

SEMANTIC SIMILARITY OF MALAYALAM SENTENCES

Main Project Report

Submitted by

HARISANKAR T P

Reg No: FIT18MCA-D022

Submitted in partial fulfillment of the requirements for the award of the degree of

Master of Computer Applications

of

APJ Abdul Kalam Technological University



FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)[®]

ANGAMALY - 683577, ERNAKULAM (DIST.)

MAY 2020

DECLARATION

I hereby declare that the report of this project work, submitted to the Department of Computer Applications, Federal Institute of Science and Technology (FISAT), Angamaly in partial fulfillment of the award of the degree of Master of Computer Applications is an authentic record of my original work.

The report has not been submitted for the award of any degree of this university or any other university.

Date :

Place:

HARISANKAR T P



FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)[®]

Accredited by NAAC with 'A' Grade

(An ISO 9001:2015 Certified Engineering College – Owned and managed by Federal Bank Officers' Association Educational Society)

HORMIS NAGAR, MOOKKANNOOR P.O., ANGAMALY - 683 577, ERNAKULAM DT., KERALA, S. INDIA.

(Approved by AICTE – Affiliated to APJ Abdul Kalam Technological University, Kerala)

Tel: (O) 0484-2725272 Fax: 0484-2725250 E-mail: mail@fisat.ac.in Website: www.fisat.ac.in

25th May, 2020

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Mr/Ms. HARISANKAR T P(Reg. No. FIT18MCA-D022) has successfully completed his/her Main Project with the title "SEMANTIC SIMILARITY OF MALAYALAM SENTENCES", in the Department of Computer Applications, FISAT, during the period from 13-01-2020 to 28-04-2020.

SANTHOSH KOTTAM
HEAD OF THE DEPARTMENT



FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)[®]

ANGAMALY, ERNAKULAM-683577



CERTIFICATE

*This is to certify that the project report titled “ **SEMANTIC SIMILARITY OF MALAYALAM SENTENCES**” submitted by HARISANKAR T P (Reg No.FIT18MCA-D022) towards partial fulfillment of the requirements for the award of the degree of Master of Computer Applications is a record of bonafide work carried out by him during the year 2020.*

Project Guide

Head of the Department

Submitted for the viva-voice held on at

Examiners :

ACKNOWLEDGEMENT

I am extremely glad to present my main project as a part of our curriculum. I take this opportunity to express my sincere thanks to those who helped me in bringing out the report of my project.

I would like to express my special thanks of gratitude to **Dr. George Issac** , Principal, FISAT, Angamaly, as well as our vice principal **Dr. C. Sheela** , who gave me the opportunity to avail the facilities of the college to do this project.

My sincere thanks to **Mr. Santhosh Kottam**, Head of the Department of Computer Applications, FISAT, who had been a source of inspiration. I express heartiest thanks to **Mr. Sujesh P. Lal**, Project Scrum Master for his encouragement and valuable suggestion. I express my sincere gratitude to **Ms. Joice T**, my project guide for her valuable support. I express my heartiest gratitude to all the faculty members in our department for their constant encouragement and never ending support throughout the project.

Finally I express my thanks to all my friends who gave me wealth of suggestion for successful completion of this project.

HARISANKAR T P

ABSTRACT

Semantic Similarity Of Malayalam Sentences is the process of calculating how much the meaning of words, sentences or documents are similar. Semantic analysis gives the exact meaning or dictionary meaning of each word or sentence. Semantic analysis works with the meaning of words and sentences and refers to elements in the world. It is the more difficult part of NLP and not yet solved fully. Semantic analysis can be used in many NLP applications like Information Retrieval (IR), text classification, text summarization, topic detection, plagiarism detection etc. Semantic similarity measurement is implemented in many languages which is useful for various applications. For the measurement of semantic similarity between two sentences the important task is to find out similarity between terms or words with in the sentences.

Similarity between words can be calculated in many ways. One of the best methods to find out semantic similarity is by using wordnet of corresponding language. Two terms are matched via wordnet is by matching synonyms of the words. In many languages the semantic similarity is implemented and many researches are on-going in this area. Propose a system which will compute the semantic similarity of Malayalam via graph constructed from the given input. Similarity will compute in sentence level and similarity score ranges from 0 to 1.

Contents

1	INTRODUCTION	1
1.1	General Background	1
2	PROOF OF CONCEPT	4
2.1	Proof Of Concept	4
2.2	Objectives	5
3	IMPLEMENTATION	6
3.1	Proposed System Architecture	6
3.2	System Architecture	7
3.3	Dataset	7
3.4	Modules And Algorithms	9
3.4.1	Pre-Processing	9
3.4.2	Parse Tree Generation	20
3.4.3	Semantic Similarity Computation	24
4	RESULTS AND DISCUSSION	33
4.1	Experimental Setup	33

4.1.1	Dataset Collection	33
4.2	Sentence Comparison	34
5	CONCLUSION AND FUTURE SCOPE	35
5.1	Conclusion	35
5.2	Future Scope	36
6	APPENDIX	37
6.1	Source Code	37
6.1.1	UserApp	37
6.1.2	Main Code	42
6.2	Screenshot	46
7	REFERENCES	49

Chapter 1

INTRODUCTION

1.1 General Background

Dealing of communication between computer and humans in natural language is called Natural Language Processing (NLP) which combines many technologies namely Computer Science, Machine Learning and Linguistics. NLP is the sub-field of Artificial Intelligence (AI) which enables the computers to understand human languages. NLP has various techniques for interpreting human languages. Tokenization, parsing, lemmatization, stemming, part-of-speech tagging, language detection and identification of semantic relationships are the basic tasks included in NLP. NLP makes computers to recognize the statements or words written in human languages like English, Hindi and Malayalam etc[1].

Human language understanding is actually a difficult task due to its complexity. Semantic is a linguistic term which means something related to meaning or logic. Semantic Similarity is the process of calculating how much the meaning of words, sentences or documents are similar. Semantic analysis gives the exact meaning or dictionary meaning of each word or sentence. Semantic analysis works with the meaning of words and sentences and refers to elements in the world [2]. It is the more difficult part of NLP and not yet solved fully. Semantic analysis can be used in many NLP applications like Information Retrieval (IR), text classification, text summarization, topic detection, plagiarism detection etc.

Malayalam is the language spoken mostly in the state of Kerala and adjoining areas. Existence of Malayalam as a separate language got accepted in 15th century. After that some changes was brought by Thunchathu Ezhuthachan in 16th century who is known as father of modern Malayalam[20]. Malayalam is spoken by 4 % of India's population. Vocabulary of Malayalam borrowed from other languages like Sanskrit, Tamil etc. Two characteristics make Malayalam unique from other languages: it does not have predicate agreement between subject or object and verb for gender, number and person , and having prevalent usage of copulas in utterances.

Semantic similarity measurement is implemented in many languages which is useful for various applications. For the measurement of semantic similarity between two sentences the important task is to find out similarity between terms or words with in the sentences. Similarity between words can be calculated in many ways. One of the best methods to find out semantic similarity is by using wordnet [21] of corresponding language. Two

Semantic Similarity of Malayalam Sentences

terms are matched via wordnet is by matching synonyms of the words. In many languages the semantic similarity is implemented and many researches are on-going in this area.

