**SIMPLY5**

A SYSTEM FOR TEXT SIMPLIFICATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

OF THE DEGREE OF

BACHELOR OF ENGINEERING IN COMPUTER ENGINEERING

BY

MEGHNA PAI (14102A0004)

ROHIT KOTHAVADE (14102A0041)

AKSHAY DANGARE (14102A0046)

UNDER THE GUIDANCE OF

PROF. MOHINI KAMAT

DEPARTMENT OF COMPUTER ENGINEERING



VIDYALANKAR INSTITUTE OF TECHNOLOGY

WADALA(E), MUMBAI-400437

UNIVERSITY OF MUMBAI

2017-18

CERTIFICATE OF APPROVAL

This is to certify that the project entitled

“SIMPLY5 – A SYSTEM FOR TEXT SIMPLIFICATION”

is a bonafide work of

Meghna Pai (14102A0004)

Rohit Kothavade (14102A0041)

Akshay Dangare (14102A0046)

submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of

Undergraduate in “Bachelor of Engineering in Computer Engineering”.

Guide Head of Department Principal

Prof. Mohini Kamat (Dr. Arun Chavan) Dr. S.A.Patekar

PROJECT REPORT APPROVAL FOR B. E.

This project report entitled Simply5 – A System For Text Simplification by

1. *Meghna Pai (14102A0004)*
2. *Rohit Kothavade (14102A0041)*
3. *Akshay Dangare (14102A0046)*

is approved for the degree of

*Bachelor of Engineering in Computer Engineering.*

Examiners

1.---------------------------------------------

2. ---------------------------------------------

Date:

Place:

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name of student Roll No. Signature

1. Meghna Pai 14102A0004
2. Rohit Kothavade 14102A0041
3. Akshay Dangare 14102A0046

Date:

ACKNOWLEDGEMENTS

We take this opportunity to present our sincerest vote of thanks to all those guide posts who acted as lightening pillars to enlighten our way throughout this project which has led to the successful and satisfactory completion of this project.

Prima facie, we thank the Department of Computer Engineering, Vidyalankar Institute of Technology, Wadala for giving us the opportunity, and providing us with all the tools and the knowledge required to undertake this project and see it to completion.

We wish to express our deep gratitude and sincere thanks to our guide, Prof. Mohini Kamat for her invaluable guidance, constant encouragement, constructive criticism, diligent review and immense motivation, which enabled us to make sustainable progress without any breakdowns at every stage of this project’s development. Her invaluable advice and suggestions enabled us to make the corrections, modifications, and improvements, required to make this into a better project.

We would also like to express special gratitude to Prof. Pankaj Vanwari, who took time off his busy schedule to guide us through the project development stages and workout through any hurdle we were stuck at.

ABSTRACT

Thanks to the availability of texts on the Web in recent years, increased knowledge and information have been made available to broader audiences. However, the way in which a text is written—its vocabulary, its syntax—can be difficult to read and understand for many people, especially those with poor literacy, cognitive or linguistic impairment, or those with limited knowledge of the language of the text.

Texts containing uncommon words or long and complicated sentences can be difficult to read and understand by people as well as difficult to analyze by machines. Automatic text simplification is the process of transforming a text into another text which, ideally conveying the same message, will be easier to read and understand by a broader audience. The process usually involves the replacement of difficult or unknown phrases with simpler equivalents and the transformation of long and syntactically complex sentences into shorter and less complex ones.

Automatic text simplification, a research topic which started 20 years ago, now has taken on a central role in natural language processing research not only because of the interesting challenges it possesses but also because of its social implications. Combining lexical simplification and sentence generation is a challenge as it not only focuses on improving the understandability of the system but also readability by performing sentence generation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

* 1. Introduction to Natural Language Processing

Natural language processing is a field of artificial intelligence, computer science and linguistics concerned with the interaction between human and computers. It is related to the area of human computer interaction. It enables computers to derive meaning from natural language input or human which involves understanding of natural language processing. NLP is a process of developing a system that can process and produce language as good as human can produce. Natural language may be any regional language such as English, Hindi, and Marathi as opposed to programming languages, like C, C# C++, Java etc. Text simplification is one of the applications of NLP.

* 1. Text Simplification

Text Simplification (TS) is the process of modifying natural language to reduce its complexity and improve both readability and understandability. It may involve modifications to the syntax, the lexicon or both. The automation of this process is a difficult problem which has been explored from many angles since its conception in the nineties.

There has been much work in manual TS over the years, especially with a focus on second language learners, there is less work in automated simplification. The first effort towards automated simplification is a grammar and style checker developed for writers of simplified English. This was developed at Boeing for the writers of their commercial aircraft manuals, to help them keep in accordance with the ASD STE100 standard for simplified English. Further work to automate simplification for controlled language was undertaken. This was later extended for the case of general language in the areas of syntactic simplification and lexical simplification. These methods have heavily influenced future efforts to date. Work to improve the preservation of discourse in syntactic simplification and to improve the context–awareness of lexical simplification has been carried out.

