY

2.1 Stemming Techniques

Several automated procedures for stemming are--table lookup, successor variety, n-gram, and affix removal. For this study the longest match approach one of the affix removal is chosen. Each procedures are discussed as follows:

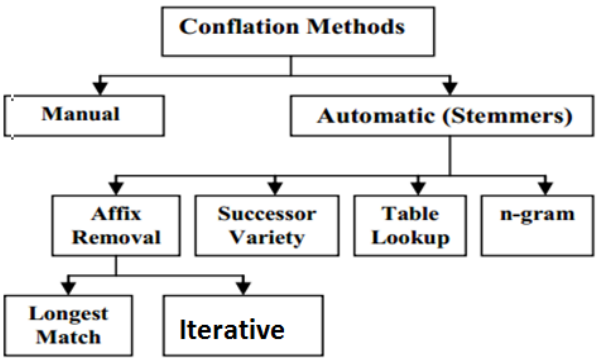

Figure 1: The approaches of stemming

2.2 Table lookup

Table lookup is the way of storing a table of all index terms and their stems. Terms from queries and indexes could then be stemmed via table lookup. The problem of this procedure is the storage overhead for such a table, though trading size for time is sometimes warranted.

2.3 Successor variety

Successor variety stemmers use the frequencies of letter sequences in a body of text as the basis of stemming. It based on work in structural linguistics which attempted to determine word and morpheme boundaries based on the distribution of phonemes in a large body of utterances. The successor variety of a string is the number of different characters that follow it in words in some body of text. The successor variety of substrings of a term will decrease as more characters are added until a segment boundary is reached. It scans the word to be stemmed and finds the cut point where the successor variety increases sharply. Several variations are possible, including: cut-off, peak and plateau, complete word and entropy.

2.4 Statistical Approach

N-gram is one of statistical method which conflates terms based on the number of di-grams or n-grams they share. A di-gram is a pair of consecutive letters. Since trigrams or n-grams could be used, we have called it the n-gram method. The N-gram stemming underperformed stems and required significant amount of memory and storage for index, but its ability to work with an arbitrary language makes it useful for many applications.

2.5 Affix removal

Affix removal algorithms remove suffixes and/or prefixes from terms leaving a stem. These algorithms sometimes also transform the resultant stem. Basically, it has two approaches (longest match and iterative). Longest match involves only a single pass, i.e., if more than one suffix matches the end of the word, the longest one is removed. Lovins first introduced affix removal stemming algorithm which used longest match approach [6]. Iterative approach - iterates over a set of predefined classes of suffixes. Porter's affix removal algorithms used iterative approach [7].

2.6 Tokenization

Tokenization is a tasks chopping it up into pieces called tokens, perhaps at the time throwing certain characters, such as punctuations, and only remains white space separated words. For this I provided a C# application which screens (tokenizes) the words going to be stemmed and save it another independent folder.

The following shows the code used for tokenization

```
private void wordCountit(string s, int a)
{
String text = s.Trim();
String worda = s.Trim();
wordCount = 0;
int index = 0;
string wordText = "";
while (index < text.Length)
{
                    check if current char is part of a word
while (index < text.Length && Char.IsWhiteSpace(text[index]) == false)
{
        if (text[index] == '.' || text[index] == ',' || text[index]==  '(' || text[index] == ')' || text[index] == '?'
|| text[index] =='.' || text[index]== '/' || text[index] == '#' || text[index] == '$' || text[index] == '%' || text[index]
== '}' || text[index] == '{' || text[index] == '@' || text[index] == '~' || text[index] == ';' || text[index] == ':' ||
text[index] == '?' || text[index] == '.' || text[index] == "" || text[index] == ']' || text[index] == '<' || text[index] ==
'>' || text[index] == '□' || text[index] == '□' || text[index] == '□' || text[index] == '©' || text[index] == '‹' ||
text[index] == '›' || text[index] == '-' || text[index] == '•' || text[index] == '&' || text[index] == '^' || text[index]
== '»' || text[index] == '«' || text[index] == '÷' || text[index] == '–' || text[index] == "" || text[index] == '□' ||
text[index] == '*' || text[index] == '`' || text[index] == '□' || text[index] == '´' || text[index] == '□' || text[index]
== '_' || text[index] == '_' ||
text[index] == '!' || text[index] == '+' || text[index] == '[')
{ }
else if (char.IsNumber(text[index]))
{ }
else
{
wordText += text[index].ToString();
```