But in case of Malayalam a little amount of works are done in the area of semantic similarity. Semantic similarity calculation of Malayalam language is difficult due to entirely different structure from other languages. In this work propose a system which will compute the semantic similarity of Malayalam via graph constructed from the given input. Similarity will compute in sentence level and similarity score ranges from 0 to 1. The score 0 indicates that the meaning of two sentences are entirely different, 1 indicates that the meaning of sentences are exactly same and score in between 0 and 1 means that semantics of sentences are same in some aspects.

Chapter 2

PROOF OF CONCEPT

2.1 Proof Of Concept

The project called “Semantic Similarity Of Malayalam Sentences” is based on similar project base papers, **(1. Sentence Similarity Detection in Malayalam Language using cosine similarity, 2nd IEEE International Conference On Recent Trends in Electronics Information Communication Technology (RTEICT), India. (May 19-20, 2017) , 2. Semantic Document Clustering Using a Similarity Graph, IEEE Tenth International Conference on Semantic Computing (ICSC) Laguna Hills, (2016) , 3. Sentence based Semantic Similarity Measure for Blog-Posts, arXiv:1201.2084 [cs.AI]. (2016) , 4. A study on similarity and relatedness using distributional and WordNet based approach, in Proc. Human Language Technol. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics, pp. 19-27. , 5. An approach for measuring**

semantic similarity between words using multiple information sources, IEEE Trans. Knowl. Data Eng. Vol. 15, no. 4, pp. 871-882.(Jul. /Aug 2003) , 6. Computing Semantic Similarity of Concepts in Knowledge Graphs, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 29, NO. 1.(JANUARY 2017)).

The proposed system aims to provide a semantic graph and semantic similarity score .

2.2 Objectives

- The goal of a system which will compute the semantic similarity of Malayalam sentences using semantic graph.
- Similarity will compute in sentence level and similarity score ranges from 0 to 1.

Chapter 3

IMPLEMENTATION

This section gives a detailed explanation about the proposed system which is used to measure the semantic similarity between two Malayalam sentences. Malayalam is the official language used by people in Kerala. Since it is an agglutinative language, the Malayalam sentence processing is actually a difficult task.

3.1 Proposed System Architecture

The proposed system architecture is shown in the figure 3.1. The system having following modules: Pre-processing, Parse tree generation and Semantic similarity computation.

3.2 System Architecture

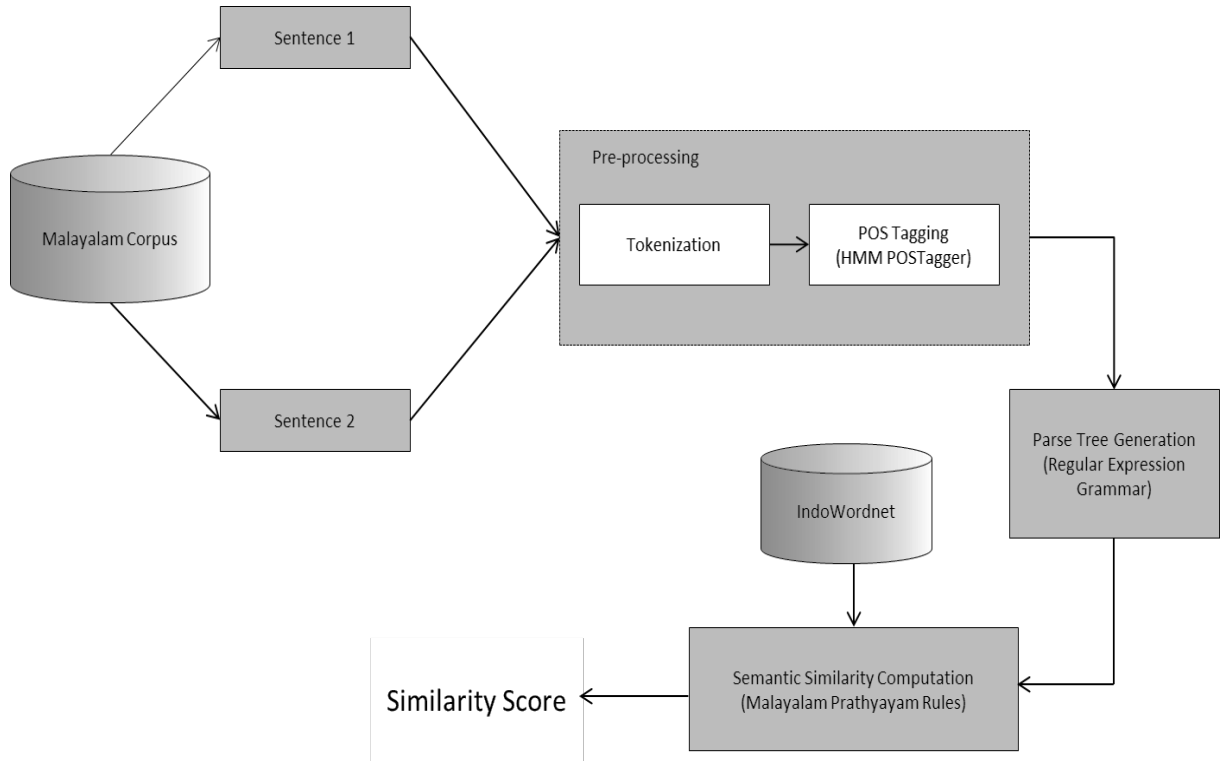



Figure 3.1: Architecture of proposed system

3.3 Dataset

The data requirements is very high for the project. Malayalam tagged corpus of Cusat dataset[22] is used in this project. This data is used for training and model file creation. The sample dataset is as follow.

 dataset.txt - Notepad

File Edit Format View Help

പ്രദേശത്ത്\\N_NN
ഒരു\\QT_QTO
മനുഷ്യനു\\N_NN
ഇല്ല\\RP_NEG .\\RD_PUNC

ഞാൻബലിക്കപ്പുരയിലേക്കു\\N_NN
കടന്നു\\V_VM_VNF .\\RD_PUNC

അതിനുള്ളിലെ\\N_NST
മങ്ങിയ\\JJ
വെളിച്ചത്തിൽ\\N_NN
എന്റെ\\PR_PRF
അച്ഛനതാ\\N_NN മലർന്നുകിടക്കുന്നു\\V_VM_VF .\\RD_PUNC

ഒരു\\QT_QTO
സത്വം\\N_NN
വയറ്റത്തു\\N_NN
കയറിയിരുന്ന്\\V_VM_VNF
അതിന്റെ\\DM_DMD
കൂർത്തു\\N_NN
നീണ്ട\\JJ
നഖങ്ങളെക്കൊണ്ട്\\V_VAUX
അച്ഛന്റെ\\N_NN
ദേഹമതാ\\N_NNP
മാന്തിപ്പൊളിക്കുന്നു\\V_VM_VF
അയ്യയോ\\N_NN
ഞാൻ\\PR_PRP
പേടിച്ചുരണ്ടുനിലവിളിച്ചു\\V_VM_VF
.\\RD_PUNC

ഉടനെ\\RB
സത്വം\\N_NN
അച്ഛനെ\\N_NN
വിട്ട്\\V_VM_VNF
എന്റെനേർക്കു\\N_NN
തിരിഞ്ഞു\\V_VM_VF .\\RD_PUNC

Figure 3.2: Sample Dataset

3.4 Modules And Algorithms

3.4.1 Pre-Processing

Pre-processing step makes the sentence understanding easier and reduce complexity of other tasks performed on the sentence. This section explains two steps performed in pre-processing stage which is tokenization and POS tagging. Tokenization reduces the complexity of sentence structure and POS tagging helps to label text with in a sentence.