TS is within the field of natural language processing. Within this field it is very similar to other techniques such as machine translation, monolingual text-to-text generation, text summarization and paraphrase generation. These fields all draw on each other for techniques and resources and many techniques within TS come from these other fields. TS is different to text summarization as the focus of text summarization is to reduce the length and content of input. Whilst simplified texts are typically shorter, this is not necessarily the case and simplification may result in longer output — especially when generating explanations. Summarization also aims at reducing content — removing that which may be less important or redundant. This is typically not explored within simplification, where all the content is usually kept.

Simplicity is intuitively obvious, yet hard to define. Typical measures take into account factors such as sentence length, syllable count and other surface text factors. Whilst these give a good estimate, they are not always accurate. For example, take the case of sentence length. One long sentence may use many complex terms and anaphora (words which refer to a previous entity: he, she, it, etc.). A simplified version of this would be lexically longer, but may be more explicative. In the case of explanation generation, complex words are appended with short definitions, increasing sentence length. Automatic heuristic measures will judge these sentences which have been simplified to be more complex. The final text may be longer, however it is also easier to understand and therefore simpler.

Whenever an individual goes through any article and he/she finds a difficult word in a sentence they either assume the meaning of that particular word or if they have time to spare, they refer to a dictionary. Even this might not solve their problem most of the times as in a dictionary for a particular word, all of its different meanings are mentioned whose use depends upon the context in which the word is used in the given sentence. It becomes confusing for a naïve or an intermediate user, sometimes even for an expert to select the appropriate meaning and frame it in the sentence. To reduce these efforts, a simplification system which can do all the above mentioned tasks at an instance of a single click could come real handy.

* 1. Text Simplification Methods

1. Lexical Simplification

It is a sub-task of text simplification. Lexical substitution generally refers to substitution of the word with its meaning which is a direct replacement.

Consider the following substitutions:

supports → backs motorists → drivers outraged → shocked

Issues with lexical simplification

It does not consider the semantics and the syntactic structure of the text.

1. Syntactic Simplification

Reading comprehension requires more than just knowledge of words and grammar. The reader also needs a cognitive base for language in order to construct a plausible meaning for a sentence. Considering the deaf children who face reading difficulties due to experiental and linguistic deficits incurred in their childhood usually learn to read with inadequately developed cognitive and linguistic skills. As syntactic analysis and semantic interpretation are constrained by same working memory that is required for storing information during a parse, lesser memory remains to process the “meaning” of the sentence. As a result, syntactic simplification helps to shorten or breakdown the complex sentences into smaller parts which are easy to remember and can also be used to pre-process the texts before feeding them to a full – blown parser.

1. Semantic Simplification

Semantic simplification or compression is the process of compacting a lexicon used to build a textual document by reducing the textual heterogeneity, while maintaining text semantics i.e. linguistics. As a result same idea can be represented using smaller set of words.

Issues with semantic simplification

Semantic compression is a lossy compression, that is, some data is being discarded, and an original document cannot be reconstructed in a reverse process.

* 1. Objective

Text Simplification helps an user to understand any difficult or complex piece of text by simplifying it. The overall goal is to provide a lexically, grammatically and contextutally simplified correct output for the given input complex sentence. Every individual has a different understanding level and capability to understand the meaning of the context so a tool which understands the context of a given sentence by itself, finds a correct substitution for the difficult words and provides a grammatically acceptable framework for the simplified output. The software will take a complex English sentence as a input, perform the above mentioned steps with the help of a pre-defined dictionary and create a simplified output for the same. Techniques till today focused on either lexical simplification or syntactic simplification or semantic compression alone but not all of them combined together.

* 1. Problem Statement

Every year lakhs of students appear for competitive exams like CAT, GRE, GMAT. These exams consist of Verbal section which has got a high weightage. Students take a lot of efforts to prepare for it as this Verbal section consists of higher level vocabulary words almost in every sentence. Students need to remember those thousands of words and try to understand the passage or a sentence and then solve the questions based on it. The idea is to develop a system which would make this learning process more comfortable and reduce the efforts. The input given will be a complex sentence. The system will scan it for difficult word and match those with the pre-defined dictionary, find a proper substitute and generate a grammatically correct and lexically simplified output.

If we think of today’s news articles, scientific journals, competitive exam preparation material, the level of vocabulary used is quite complex and each and every individual comes across some words which he/she cannot understand without using the dictionary. To reduce the task of going through the dictionary, searching the synonyms, finding the best suitable replacement; our system would provide a much better alternative wherein the user can just enter the entire piece of text i.e. a comprehension or a sentence into the input box and get a simplified version of the input at a click of a button. The output provided would not only be simple to understand but also would retain its proper grammatical structure.

* 1. Scope of the Project

The results of different text simplification techniques such as lexical, syntactic and semantic compression when performed individually are not very efficient and sensible as lexical simplification just performs substitution, syntactic simplification breaks the complex sentence into smaller parts and semantic compression reduces the number of words in a sentence. The existing systems perform either of the above methods at a time which is not very efficient. Also the output produced many a times is not very different from the input.

The proposed system accepts a single English sentence as an input. The scope of the project is now limited to a single sentence and manually fed inputs which can be extended to extract the inputs from a pdf or word document or through Optical Character Recognition. Instead of reading a complex sentence and understanding its meaning contextually by referring to a dictionary, such a system proves useful.