2.7 Data collection and suffix compilation

For this study, corpuses were collected from different domains (from dictionary, bible, and Wolaytta educational materials). These collected sample datasets were preprocessed (tokenized and redudandacy eliminated) and 12789 unique words were left, which included the inflected and derived form of words according to their tense, number, gender and moods. To test the performance of the prototype, 1200 words of the sample texts were randomly selected.

2.8 Source code for prototype

```
private void strip()main thing is here
        {
int dd = 0;
int tt = 0, kir = 0;
string listItemsG;
foreach (Test test1 in tests)indicates corpus
{
  dd = 0;
  kir = 0;
  l = new int[j];
  if (isStopWord(test1.X2) == true)
  {
    foreach (Sample sample1 in samples)indicates suffix
    {
      listItemsG = sample1.X1;
      listItemsG = vowelReduction(listItemsG);
      if (test1.X2.Length > listItemsG.Length)
      {
                  if (test1.X2.Length > 2)controls teh length of the word to be stemmed
        {
          int dt = test1.X2.Length;-1
          tt = listItemsG.Length;
          for (int i = tt; i >= 1; i--)-1 from tt
          {
          if (test1.X2[dt - b] == listItemsG[i])
          {
            dd++;
          }
          if (sample1.X1.Length >= test1.X2.Length)
          {
            tt -= 2;                    }
```

```
            if (test1.X2.EndsWith(sample1.X1))
            {
                                                    if (suffixStripper(stringReduction(test1.X2, sample1.X1), sample1.X1))
              {
                for (int i = 1; i <= l.Length; i++)
                                                    richTextBox2.Text += "\n " + "\t" + stringReduction(test1.X2, listItemsG);
                richTextBox2.Enabled = false;
                kir = 1;
                break;
             }                          }
            else
            {
               richTextBox2.Text += " " + test1.X2;
            }
            }
        }
        if (dd == tt)
        {
            kir++;
            ListViewItem uuu = new ListViewItem(test1.X2, 1);
            listView3.Items.Add(uuu);

                                        richTextBox2.Text += " " + stringReduction(test1.X2, listItemsG);
            break;
        }
        else {
        }
        }
        }
        label1.Text = listView3.Items.Count.ToString();
    }               }
```

```
            if (kir == 0)
            {
                richTextBox2.Text += " " + test1.X2.ToString();X2 for the test1 & X1 for the sample1
            }
             }
            else
                foreach (Stop s in stops)
                {
                    richTextBox2.Text += " " +s.S1.ToString();
                }               }
        else
        {
            richTextBox2.Text += "\n" + test1.X2.ToString() + "";
        }
    }
}
public bool isStopWord(string p)
{
    foreach (Stop stop1 in stops)
    {
        if (p == stop1.S1)p == stop1.S1
            return false;
    }
    return true;
}
 private bool suffixStripper(string x, string y)
{
    if(x == "k" && y == "eetta")
    {
        return false;
    }
    else if (x == "ke" && y == "etta")
```

```
{
   return false;
}
else if (x == "y" && y == "oota")
{
   return false;
}
else if (x == "yo" && y == "ota")
{
   return false;
}
else if (x == "w" && y == "aani")
{
   return false;
}
else if (x == "n" && y == "uussi")
{
   return false;
}
else if (x == "n" && y == "eeni")
{
   return false;
}

else if (x == "ara" && y == "ata")
{
   return false;
}
else if (x == "n" && y == "uussi")
{
   return false;
}
```

```
else if (x == "ma" && y == "ata")
{
   return false;
}
else if (x == "ma" && y == "ataa")
{
   return false;
}
else if (x == "at" && y == "otetta")
{
   return false;
}
else if (x == "ma" && y == "ayo")
{
   return false;
}
else if (x == "aw" && y == "ude")
{
   return false;
}
else if (x == "axir" && y == "aaro")
{
   return false;
}
else if (x == "mat" && y == "atetta")
{
   return false;
}
else if (x == "ays" && y == "aara")
{
   return false;
}
```

```
else if (x == "aysa" && y == "ara")
{
    return false;
}
else if (x == "attum" && y == "aasa")
{
    return false;
}
else if (x == "attuma" && y == "asa")
{
    return false;
}
else if (x == "a" && y == "atti")
{
    return false;
}
else if (x == "a" && y == "ayo")
{
    return false;
}
else if (x == "t" && y == "aagaa")
{
    return false;
}
else if (x == "n" && y == "eegaa")
{
    return false;
}
else if (x == "h" && y == "eera")
{
    return false;
}
```

```
    else if (x == "he" && y == "era")
    {
        return false;
    }
    else if (x == "at" && y == "ara")
    {
        return false;
    }
    else if (x == "ata" && y == "ra")
    {
        return false;
    }
    else if (x == "ag" && y == "uppe")
    {
        return false;
    }
    else if (x == "t" && y == "ani")
    {
        return false;
    }
    return true;
}
private string stringReduction(string a, string b)
{
    string asw = "";

    for (int i = 0; i < a.Length - b.Length; i++)
    {
        asw+= a[i].ToString();
    }
    return asw+"-";
}
```