3.4.1.1 Tokenization

Tokenization[23] is the process of splitting up a sentence into small pieces known as tokens or words. Tokenization is performed based on the white space character with in the given sentence. It also removes unwanted or meaningless words or tokens from the sentence and keeps only the required words. Larger sentences can be divided into pieces of words by performing tokenization. By performing tokenization, instead of processing any task in entire sentence at a time task can be done in pieces of a sentence. After performing tokenization on the Malayalam sentence “ഇതു ഒരു സ്ഥലമാണ്” Then the resulting output will be: [‘ഇതു’, ‘ഒരു’, ‘സ്ഥലമാണ്’].

3.4.1.2 POS-tagging

Part-Of-Speech (POS) tagger [13] or grammatical tagger is used to label words with one of the several categories to identify the purpose of word in that sentence. In Malayalam there exist various categories of POS tags including noun, verb, adverb, adjectives etc. each tag contains a set of sub tags. The tag set used in this project is Bureau of Indian Standards (BIS) [14] which is developed by POS Tag standardization committee of Department of Information Technology. The BIS tag set is given in table 3.1.

POS tagging is also known as word category disambiguation which has two methods: rule based POS tagging and stochastic POS tagging. In rule-based POS tagging, the tags are assigned based on the rules defined by the user. For example, if there is a rule like “word ending with ‘ed’ or ‘ing’ must be assigned to a verb”. Then every words ending with ‘ed’ or ‘ing’ is assigned with verb instead of looking other things. One of the first POS tagger developed was E. Brill which is a rule based tagger.

Table 3.1: BIS Tag set

Category			Annotation
Top Level	Subtype	Subtype	
Noun			N
	Common		N_NN
	Proper		N_NNP
	<u>Nloc</u>		N_NST
Pronoun			PR
	Personal		PR_PRP
	Reflexive		PR_PRF
	Relative		PR_PRL
	Reciprocal		PR_PRC
	<u>Wh-word</u>		PR_PRQ
Demonstrative			DM
	Deictic		DM_DMD
	Relative		DM_DMR
	<u>Wh-word</u>		DM_DMQ
Verb			V
	Main		V_VM
		Finite	V_VM_VF
		Non-Finite	V_VM_VNF
		Infinite	V_VM_VINF
	Verbal		V_VN
	Auxiliary		V_VAUX
Adjective			JJ

Semantic Similarity of Malayalam Sentences

Adverb			RB
Postposition			PSP
Conjunction			CC
	Co-ordinator		CC_CCD
	Subordinator		CC_CCS
		<u>Quotative</u>	CC_CCS_UT
Particles			RP
	Default		RP_RPD
	Classifier		RP_CL
	Interjection		RP_INJ
	Intensifier		RP_INTF
	Negation		RP_NEG
Quantifiers			QT
	General		QT_QTF
	Cardinals		QT_QTC
	Ordinals		QT_QTO
Residuals			RD
	Foreign words		RD_RDF
	Symbol		RD_SYM
	Punctuation		RD_PUNC
	Unknown		RD_UNK
	Echo words		RD_ECH

In stochastic tagging, it combines both frequency and probability of words to tag. In frequency calculation method, the tag encountered most frequently in the training set with the word is the one assigned to an ambiguous instance of that word. But the problem with this method is that it may yield a valid tag for a given word, it can also yield inadmissible sequence of tag that is a word may assign valid or invalid tag. In the proposed method a statistical model named Hidden Markov Model (HMM) [15] is used. HMM is a variation of Markov Model, where the property of Markov Model states that the distribution for a random variable in the future depends only on its distribution in the current state and none of the previous states have any impact on the future states. Following terminologies are used in HMM POS tagging: set of states which are POS tags used, observations are word themselves. The initial state is the beginning of the sentence and the model use two probability named emission and transition probability. Emission probability is the probabilities of making certain observations given a particular state $P(w_i|t_i)$ where w_i is the i^{th} word and t_i is i^{th} tag. Transition probability is the probability of transitioning to another state given a particular state $P(t_i|t_{(i-1)})$ where t_{i-1} is the $(i - 1)^{th}$ tag. In HMM POS tagging it does not have any states, instead of that it only contains a sequence of observations. This is why the model is referred to as hidden since the actual states over time are hidden. HMM tagger assigns a tag by finding out probability of current word having a tag of VP given that the previous tag was NP.

(i). Training Algorithm

In the training process both transition and emission probabilities are calculated. For the training purpose of POS tagger CUSAT Malayalam tagged corpus [22] is used in which data in the following format:

പ്രഭാശരൻ\\N_NN ഒരു\\QT_QTO മനുഷ്യനു\\N_NN ഇല്ല\\RP_NEG .\\RD_PUNC.

9460 unique words are there in the training input file and the tags are assigned and evaluated manually by the students and faculties from CUSAT. From the training input set calculate 3 values: occurrence of each tag in that file, occurrence of (t_i, w_i) pair and occurrence of (t_{i-1}, t_i) pair. Then find out the transition probability for each (t_{i-1}, t_i) pair as:

$$P(t_{i-1}, t_i) = \frac{\text{total occurrence of } (t_{i-1}, t_i)}{\text{total occurrence of } (t_{i-1})} \quad (3.1)$$

Finally calculate emission probability of each (t_i, w_i) pair as:

$$P(t_i, w_i) = \frac{\text{total occurrence of } (t_i, w_i) + 1.0}{\text{total occurrence of } (t_i) + \text{no. of unique words in training file}} \quad (3.2)$$

Semantic Similarity of Malayalam Sentences

Write down both transition and emission probabilities into the model file. After training, a model file is generated which contains both transition and emission probabilities. At the time of tagging this file is used to load the model. A portion of model file after training is given in the figure 3.3:

```
T N_NN V_VM 0.000244798041616
T QT_QTF V_VAUX 0.00552486187845
T <s> JJ 0.0666722576317
T N_NNP RD_PUNC 0.0017825311943
E V_VM_VNF തൃജിച്ച 0.000242160067805
E V_VM_VF പതിപ്പിക്കണം 0.00025823111685
E N_NN മനംമാറ്റം 0.000181801654395
E N_NN ക്ഷേമത്തോടും 0.000181801654395
E JJ തുടർച്ചയായുള്ള 0.000258297817383
E N_NN ഉള്ളിലുള്ളതി 0.000181801654395
```

Figure 3.3: Model File Format

Algorithm3.1: HMM_POSTAG_TRAINING

Input: File contains data in the format: "പ്രഭാശത്ത്\\N_NN ഒരു\\QT_QTO മനുഷ്യനു\\N_NN ഇല്ല\\RP_NEG .\\RD_PUNC"

Output: Model file that contains transition and emission probabilities.

Step1: Create maps emit, transition and emission

Step2: For each line in the file perform the following steps

Step2.1: Mark sentence start as "<S>" and previous \leftarrow <S>

Step2.2: context[previous] \leftarrow context[previous]+1

Step2.3: split line into wordtags with space character " "

Step2.4: for each word tag in wordtags

Step2.4.1: Split wordtag into word, tag with "\\".