CHAPTER 2

MOTIVATION

Natural language processing has become an essential field of computer science. The applications of NLP are gaining more importance due to information overload. Different applications are text categorization, text summarization, text simplification, information retrieval, machine translation. Nowadays text simplification has become a booming field in NLP. Many a times students, after completing their undergraduate studies prefer going for higher studies for which they need to appear for exams like CAT, GRE, GMAT, etc. Verbal reasoning is a major part of these exams and have a high weightage. However, it might sound simple but this verbal reasoning is at a level higher than the one we use in our daily lives. It consists of complex vocabulary words, sophisticated sentences and passages, pieces of text from old English and what not. Students need to memorize these thousands of vocabulary words and each word might have different meanings depending upon the context in which it is used. For example, the word ‘crane’ has different meanings when used as noun, verb, etc. When used as a noun - They had to use a crane to lift the object. When used as verb - She had to crane her neck to see the movie.

The above example is a simple word. But when words like abnegation, anachronism, requiem, etc. come into picture, the users refer to a dictionary. But there they may find several meanings for a given particular word. So generally a user tries to fit in every meaning into the sentence and selects the one which fits correctly and then try to reframe the sentence which may or may not be correct grammatically. This steps incur a lot of efforts and time as well. So we decided to develop a system which would do these steps at just a click of a button. There are certain existing simplification systems which we referred and found out that they are not accurate at all to provide the desired output as required by a student.

Consider simplish.org which is an online tool for text simplification:

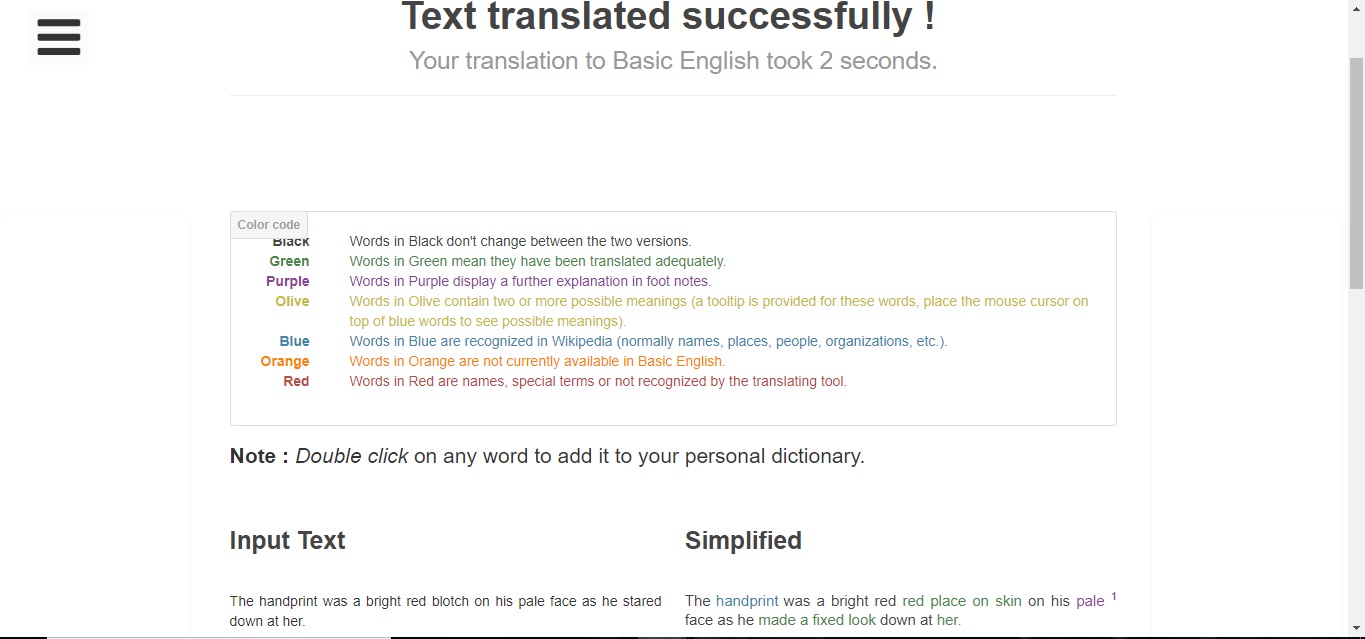


Figure 2.1 simplish.org [23]

As we can see the Input text is “The handprint was a bright red blotch on his pale face as he stared down at her.” And the output simplified sentence is “The handprint was a bright red red place on skin on his pale face as he made a fixed look down at her.” We can clearly see that the meaning of the word blotch has been directly substituted by its meaning without considering the later sentence framework which doesn’t sound grammatically proper. Such simplification is not very useful when it comes to understanding a sentence, finding synonyms and antonyms for a given word, or to refer to a passage and answer the questions based on it.

Rewordify is another online text simplification tool.