III. Discussion and Result

This work has consisted three modules. The first one that of tokenizing the data that comes to be stemmed. The second module is actual prototype which strips certain words based on pre-set rules. And the third one is the module which is developed for testing purpose. Then the software that was developed for test purpose compares both manually stemmed datasets and system stemmed ones and showed 91.84% accuracy. Prototype generated 8.16% errors on unseen test datasets. Of which 2.08% are under stemmed errors and 6.08% are over stemmed errors. The following table shows the detail.

Table 1: The results of stemmer

Number of incorrectly stemmed=98 i.e.,(98/1200)*100=8.16%	Under stemmed=25 i.e., (25/1200)*100=2.08%
	Over stemmed=73 i.e.,(73/1200)*100=6.08%
Nº of Correctly stemmed=1102	(1102/1200)*100=91.84%
The accuracy of the stemmer	Nº of Correctly stemmed /total=91.84%

The main advantage of stemming is reducing one class of a variety of words to a single stems, the process is called word compression. In terms of compression, i.e., reduction of dictionary size, using Paice [20] formula:

$$C = 100 \times \frac{(T_W - D_S)}{T_W} \quad \text{.....(1)}$$

Where, C is the compression value (in percentage), T_w is the number of the total words and D_s is a distinct stem after conflation. Accordingly,

Size of the test datasets (Tw)= 1200

Number of distinct stems(Ds) = 755,

This 755 distinct stems are obtained by removing duplicates from the output of the stemmer because; however before running into the system we have no duplicate terms. Hence, the percentage of compression for Wolaytta text based on sample test dataset applying the above formula becomes:

C= 100 * (1200 - 755) / 1200 = 100*(445/1200) = 37.08%.

This indicates that after stripping the suffixes, 445 words of duplicates have found and removed. Then 755 distinct words are remained, and this also shows the balanced distribution of test datasets. Thus, the compression rate of test datasets from 1200 unique words to 755 distinct words is 37.08%.

Generally, if the words returned from the stemmer as it is, it must be personal pronouns (stop words) such as {a, i, o} otherwise every word should be stemmed correctly or incorrectly. Because every Wolaytta words end with one of the five vowel phonemes and they are present in the suffix list as suffixes, at least one character of suffix will be matched and removed.