Step2.4.2: transition[previous tag] \leftarrow transition[previous tag]+1

Step2.4.3: context[tag] \leftarrow context[tag]+1

Step2.4.4: emit[tag word] \leftarrow emit[tag word]+1

Step2.4.5: previous \leftarrow tag

Step2.5: transition[previous </S>] \leftarrow transition[previous </S>]+1 where </S> indicate end of line

Step3: Open model file in write mode

Step4: For each key, value in the map transition

Step4.1: Split key into previous, word with space character " "

Step4.2: Write ("T" key value/context[previous]) into the model file

Step5: For each key, value in emit

Step5.1: Split key into previous, word with space character

Step5.2: Write ("E" key value/context[previous]) into the model file

Step6: Stop

(ii).Viterbi Algorithm

The problem of POS tagging is to find out best tag sequence T^* for a given word sequence S. Enumerating all possible tags sequence, compute their probability and choose the best one is a complex task. Since, if there are 'c' possible tags for each 'n' words then there are c^n possible tag sequences. So Viterbi algorithm [17] is used to find out the best path without explicitly enumerating all paths. It is a dynamic programming approach which stores partial output to avoid re-computation and find out best tag sequence for a given word sequence. This algorithm contains two main steps namely forward step and backward step. Forward step calculate best path to a node by using lowest negative log probability and backward step reproduce the path to get best one.

Semantic Similarity of Malayalam Sentences

Algorithm 3.2: POSTAGGING_VITERBI

```
Input: Malayalam sentence
Output: Sequence of tags corresponding to input list

Step1: Tokenize the sentence based on white space and generate a list, words.
Step2: Create empty list listoftags and listofwords
Step3: Previous←<S> and endofline←</S>
Step4: Find number of unique words in the training dataset and assign into
numofuniquewords
Step5: Load model file created from training and assign it into transition and emit
map accordingly
Step6: best_score[0 <S>]←0.0 and best_edge[0 <S>]←None
Step7: repeat step (8)-(9) for i in range of 0 to length of words
Step8: repeat step (9) for prev in listoftags
Step9: repeat following steps for next1 in listoftags
Step9.1:    best_score_key←str(i)+" "+prev, transition_key←prev+" "+next1 and
emit_key←next1+" "+words[i]
Step9.2:    if best_score_key in best_score.keys() and transition_key in
transition.keys() then go to step (9.3)
Step9.3:    if emit_key not in emit then perform following
Step9.3.1:    vocabsz←numofuniquewords, numoftimesymemittedats2←0.0 and
totalnumofsymemittedbys2←context[next1]
Step9.3.2:    emit[emit_key]←(numoftimesymemittedats2+1)/(totalnumofsymemit
tedbys2+voabsz)
Step9.4:    score←best_score[best_score_keys]-
math.log(transition[transition_key])-math.log(emit[emit_key])
Step9.5:    assign    best_score_key,    best_edge_key,    transprob_key    and
emitprob_key as (str(i+1)+" "+next1)
Step9.6:    if best_score_key not in best_score go to step (9.6.1) else go to step
(9.7)
Step9.6.1:    best_score[best_score_key]←score
Step9.6.2:    best_edge[best_edge_key]←str(i)+" "+prev
Step9.6.3:    transition_score[transprob_key]←transition[transition_key]
Step9.6.4:    emission_score[emitprob_key]←emit[emit_key]
Step9.7:    if best_score_key in best_score and best_score[best_score_key]>score
then
```

Semantic Similarity of Malayalam Sentences

```
Step9.7.1: best_score[best_score_key] ← score
Step9.7.2: best_edge[best_edge_key] ← (str(i) + " " + prev)
Step9.7.3: transition_score[transprob_key] ← transition[transition_key]
Step9.7.4: emission_score[emitprob_key] ← emit[emit_key]
Step10:    repeat following for tag in listoftags
    Step10.1: best_score_key ← str(wordslen) + " " + tag
    Step10.2: transition_key ← tag + " " + </S>
    Step10.3: if best_score_key in best_score.keys() and transition_key in
               transition.keys()
    Step10.3.1: append tag to listoftagsbeforeend
    Step10.3.2: append (best_score[best_score_key] -
                       math.log(transition[transition_key])) to listofprobtagsbeforeend
Step11:    assign best_score_key and best_edge_key as str(wordslen+1) + " " + </S>
Step12:    best_score[best_score_key] ← min(listofprobtagsbeforeend)
Step13:    best_edge[best_edge_key] ← str(wordslen) + " " + listoftagsbeforeend[listofprobtagsbeforeend
                                         index(min(listofprobtagsbeforeend))]
Step14:    best_edge_key ← str(wordslen+1) + " " + </S>
Step15:    next_edge ← best_edge[best_edge_key]
Step16:    append last best_score to bestscores
Step17:    repeat steps until next_edge == 0 <S>
Step17.1:  split next_edge into (position, tag), append tag to tags, append
            bestscore[next_edge] to bestscores, append transition_score[next_edge] to
            transitions and append emission_score[next_edge] to emissions
Step17.2:  next_edge ← best_edge[next_edge]
Step18:    return (tags.reverse())
Step19:    stop
```

3.4.2 Parse Tree Generation

Parsers are most important components of many natural language processing systems for natural language understanding, machine translation etc. parsers use syntax of the language to create parse tree. Parse tree is the most effective way to understand the structure of a language and various activities on the language become very easier. Parsing is performed to determine the syntactic structure of expression, natural language sentence etc. In parse tree, points are called nodes and each node has a label on it. The node at topmost level is called root and nodes at lower level are called leaves. The root node of parse tree for grammar G is the start symbol of that grammar and leaves are the terminal symbols of G. The intermediate nodes are non-terminals of the grammar G. parsers are of two types: top-down parser and bottom-up parser. In top-down parsing, the parser start from starting symbol of grammar G and travel towards leaf nodes which are terminals of G while in bottom-up parsing, parser start from leaf nodes and travel towards top level until it reaches to the starting symbol.

To reduce the processing complexity of Malayalam, produce tree structure of Malayalam sentence. To generalize the tree structure, the given sentences are parsed and generate parse tree of that sentences. To perform parsing RegexpParser [18] is used which is based on regular expression grammar of Malayalam.

3.4.2.1 RegexpParser

It is a grammar based chunk parser where a given Malayalam sentence is structured as Noun Phrase (NP) and Verb Phrase (VP). Chunking find out contiguous, non-overlapping spans of related tokens and group them together into chunks. The parser use

Semantic Similarity of Malayalam Sentences

tagged text to perform chunking and use tags to take chunking decision. A grammar is developed for Malayalam sentences to perform the parsing. Grammar is created simply based on the structure of various Malayalam sentences. The grammar generated for parsing is given in the below table 3.2:

Table 3.2: Regular Expression for Parsing

N: {<N_NN>*<N_NNP>*<N_NST>*
JJ: {<JJ>}
RB: {<RB>}
PR: {<PR_PRP>}
QT: {<QT_QTF QT_QTC QT_QTO>}
RP: {<RP_RPD RP_CL RP_INJ RP_INTF RP_NEG>?}
VM: {<V_VM>*<V_VM_VF>*<V_VM_VNF>*<V_VM_VINF>*
VP: {<V_VN>*<V_VAUX>*
VP: {<RB>?<VM><RP>?<VP>*<NP>*<VP>*<RP>?}
NP: {<QT>?<N NP>*<N PR><JJ>?<RP>?}

The tagged sentence obtained from the POS tagging module is converted

Semantic Similarity of Malayalam Sentences

into a pair of word and corresponding tag. That is:

“അവ\\N_NN ഇപ്പോളത്തന്നെ\\RB പഠിക്കും\\V_VM_VNF” is converted into
[(u'അവ','N_NN'),(u'ഇപ്പോളത്തന്നെ','RB'),(u'പഠിക്കും','V_VM_VNF')].