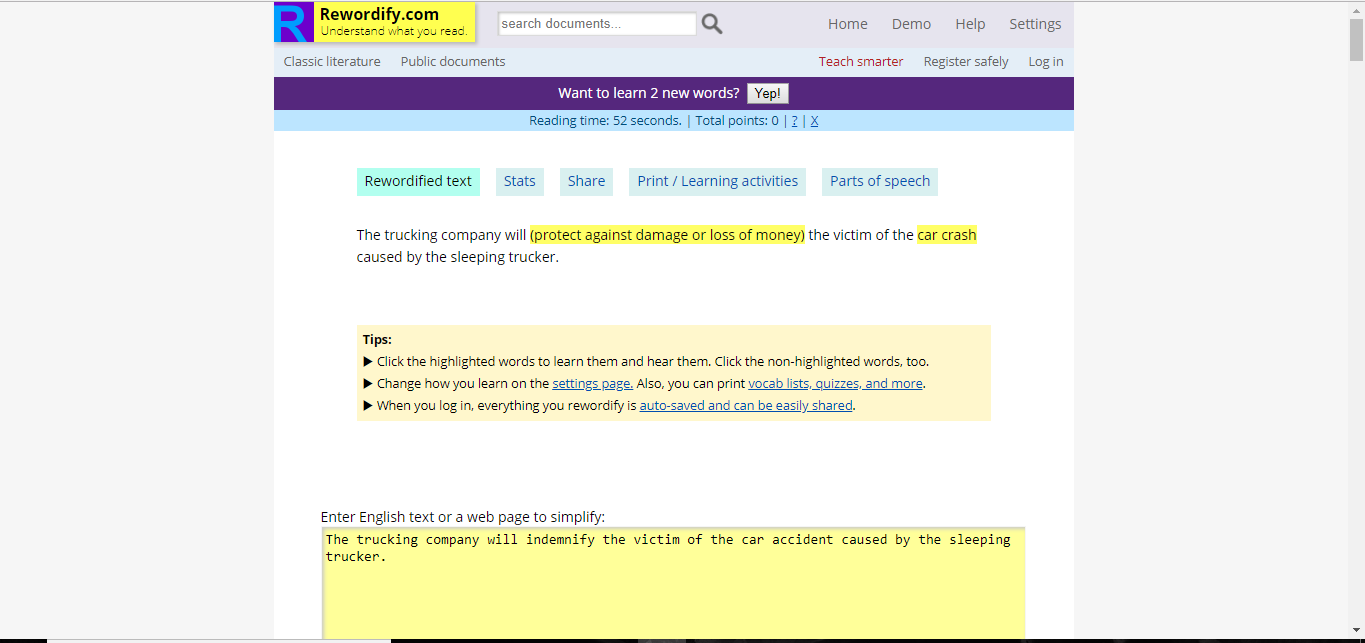


Figure 2.2 Rewordify text simplification system [26]

The tool identifies the word “indemnify” to be a difficult one and replaces it with its meaning. But the output losses its structure after the simplification process. This tool gives much better results as compared to simplish.org but it still neglects the problem of maintaining the tense and semantics of the input sentence,

From the above example it is clear that a much more accurate and efficient system is needed to solve the text simplification issues. There lies a motivation for proposing and designing a system which takes an input from the user and gives an output in the format as desired by the user – easy to understand and with minimum grammatical errors.

CHAPTER 3

REVIEW OF LITERATURE

Text Simplification refers to the process of converting complex words to simpler words in a sentence increasing the scope of understanding the context by a layman.