IV. Conclusion

Stemming is important for highly inflected languages like Wolaita for many applications that require the stem of a word. In this work, a context sensitive longest-match based stemmer was developed, and attempted to determine the stem of a word according to linguistic rules. As the compression rate of the test dataset shown, Wolaita is a morphologically rich language. The effect is due to its inflectional and derivational morphologies. Suffixes in Wolaita plays a great role forming many variants of words. As a result, more than one combination of the suffixes can be appended to the root and; thus the length of the words in Wolaita is very long. Thus, the longest match approach one of affix removal procedure has been employed in this study. According to the evaluation result of the stemmer on test datasets, about 91.84% of accuracy with actual stemmed words and which is an encouraging figure.

References

1. Lemma Lessa. "Development of stemming algorithm for wolaytta text."M.Sc. Thesis, Addis Ababa University, Department of Information Science, Addis Ababa,(2003).
2. Salton,G. & McGill, N."Introduction to Modern Information Retrieval". New York: McGraw-Hill, (1983).
3. Liddy, E."Enhanced text retrieval using natural language processing." Bulletin of the American Society for Information Science, 24, PP.14-16, (1983).
4. Schinke, R, et al. "A Stemming Algorithm for Latin Text Databases." In Journal of Documentation. 52(2), PP. 172 – 187, (1996).
5. Lamberti, Marcello and Sottile, Roberto. "The Wolaytta Language.Koln: Rudiger Koppe Verlag.", (1997).
6. Lovins J.B. 1968: "Development of a stemming algorithm".Mechanical Translation and Computational Linguistics 11, 22-31, 1968.
7. Porter Porter, M.F. 1980: "An algorithm for Suffix Stripping"Program 14, pp 130-137, 1980.
8. Yonas Fiesseha,"Development of Stemming Algorithm for Tigregna text." M.Sc.Thesis, Addis Ababa University, Department of Information Science, Addis Ababa, (2011)
9. "The Language Gulper" [Online]. Available: http://www.languagesgulper.com/eng/Omotic.html. [Accessed: May 10, 2014].
10. Nega Alemayhu."Development of a Stemming Algorithm for Amharic Language Text for Retrieval".Ph. D. Thesis. University of Sheffield, (1999).
11. Girma Berhe. "A Stemming algorithm development for Tigrigna language text documents. ",M.Sc.Thesis,Addis Ababa University,Department of Information Science, Addis Ababa,(2001).
12. Motomichi, Wakasa. "A descriptive study of modern Wolaytta language",Doctoral Dessertation ,Tokyo,(2008).
13. Adams, B.A. "A Tagmemic Analysis of the Wolaitta Language." Ph.D.Thesis. University of London (unpublished), (1983).
14. FRAKES, W. B."Term Conflation for Information Retrieval", Doctoral Dissertation, Syracuse University. (1982).
15. LENNON, M., D. PIERCE, B. TARRY, and P. WILLETT. (1981) "An Evaluation of Some Conflation Algorithms for Information Retrieval." Journal of Information Science, 3, PP.177-83.
16. SALTON, G. "Automatic Information Organization and Retrieval". New York: McGraw-Hill,(1968).
17. MITH, L. C. "Selected Artificial Intelligence Techniques in Information Retrieval."Doctoral Dissertation, Syracuse University,(1979).
18. Savoy, Jacques. (1993). "Stemming of French Words Based on Grammatical Categories." In Journal of American Society for Information Science, 44(1):PP. 1-9.
19. "Snowball, a language for stemming algorithm."[Online].Available: http://snowball.tartarus.org/texts/introduction.html [Accessed: Jul 21, 2014].
20. Paice C.D. "An Evaluation Method for Stemming Algorithms". ACM SIGIR Conference on Research and Development in Information Retrieval. 1994, 42-50.