Chunking start with flat structure in which no tokens are chunked then each rule in the grammar is applied iteratively and once all the rules applied a chunk structure is returned. The developed grammar contains only 10 rules so the chunk structure having maximum of depth 10. It is a bottom-up parser in which parsing start from tagged tokens and iteratively grouped them as either NP or VP based on the rules created. Other tagged tokens which are not in the grammar are chunked separately. The iteration is repeated until it reaches starting symbol. After chunking the chunk structure of sentence is converted into tree format which makes processing of Malayalam easier. The parse tree of two Malayalam sentences given in figure 3.4.

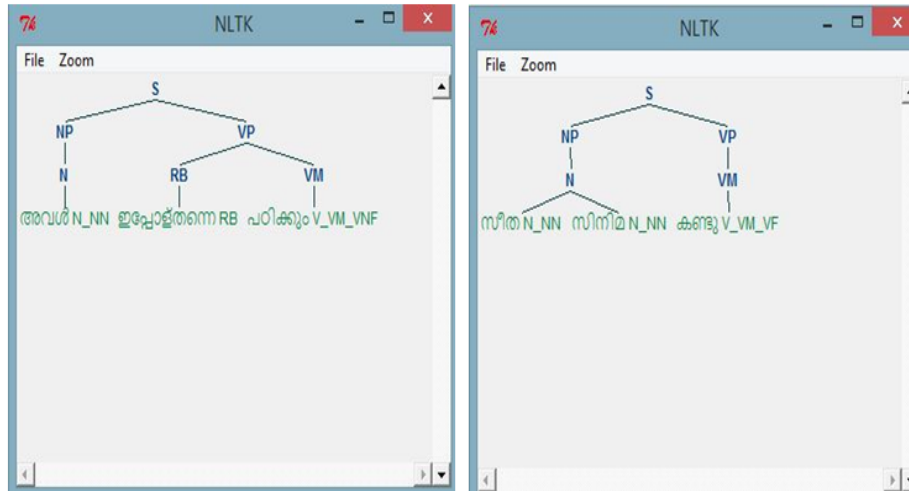


Figure 3.4: Parse Tree of Malayalam Sentences

Here S is the starting symbol of the grammar G, which is the root of tree. Leaves represent the tokens or single word of sentence along with its corresponding tag and the intermediate nodes are the non-terminals defined in G which are chunk label of group of tags.

3.4.3 Semantic Similarity Computation

This module compute the semantic similarity between the given two sentences by comparing the tree generated in the previous step. First it checks whether two trees are exactly same or not. If the trees have same structure and entries then score will be returned as 1 which means that the given sentences are semantically similar. Otherwise needs to check that each tokens in the trees are semantically similar or not by traversing the trees. By performing tree traversal nouns, verbs, adverbs, adjectives, negative words and other words are stored separately for both trees and perform similarity matching on each array. If the syntactic structure of words are different then need to access WordNet of Malayalam language. Before similarity checking stemming of words performed to get root words. But stemming of verbs and negation words are not performed in some scenario. While accessing WordNet it's easy to check whether meaning of two words are similar by looking up synonyms of words which are available from WordNet. If the synonyms are same then score value is updated.

The similarity computation start with setting total_score as 0 and for each matching total_score is updated by 1. Firstly it check whether the verbs of two sentences are same or not, because if verbs are different then the meaning of the sentences become entirely different otherwise there is a chance that the sentences having same meaning. Go to next step only if the verbs are same otherwise similarity becomes 0. If verbs are same then it checks the existence of any negative words in both sentences. If there is no such word then no score is updated and process will continue. If one sentence having negation then the similarity becomes 0, otherwise checks whether negation of sentences are same. If

they are same score is updated else similarity becomes 0.

After this it checks whether subjects and objects of sentences are same or not. For that first need to separate subject and object of each sentence from noun phrases. ‘Prathyayam rules’ are used to identify subject and object of a Malayalam sentence. Then check for subject and object matching separately and for each matching total_score is updated by one. Then checks whether the sentences having any adjectives and adverbs, if exist then compare it for matching purpose. Finally checks for the semantic similarity of other tokens in the sentence. After performing all these steps, it returned an integer value, total_score which is the total count of matching found between two sentences. Then total_score will be normalized [0, 1] by using the equation:

$$Sim(S_1, S_2) = \frac{total_score}{\max(len(S_1), len(S_2))} \quad (3.3)$$

S1 and S2 are the input sentences given for similarity checking, total_score is same as explained above. Then the similarity of two sentences will be getting by dividing total_score with the length of longest sentence from the input. So the score will be in between 0 and 1. If all the words in the sentences are matching semantically then Sim(S1,S2) is always 1 and it having value between 0 and 1 means that the sentences are not exactly same but they are paraphrases. If the sentences are not paraphrases or meaning of sentences are not matching then the score is always 0 and concluded that sentences are semantically different. The following sections explains different helping modules used in similarity computation.

Semantic Similarity of Malayalam Sentences

Algorithm 3.3: SIMILARITY CHECKING OF SENTENCES

Input: Parse trees T_1 and T_2 for sentences S_1 and S_2 respectively.
Output: float value which is similarity score between S_1 and S_2 .

Step1: Assign values to variables, $score \leftarrow 0$ and $l \leftarrow \max(\text{len}(S_1), \text{len}(S_2))$
Step2: If two trees T_1 and T_2 are same then go to step (3) else go to step (4)
Step3: Return score as 1 and exit
Step4: Perform tree traversal on both trees and store nouns, verbs, adjectives, adverbs, negation words and other phrases in separate list
Step5: Check if verbs are same then go to step (6) else go to step (7)
Step6: $score \leftarrow score + 1$ and go to step (9)
Step7: Perform stemming on each verb
 Step7.1: If stemmed verbs are same then go to step (7.2) else go to step (7.3)
 Step7.2: $score \leftarrow score + 1$ and go to step (9)
 Step7.3: If synonyms of stemmed verbs are same then, $score \leftarrow score + 1$ and go to step (9) else go to step (8)
Step8: Return score as 0 and exit
Step9: Check whether the sentence having negation word or not. If it having negation words then perform following steps, else go to step (10)
Step9.1: If only one sentence having negation, then return score as 0 and exit. Else go to step (9.2)
Step9.2: If negation words are same the $score \leftarrow score + 1$ and go to step (10), else return score as 0 and exit
Step10: Separate subjects and objects from list of nouns using Malayalam prathyayam rules
Step11: For all subjects repeat the following steps
Step11.1: If subjects are matching then $score \leftarrow score + 1$, else go to step (11.2)
Step11.2: Perform stemming and if they are same then $score \leftarrow score + 1$, else go to step (11.3)
Step11.3: If synonyms of stemmed words are same then $score \leftarrow score + 1$
Step12: Repeat step (11) for all list of objects, adverbs, adjectives and other phrases of both sentences and return the final score
Step13: Converge score in the range of [0,1] by $score \leftarrow \frac{score}{l}$
Step14: Stop

3.4.3.1 Malayalam Prathyayam Rules

An important step in semantic similarity is to check whether the subject and object of sentences are same or not. But both subject and object are grouped as noun phrase. So it is need to find out which are subject and object in noun phrase. For that prathyayam rule of Malayalam language [19] is used. Based on the suffix of word this rule will classify that word as subject or object.