|  |  |  |
| --- | --- | --- |
|  | Paper Title | Description |
| [1] | R.Chandrasekar, Christine Doran, B.Srinivas “Motivations and Methods for Text Simplification”. Article published in a journal in Research Gate,1998. Copenhagen, Denmark — August 05 - 09, 1996 | It focuses on the reason for Text Simplification. It states that if texts are made simpler, sentences become easier to process for both programs and humans. It discusses a process wherein the components of the sentences that can be separated out and transforms each of those into free-standing simpler sentences. |
| [3] | Siddharthan, Advaith. "A survey of research on text simplification." ITL-International Journal of Applied Linguistics 165.2 (2014): 259-298. | This paper focuses on distinguishing the text simplification systems based on the target readers. It includes factors like age groups, different kinds of disabilities like dementia, deafness, dyslexia, etc. This paper is the main motivation for our selection of target audience. It explicates the use of text simplification for Reading Comprehensions for learners. |
| [5] | Shardlow, Matthew. "A survey of automated text simplification." International Journal of Advanced Computer Science and Applications 4.1 (2014): 58-70. | In this paper the different approaches used for Text Simplification namely Lexical Approach, Syntactic Approach, Explanation generation, Statistical Machine Translation and Non-English Approaches are explained. The research challenges that need to be overcome in the future are explained in detail. It includes-  Resources, Systems and Techniques. |
| [15] | Arendse Bernth “EasyEnglish: A Tool for Improving Document Quality”. Proceeding ANLC '97 Proceedings of the fifth conference on Applied natural language processing Pages 159-165 | It discusses a tool named EasyEnglish by IBM that helps writers produce clearer and simpler English by pointing out ambiguity and complexity. This tool consists of about forty checks such as grammar checker, spell checker, word count etc. It states that controlled language have been invented to solve the problems associated with readability and translatability. |
| [9] | Phil Katz,“Supervised Word Sense Disambiguation using Python” Research Gate, 2005. | In this paper the information about Word Sense Disambiguation is discussed. It is considered to be one of the major approaches to solve the lexical sample problem. Different methods to implement WSD such as Naïve Bayes, Cosine comparison of word-frequency vectors, decision lists and Latent Semantic Analysis. This paper implements a simpler classifier combination system that combines these classifiers into one module. |
| [11] | Robin Keskisärkkä “Automatic Text Simplification via Synonym Replacement” Robin Keskisärkkä, 2012 | In this paper lexical simplification for Swedish language is investigated. Three different strategies – word frequency, word length and level of synonymy are discussed. The system discussed in this paper describes it as a two stage system, where the text is first analyzed using a lexical tagger, a morphological analyzer and a parser and then passed to a simplifier. |
| [13] | Gondy Leroy “User Evaluation of the Effects of a Text Simplification Algorithm Using Term Familiarity on Perception, Understanding, Learning and Information Retention”. Journal of Medical Internet Research, 2013. | This paper describes the user evaluation with an independent writer of an automated simplification algorithm using term familiarity. Term familiarity indicates how easy words are for readers and is estimated using term frequencies in the Google Web Corpus. The authors conducted a controlled user study with a representative writer who used their simplification algorithm to simplify texts. They tested the impact with representative consumers. Ninety-nine participants completed the study. They found strong beneficial effects on both perceived and actual difficulty. After simplification, the text was perceived as simpler with simplified text scoring 2.3 and original text 3.2 on the 5-point Likert scale (score 1: easiest). It also led to better understanding of the text Swith 11% more correct answers with simplified text (63% correct) compared to the original (52% correct). |
| [18] | S. Rebecca Thomas and Sven Anderson “WordNet-based Lexical Simplification of a Document”. Proceedings of KONVENS, 2012. | This report explores algorithms for generation of lexicon from a document such that the lexicon covers as much as semantic space of the original document. It mainly focuses on the word sense disambiguation and the ranking within the synsets. The lexical choice decisions made are for the entire document and not for a single sentence. It combines Disambiguation step and lexicon reduction step to evaluate six different algorithms to create lexicons of pre-specified size. |
| [7] | Siddharthan, Advaith. "Syntactic simplification and text cohesion." Research on Language & Computation 4.1 (2006): 77-109. | This paper explains the different text simplification techniques including Lexical, syntactic and semantic. It manly focuses in the syntactic simplification as the importance of shorter versions of complex sentences in human life as well as for parsing complex sentences is described well in the paper. |
| [10] | Jonas EKEDAHL and Koraljka GOLUB “Word sense Disambiguation using WordNet and the Lesk algorithm”. 2004 | This paper describes the process of Word Sense Disambiguation whose popularity has increased in the last two decades. It focuses on the Lesk approach of performing the Word Sense Disambiguation considering different senses which are derived from WordNet which is a huge lexical database. |

Table 3.1 Literature Survey table

CHAPTER 4

PROPOSED SYSTEM

The idea is to pass an English sentence as an input to the system and to get a lexically simplified and grammatically correct output. This process consists of following steps:

1. Tokenization of input sentence
2. Part-of-Speech (POS) Tagging
3. Filtration of required words
4. Stemming the filtered words
5. Matching the words with the dictionary through Indexed Sequential Search
6. Performing Word Sense Disambiguation to select the accurate meaning based on context of the input. Modified LESK Algorithm is used for the same.
7. Using the designed Rulebase to perform Sentence generation.

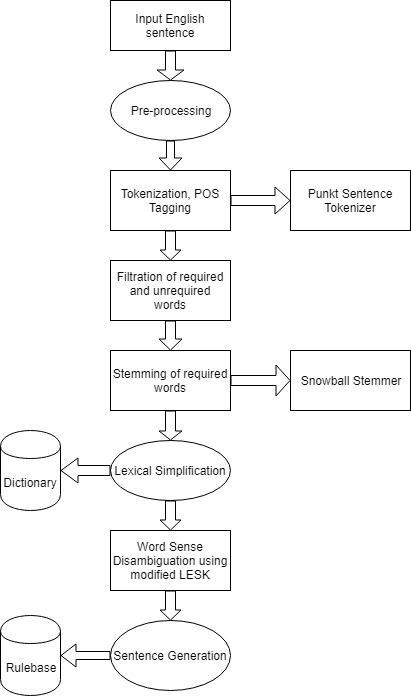


Figure 4.1 Proposed System

First step is to pre-process the input. For each word in the input perform tokenization and Part-of-Speech Tagging. The tags which are considered to represent the difficult words most of the times, are then filtered. These words are then stemmed to their base forms. The words occurring in the dictionary and those which are generated from the stemmed words are checked in the code and if the appropriate word is found in the dictionary, its meanings are taken from WordNet which is a global dictionary designed for performing NLP tasks. Word Sense Disambiguation is performed on those meanings using modified LESK algorithm and depending upon the context accurate meaning is chosen most of the times. This meaning is placed as it is at the place of the word. This completes the Lexical simplification part.