Table 3.3: Malayalam Vibhakti Prathyayams

Vibhakti	Prathyayam	Example
Nirdheshika	Nil	അമ്മ
Prathigrahika	എ	അമ്മയെ
Samyojika	ഓട്	അമ്മയോട്
Udheshika	ക്ക്, ന്	അമ്മയ്ക്ക്
Prayojika	ആ, ക്കൊണ്ട്	അമ്മയെക്കൊണ്ട്
Sambhandhika	ഉട, ന്റെ	അമ്മയുടെ
Adharika	ഇ, ക	അമ്മയി

In Malayalam there are 7 vibhakties which is shown in the table 3.3 along with its suffix notation. This rule states that any word ends with sounds like ‘എ’, ‘ഓട്’ etc. then the word categorized as object and if any word ends with sounds like ‘ആൽ’, ‘ന്’, ‘ക്’,

‘ഒരു’, ‘ഉടെ’, ‘ഇത്’ etc. then this rule categorized the word as subject.

```
rules={u'നെ':u'എ', u'യെ':u'എ', u'ളെ':u'എ', u'ലെ':u'എ',  
u'ണെ':u'എ', u'ിനെ':u'എ', u'നോട്':u'ൊട്',  
u'യോട്':u'ൊട്', u'ളോട്':u'ൊട്', u'ലോട്':u'ൊട്',  
u'നോട്':u'ൊട്', u'രോട്':u'ൊട്', u'ന്':u'ന്', u'യ്ക്ക്':u'ക്ക്',  
u'നാൽ':u'ആൽ', u'യാൽ':u'ആൽ', u'ളാൽ':u'ആൽ',  
u'ലാൽ':u'ആൽ', u'രാൽ':u'ആൽ', u'ണാൽ':u'ആൽ', u'യാൽ':u'ആൽ',  
u'റെ':u'റെ', u'യുടെ':u'ഉടെ', u'ളുടെ':u'ഉടെ', u'ലുടെ':u'ഉടെ',  
u'ണുടെ':u'ഉടെ', u'രുടെ':u'ഉടെ', u'നിൽ':u'ഇൽ', u'ളിൽ':u'ഇൽ',  
u'ലിൽ':u'ഇൽ', u'ണ്ണിൽ':u'ഇൽ', u'ണിൽ':u'ഇൽ', u'രിൽ':u'ഇൽ',  
u'യ്യിൽ':u'ഇൽ', u'ല്ലിൽ':u'ഇൽ', u'യിൽ':u'ഇൽ',}
```

Figure 3.5: Malayalam Prathyayam Rules

3.4.3.2 Stemming

The process of getting stem or root of a given word by removing the suffix is called stemming [11]. Stemming is used in many of NLP tasks like IR, text summarization etc. In this rule based stemmer is used where rules are in the form of suffix and corresponding letter to replace. Stemming is avoided for negation words and some of the verbs.

Semantic Similarity of Malayalam Sentences

Stemming of verbs depends upon the suffix of word. Below table shows some of stemming rules used. For example by performing stemming on word ‘മരത്തിൽ’ will produce stem as ‘മരം’.

In this project stemming is done iteratively by looking one by one Unicode character from the end of given word. Before stemming there must define which of the words need not to perform stemming? At first it takes last Unicode character of given word and append it to a variable suffix and check for stemming rule with that character. If any rule found then check whether stemming is performed or not for the word. If the word not in the defined list then replaces the end character with the RHS of the rule. If no rule found then append next Unicode character from the end of word to the beginning of variable suffix and check for stemming rule. This will continue until and until a rule will found.

Algorithm 3.4: STEMMING

```
Input: Word to perform stemming
Output: Stemmed result of word

Step1: word_buffer  $\leftarrow$  "", suffix  $\leftarrow$  "", and new_ending  $\leftarrow$  ""
Step2: word_buffer  $\leftarrow$  word
Step3: repeat until (word_buffer == "")
    Step3.1: end_char  $\leftarrow$  word_buffer[len(word_buffer)-1]
    Step3.2: suffix  $\leftarrow$  end_char + suffix      #add end_char at the beginning of suffix
    Step3.3: word_buffer  $\leftarrow$  word_buffer[:-1] #remove end_char from word_buffer
    Step3.4: Search stem rule corresponding to suffix
    Step3.5: If stem rule founds go to step (4), else return to loop
Step4: Split rule into LHS and RHS
Step5: new_ending  $\leftarrow$  RHS
Step6: word_buffer.append(new_ending)
Step7: return word_buffer
Step8: Stop
```

3.4.3.3 IndoWordNet

Wordnets are lexical structures which contains synsets and semantic relations. Synsets are called synonyms. First wordnet in the world was constructed for English at

Semantic Similarity of Malayalam Sentences

Princeton University by George Miller and Christiane Fellbaum. IndoWordNet [21] is the wordnets of 18 scheduled Indian languages including Malayalam. The Malayalam wordnet was created by Amrita University Coimbatore, Tamil Nadu. A wordnet provide many informations like:

- Synonymy: links words that have similar meaning.
- Antonymy: links words that have opposite meaning
- Hypernymy: represents hierarchical relationships between words. For example vehicle is hypernymy of car.
- it also represents hierarchical relationships between words but opposite to hypernymy. That is car is hyponymy of vehicle.
- Meronymy: it refers to part of / whole relationship. For example paper is meronymy of book. That is paper is a part of book.
- Troponymy: semantic relationship of doing something in the manner of something else.
- Entailment: relationship between verbs where doing something requires doing something else.

In this project it needs only the synonymy of word. To access synonyms of words there exist a python package namely Pyiwn [16]. Pyiwn is an API for accessing Indian language wordnets in the IndoWordNet using python. The steps to setup pyiwn package are as follows:

Semantic Similarity of Malayalam Sentences

- Installation of pyiwn using pip:

```
pip install pyiwn
```

- This package is imported in python program as:

```
from pyiwn import pyiwn
```

- To download the IndoWordNet synset data:

```
pyiwn.download()
```

- Then choose the language for wordnet:

```
iwn = pyiwn.IndoWordNet('malayalam')
```

- Syntax to access synset of a particular word:

```
syn = iwn.synsets(word)
```

- Here only needs synonyms of the word. For that ‘.lemma_names()’ method is used.

Chapter 4

RESULTS AND DISCUSSION

This section presents the evaluation of semantic similarity method that is proposed. This section also explains the dataset collected for evaluation and for testing different categories of Malayalam sentences are considered.

4.1 Experimental Setup

The system for Malayalam sentence comparison is developed with the help of python programming language. It also uses NLTK, Pyiwn and other python packages for similarity calculation. For the testing of similarity method a dataset of pair of Malayalam sentences are required. The dataset creation explained in the following section.

4.1.1 Dataset Collection

Various Malayalam sentences are stored in an excel sheet where first and second column of the sheet are sentences for comparison and third column tells whether the

sentences are similar or not. The sentences labelled as P which means they are paraphrase or semantically similar and sentences that are labelled as NP which means they are not paraphrase or semantically different. The labelling of sentences is done manually. Some of the sentences for testing taken from publically available dataset [22] and other sentences are created manually. The testing set contains 100 sentence pairs in which 50 pairs are paraphrase and other 50 pairs are not paraphrase.

4.2 Sentence Comparison

To perform evaluation of sentences is taken randomly from the dataset prepared or normally input two sentences. Each pair of the sentences is given to the system for evaluation. Then produce the parsed tree output of a pair of sentences given above and corresponding tagged output and similarity score. From the result it can be analyse that the sentence that are exactly similar get score 1 and entirely different sentences will get score 0. The score between 0 and 1 means that the sentences share same concepts but indicating that some having more and some having lesser information about the concepts. The result will be also producing a pichart for better understanding.

Chapter 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

The proposed method which is used to find out how much similar the meaning of two Malayalam sentences via parse tree produces good results for simple as well as complex sentences. Tokenization makes sentence processing easier and for POS tagging Viterbi algorithm is used. Viterbi algorithm uses dynamic programming approach that reduces time consuming by storing results of each step which avoid re-computation of previous steps. The parsing process represents every sentence in a general structure which makes comparison easier. The stemming also important in similarity calculation which produces root words for word comparison. It produces the similarity score in the interval of 0 and 1. Different types of sentences are given for testing and it having higher accuracy for simple sentences whereas the accuracy of similarity calculation of complex sentences is lesser.

5.2 Future Scope

There are many possibilities for future works in the area of Malayalam semantic similarity. The proposed similarity method can be extended to document level by combining the sentence similarity. In this project the parsing chunks the sentence into group of noun phrase and verb phrase. So the parsing can be improved by grouping a sentence into various chunk groups in addition to noun and verb namely adjectives, adverbs etc. It makes the similarity measurement of complex sentences easier and will produce good results. In future by considering the tenses of sentences and by making a dependency between meaning of a word in a sentence with meaning of the previous word will improves the accuracy and efficiency of the system.

Chapter 6

APPENDIX

6.1 Source Code

6.1.1 UserApp

6.1.1.1 app.py

```
1 from flask import Flask , render_template , request , session , flash , redirect ,  
    url_for  
2  
3 import os  
4 import time , threading  
5 from os.path import dirname , abspath  
6 import unicodedata  
7 from tag import tagger  
8 import xlrd  
9 from itertools import zip_longest
```

```
10 from parse_tree1 import ptg
11 import json
12
13 app=Flask(__name__)
14
15
16 @app.route('/')
17 def index():
18     loadfolder=os.path.dirname(os.path.abspath(__file__))
19     finalfolder=os.path.join(loadfolder,'resultstore.json')
20     if os.path.isfile(finalfolder):
21         os.unlink(finalfolder)
22
23     return render_template("dashboard.html")
24
25
26
27
28
29 @app.route('/home')
30 def home():
31
32     loadfolder=os.path.dirname(os.path.abspath(__file__))
33     finalfolder=os.path.join(loadfolder,'resultstore.json')
34     if os.path.isfile(finalfolder):
35         print("aaaaaaaaaaaaaaaaaaaa")
36         os.unlink(finalfolder)
```

```
37
38     return render_template("adminhome.html")
39
40 @app.route('/analysis', methods=['GET', 'POST'])
41 def analysis():
42     if request.method == 'POST':
43         firstsent = request.form['text1']
44         secondsent = request.form['text2']
45
46         result = []
47         sentence = []
48         sentence.append(firstsent)
49         sentence.append(secondsent)
50
51         tagged_word = []
52         tagset = []
53         tagset1 = []
54         word = []
55         y = 0
56         inp = []
57         index = 0
58         for sent in sentence:
59             words = sent.split(' ')
60             for w in words:
61                 word.append(w)
62             temp = tagger(words)
63             for re in reversed(temp):
```

```
64         tagset.append(re)
65     i = 0
66     y = 0
67     s = []
68     w1 = []
69     for t in tagset:
70         if t != 'RD_PUNC':
71             s.append(t)
72             w1.append(word[y])
73             y = y+1
74     inp.append([(i, j) for i, j in zip_longest(w1, s)])
75     s = []
76     w1 = []
77     tagset = []
78     word = []
79
80     ptg(inp)
81
82     loadfolder=os.path.dirname(os.path.abspath(__file__))
83     finalfolder=os.path.join(loadfolder, 'resultstore.json')
84     if os.path.isfile(finalfolder):
85         jsontdata=json.load(open(finalfolder))
86         print(jsontdata)
87         ret_score=""
88         ret_statement=""
89         ret_dic=jsontdata[0]
90         ret_score=ret_dic['score']
```



```
91         ret_statement=ret_dic[ 'statement' ]
92         pichart=ret_score*100
93         pichart2=100-pichart
94         return render_template("finalresult.html",sent=sentence , score=
ret_score , statement=ret_statement , pichart=pichart , pichart2=pichart2 )
95     else :
96         return "retry!!"
97 if __name__=='__main__':
98
99     #app.run( host='0.0.0.0' , port=8000, debug=True )
100
101     app.secret_key = 'myappsecretkey'
102     app.run( debug=True )
103
```

Listing 6.1: code1

6.1.2 Main Code

6.1.2.1 main.py

```
1 import unicodedata
2 from tag import tagger
3 import xlrd
4 from itertools import zip_longest
5 from multiprocessing import Pool
6 from parse_tree1 import ptg
7 loc = (r'input.xlsx')
8 wb = xlrd.open_workbook(loc)
9 sheet = wb.sheet_by_index(0)
10 sheet.cell_value(0, 0)
11 result = []
12 sentence = []
13 tagged_word = []
14 tagset = []
15 tagset1 = []
16 word = []
17 y = 0
18 rows = sheet.nrows
19 # print(rows)
20 result.append(sheet.cell_value(0, 2))
21 # for j in range(1):
22 inp = []
23 for i in range(2):
24     sentence.append(sheet.cell_value(0, i))
25 print(sentence, "this is the checking")
```

```
26 index = 0
27 for sent in sentence:
28     # sent = sent.encode('utf8 ')
29     # print sent
30     '''sen = s.split(sent)
31     line=""
32     for ss in sen[0]:
33         ss = ss.strip()
34         line = line+" "+ss
35     line = line.strip().decode('utf8 ')'''
36     words = sent.split(' ')
37     for w in words:
38         word.append(w)
39     #fun1 tagger call
40     temp = tagger(words)
41     print(temp,"fun1 tagger return...")
42     for re in reversed(temp):
43         tagset.append(re)
44     print(sent,"**——**")
45     print("\n")
46     print(tagset)
47     print("\n")
48     i = 0
49
50     y = 0
51     s = []
52     w1 = []
```

```
53     for t in tagset:
54         if t != 'RD_PUNC':
55             s.append(t)
56             w1.append(word[y])
57             y = y+1
58     inp.append([(i, j) for i, j in zip_longest(w1, s)])
59     s = []
60     w1 = []
61     tagset = []
62     word = []
63     '''for t in tagset:
64         f.write(word[i].encode('utf8')+'\\'+t+" ")
65         i=i+1
66     f.write(";")
67     tagset1.append(tagset)
68     tagset=[]
69     tagged_word.append(word)
70     word=[]
71     #f.write('\n')'''
72
73 '''print tagset1
74 #print tagged_word
75 f.close()'''
76 ##f = open("output1.txt","r")
77 # for line in f:
78 #     line=line.rstrip(';')
79 #     s1,s2,s3,s4,s5,s6,s7,s8=line.split(';')
```