Second step is to perform sentence generation with the replaced meaning so as to make the sentence grammatically appropriate considering the tenses, stopwords, articles, etc. This is the most important step in the project which will make our system different from the already existing systems. This process is based on the rulebase which is created considering the grammar rules we have identified by reading various articles and grammar reference books.

The overall algorithm for the system is discussed below.

Input: English sentence having difficult vocabulary.

Output: Lexically and grammatically simplified sentence.

Accept the English sentence as input

For the complete sentence do

Word tokenization

POS-Tagging

For all the POS-Tagged words

Filter the required category of word

Stem the filtered words

For each stemmed word

Scan the dictionary to find the complete word

If found then

Search the meanings for the word on WordNet

Perform Word Sense Disambiguation using LESK

Substitute the selected meaning at the place of the difficult word

else

Keep the word as it is

Perform Sentence generation based on the rules

Stop.

The above mentioned algorithm can be represented as a three phase process as follows:

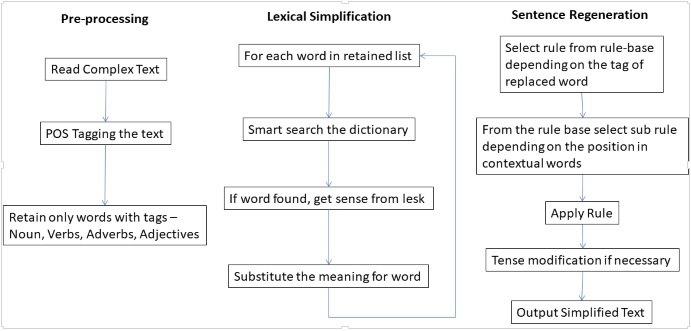


Figure 4.2 Three phase System

**4.1 Dictionary creation**

The dictionary has been created by scraping data from various websites using python data scraping and panda library. The dictionary contains around 4000 words arranged alphabetically. This dictionary is the main data source for scanning the occurrences of difficult words.

**4.2 Lexical Simplification Phase**

This phase scans the input sentence, tokenizes the sentences into words, performs POS-Tagging on the words. Depending upon the required POS tags the words are filtered as the ones that have high probability of being difficult and the ones that have least chances of being a difficult one. These words are then stemmed to their base forms as the word in the given sentence and a word in the dictionary may not match exactly i.e. can be in different forms. Once the word is matched, the meanings for that word are searched from WordNet. Word Sense Disambiguation is performed and the appropriate meaning considering the context of the sentence is selected to be substituted in place of the difficult word.

**4.2.1 Pre – processing module**

This phase accepts the input as an English sentence. In pre-processing, the entire sentence is tokenized into words. Once tokens are generated, the next step is to identify part of speech tag for each word and assign the part of speech to every word such as noun, verb, adjective, etc. These tags are useful in the filtration process of words. Next task is to perform Stemming on the filtered words. This is necessary because the word found in a sentence and the one present in a dictionary may not match spelling wise, but can be in different forms. Stemming is necessary so as to not miss out on simplifying any difficult word. Pre-processing consists of three main processes:

Tokenization, Part of Speech Tagging (POS) and Stemming.

Steps for pre-processing module:

1. Tokenization: Accept the input sentences and generate the tokens for entire text. Tokenization is a process of breaking a stream of sentences into tokens. This is done by searching a space after each word. The list of tokens becomes input for further processing such as part-of-speech tagging and named entity recognition. Tokenization is the task of chopping given a character sequence and a defined document unit it up into pieces, called as tokens and discarding certain characters such as punctuation. It locates word boundaries. End point of a word and beginning of the next word is called as word boundary. It is also called as word segmentation.
2. Part of Speech tagging (POS) and Filtration: Part-of-speech tagging is one of the most important text analysis tasks used to classify words into their part-of-speech and label them according the tagset which is a collection of tags used for the pos tagging. Part-of-speech tagging also known as word classes or lexical categories. The default POS tagger model using in NLTK is maxent\_treebank\_pos\_tagger model. These tagged words are then classified as difficult category words and non-difficult category of words depending upon the study performed.
3. Stemming: Stemmer, in stemming, is the automated process which produces a base string in an attempt to represent related words. A stemmer is an algorithm that operates on the principle of recognizing “stem” words embedded in other words. These are helpful for lexical purposes, for example, in online dictionaries, for heuristics in file management, or anywhere else that semantic tools can help create order. Stemmers pick up the inclusion of a core or stem word within a longer word. For example, a stemming algorithm might look at a word like “running” and correctly recognize that the root word or stem word is “run”. This can be a helpful element of something that parses raw text for analysis.