Semantic Similarity of Malayalam Sentences

```
80 # tree(s1)
81 # print "\n\n\n\n"
82 # tree(s2)
```

Listing 6.2: code2

6.2 Screenshot



Semantic Similarity of Malayalam Sentences

The screenshot shows a web application titled "semantic similarity". On the left is a sidebar with a "Dashboard" link and a "Semantic similarity" dropdown menu. The main content area has a breadcrumb "Home > semantic similarity checking...". Below this, there are two input boxes labeled "sentence 1" (with a blue header) and "sentence 2" (with a green header). Each box has a "Reset" button at the bottom. A red "check...." button is centered below the input fields. The footer of the application area says "Malayalam Semantic similarity ©".

This screenshot shows the same web application interface as above, but with Malayalam text entered in the input fields. The "sentence 1" box contains the text "രാജൻ സീതയെ ഇപ്രപഞ്ചിച്ചു .". The "sentence 2" box contains the text "സീത രാജനാൽ ഇപ്രപഞ്ചിക്കപ്പെട്ടു ↓". The "check...." button remains visible below the input fields. The footer still says "Malayalam Semantic similarity ©".

Semantic Similarity of Malayalam Sentences

The image displays two screenshots of a web application titled "Malayalam Semantic similarity".

Top Screenshot (Input Stage):

- The interface has a sidebar with "Home", "Dashboard", and "Semantic similarity" (selected).
- Two text input boxes are labeled "sentence 1" and "sentence 2".
 - Sentence 1 contains: "രാമൻ നീതിയെ ഉപേക്ഷിച്ചു ."
 - Sentence 2 contains: "നീതി രാമനാൽ ഉപേക്ഷിക്കപ്പെട്ടു ."
- Below the input boxes is a red button labeled "check...".
- At the bottom right, it says "Malayalam Semantic similarity ©".

Bottom Screenshot (Result Stage):

- The sidebar shows "Home" and "Result..." (selected).
- A "pi chart" is displayed, showing a red circle with "100%" in the center. A legend indicates a red dot for "match".
- A "Definition List" box contains:
 - sentence 1: രാമൻ നീതിയെ ഉപേക്ഷിച്ചു .
 - sentence 2: നീതി രാമനാൽ ഉപേക്ഷിക്കപ്പെട്ടു .
 - Analysis Score: similarity of given sentences is:1.0
- At the bottom right, it says "Malayalam Semantic similarity ©".

Chapter 7

REFERENCES

- [1] Abhimanyu Chopra, Abhinav Prashar and Chandresh Sain (2013) Natural Language Processing, in INTERNATIONAL JOURNAL OF TECHNOLOGY ENHANCEMENTS AND EMERGING ENGINEERING RESEARCH, vol 1, issue 5 ISSN 2347-4289.
- [2] Mallamma V. Redd and M Hanumanthappa (2014) Semantical and Syntactical Analysis of NLP, IJCSIT, Vol. 5 (3), 3236 – 3238.
- [3] H. Rubenstein and J. B. Goodnough (Oct. 1965) Contextual correlates of synonymy, Commun. ACM, vol. 8, no. 10, pp. 627-633.
- [4] G. A. Miller and W. G. Charles (1991) Contextual correlates of semantic similarity, Language Cognitive Processes, vol. 6, no. 1, pp. 1-28.

- [5] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman and Eytan Ruppin (Jan. 2002) Placing search in context: The concept revisited, *ACM Trans. Inf. Syst.* , vol. 20, no. 1, pp. 116-131.
- [6] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pasca and Aitor Soroa (2009) A study on similarity and relatedness using distributional and WordNet based approach, in *Proc. Human Language Technol. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, pp. 19-27.
- [7] Felix Hill, Roi Reichart and Anna Korhonen (2014) Simlex-999: Evaluating semantic models with (genuine) similarity estimation, *arXiv*: 1408.3456.
- [8] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio and Lucia Specia (2017) SemEval-2017 Task1: Semantic Textual Similarity-Multilingual and Cross-lingual Focused Evaluation, *arXiv preprint arXiv:1708.00055*.
- [9] Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi and Roberto Zamparelli (2014) A SICK cure for the evaluation of compositional distributional semantic models, In *LREC* 216-223.
- [10] Chris Quirk, Chris Brockett, and William Dolan (2004) Monolingual machine translation for paraphrase generation, In *Proceedings of the 2004 conference on empirical methods in natural language processing*.
- [11] Rinju O.R., Rajeev R. R., Reghu Raj P.C. and Elizabeth Sherly (October 2013) Morphological Analyzer for Malayalam: Probabilistic Method Vs Rule Based Method,

International Journal of Computational Linguistics and Natural Language Processing,
Vol 2 Issue 10 ISSN 2279 – 0756.

- [12] M Anand Kumar, Singh S., Kavirajan B., and Soman K. P., (2016) DPIL@FIRE 2016: Overview of shared task on detecting paraphrases in Indian Languages (DPIL), in CEUR Workshop Proceedings, vol. 1737, pp. 233-238.
- [13] Jisha P Jayan and Rajeev R R Parts Of Speech Tagger and Chunker for Malayalam – Statistical Approach, Computer Engineering and Intelligent Systems ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online) Vol 2, No.3.
- [14] Initish Chandra, Sudhakar Kumawat and Vinayak Srivastava (23rd March. 2014) Various tagsets for Indian languages and their performance in Part Of Speech Tagging, Proceedings of 5th IRF International Conference, Chennai, ISBN: 978-93-82702-67-2.
- [15] Nisheeth Joshi, Hemant Darbari and Iti Mathur (2013) HMM Based POS Tagger for Hindi, Computer Science Information Technology. pp. 341-349. 10.5121/csit.2013.3639.
- [16] Ritesh Panjwani, Diptesh Kanojia and Pushpak Bhattacharyya (January 2018) pyiwn: A Python-based API to access Indian Language WordNets, Center For Indian Language Technology, Indian Institute of Technology Bombay, India At GWC 2018, Singapore.
- [17] Manju K., Soumya S. and Sumam Mary Idicula (2009) Development of a POS Tag-

ger for Malayalam-An Experience, International Conference on Advances in Recent Technologies in Communication and Computing.

- [18] Jan Goyvaerts (2006) Regular Expressions: The Complete Tutorial, This book is available at <http://www.regular-expressions.info/print.html>.
- [19] Lectures on Malayalam grammar <https://unacademy.com>.
- [20] Malayalam language in general <https://simple.wikipedia.org/wiki/Malayalam>.
- [21] WordNet for Indian languages <https://en.wikipedia.org/wiki/IndoWordNet>.
- [22] Malayalam tagged corpus of Cusat <https://cs.cusat.ac.in/mlpos.jsp>.
- [23] Online tutorial on tokenization <https://www.guru99.com/tokenize-words-sentences-nltk.html>.