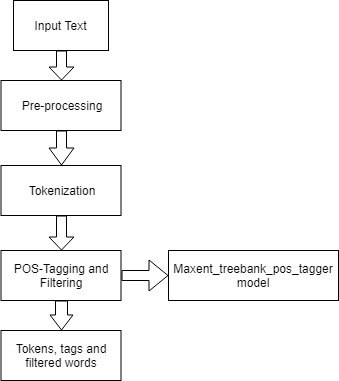


Figure 4.3 Pre-processing Module

Algorithm for pre-processing module:

Input: English sentence with difficult vocabulary

Output: Tokens, POS tags and stemmed filtered words

Accept the difficult English sentence as an input

Generate tokens for the entire text

For each token

Apply POS-Tagging and generate POS tags for all the tokens

For each POS-tag

Filter the tags that come under the category of nouns, verbs, adjectives and adverbs and save them in an array

For each word in the array

Apply stemming

Stop

1. Tokenization:

Algorithm:

Accept the input document

From beginning scan the input document and identify the white space

If space is found then the word before the space/ word between two spaces are identified as a token

Repeat the process until all the text is considered.

Input: After Katie exposed the church's secrets, she was regarded as an apostate.

Tokens generated: After, Katie, exposed, the, church’s, secrets, , , she, was, regarded, as, an, apostate, .

1. Part-of-Speech tagging:

Below table 4.1 lists few popular Penn Treebank POS tags which are used by many applications. There are total 36 tags available in Penn Treebank POS tag set. The proposed system uses Penn Treebank tag set.

Table 4.1 Part of speech tags which are considered to represent difficult words most of the times. Selected from Penn Treebank:

|  |  |  |
| --- | --- | --- |
| Number | Tag | Description |
| 1 | JJ | Adjective |
| 2 | JJR | Adjective, comparative |
| 3 | JJS | Adjective, superlative |
| 4 | NN | Noun, singular or mass |
| 5 | NNS | Noun plural |
| 6 | RB | Adverb |
| 7 | RBR | Adverb, comparative |
| 8 | RBS | Adverb, superlative |
| 9 | VB | Verb, base form |
| 10 | VBD | Verb, past tense |
| 11 | VBG | Verb, gerund/present participle |
| 12 | VBN | Verb, past participle |
| 13 | VBP | Verb, singular, present, non 3rd |
| 14 | VBZ | Verb, 3rd person singular present |

POS-Tagging:

('After', 'IN') ('Katie', 'NNP') ('exposed', 'VBD') ('the', 'DT') ('church', 'NN')

("'s", 'POS') ('secrets', 'NNS') (',', ',') ('she', 'PRP') ('was', 'VBD') ('regarded', 'VBN')

('as', 'IN') ('an', 'DT') ('apostate', 'NN') ('.', '.')

1. Filtration:

Comparing the above tags with the above mentioned table, following words are considered to be scanned for being a difficult word: “exposed”, “church’s”, “secrets”, “regarded”, “apostate”

1. Stemming:

The above filtered words are then stemmed using Snowball Stemmer which is an inbuilt module in python NLTK. The stemmed words are as follows:

exposed = expos

church’s = church

secrets = secret

regarded = regard

apostate = apost



Figure 4.4 Stemming

**4.2.2 Searching Phase**

The stage that comes after pre-processing wherein we get the stemmed words is scanning the dictionary and finding the words which represent the original words which were stemmed. Now a normal scan function would require to go through the entire dictionary to find if one particular word exists or not. This will be time consuming.

Indexed Sequential Search:

In this search method we provide the beginning index of each alphabet. The initial alphabet is matched with the index provided for that particular alphabet. This technique saves searching time and reduces the time complexity.

**4.2.3 Word Sense Disambiguation**

Input:

After Katie exposed the church's secrets, she was regarded as an apostate.

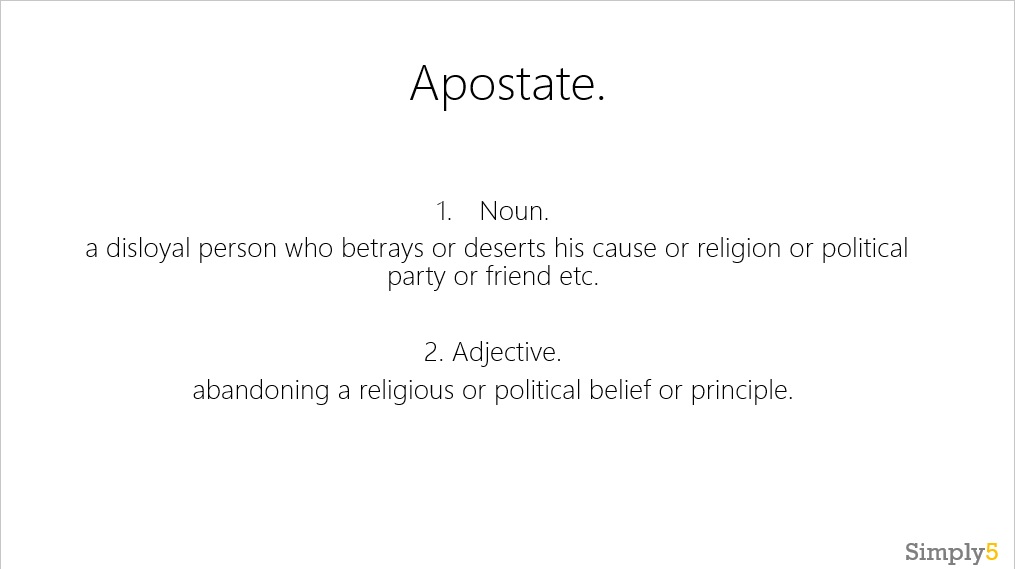


Figure 4.5 Word Sense Disambiguation

Word matched in the dictionary: apostate

Word sense disambiguation is the process of automatically clarifying the meaning of a word in its context. For example, the word contact can have nine different senses as a noun, and two different senses as a verb.

Once we reach the index from which the list of the words starting with the difficult word’s initial alphabet begins we need to search for the accurate word from the dictionary. Now it is not necessary that the words found in the input will directly match with any word in the dictionary. So let’s stem the words in the dictionary and match them with stem of the word from the input. This works a little better than direct matching, but many words may have the same stem as that of the input word. This will result in an incorrect search and will give incorrect meaning.

The solution to this is to match the occurrences of the stem of the input word and the same stems from the dictionary depending upon the context of the input. This is done using WordNet which is the biggest dataset for performing many NLP operations. The pair which has maximum occurrences is selected. This solves the Word Sense Disambiguation problem and we get a meaning according to the context of the input sentence. This overall process is performed by using Extended LESK algorithm.

LESK algorithm:

The Lesk algorithm has first been implemented in its simple form by M. Lesk (1986). It is based on the assumptions that when two words are used in close proximity in a sentence, they must be talking of a related topic and, if one sense can be used by each of the two words to refer to the same topic, then their dictionary definitions must use some common words. This approach involves looking for overlap between the words in dictionary definitions with words from the text surrounding the word to be disambiguated.

The problem of this approach is that dictionary definitions often do not have enough words for this algorithm to work well, which can be overcome by using the WordNet lexical database, because it contains different types of relationships between words, such as, for example, synonymy and hyper/hyponymy. This modified version is the Extended Lesk Algorithm. In this algorithm, the words from the input which are to be disambiguated are extracted along with the senses of surrounding words. The surrounding words can be nouns, adjectives, verbs or adverbs. If none of these are found in the WordNet, the next closest sense is selected.

Every sense of the word to be disambiguated is compared to each sense of the surrounding words. A number of combinations is derived and scores are assigned to them, based on the number of the overlapping words. For example, if a word to be disambiguated has two senses, and it is surrounded by two words, one having three different senses, and the other having two different senses, the number of derived combinations is 12, out of which six are for the first sense of the word to be disambiguated, and the other six are for the second sense of the word to be disambiguated. The sense chosen is the one in which group of six there is the combination with the highest score out of all the 12 combinations.



Figure 4.6 Lesk Algorithm

**4.2.4 Substitution**

As the Lesk algorithm returns the correct sense of the word depending upon the context in which it is used in the input sentence, we replace that word with the meaning from the WordNet.

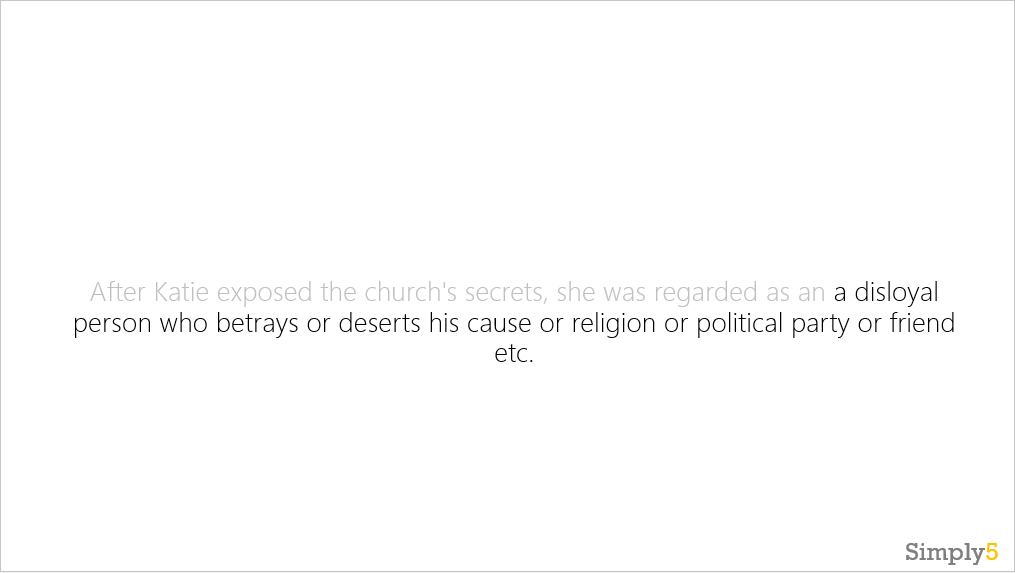


Figure 4.7 Lexical Simplification

This process completes our Lexical substitution.

**4.3 Sentence Generation**

Sentence generation, also called as Natural Language Generation (NLG) in the field of Natural Language Processing, is the computational process of automatically producing sentences in natural language based on the type of input it receives. A sentence generation component receives as input a specification of what it is supposed to communicate, and produces as output a corresponding natural language expression. In most current approaches to sentence generation, such input is either a logical form or a set of facts in some knowledge representation language, or a set of closely related words. Thus, a generation component might accept as input a set of words such as(on), (book), (red), (table) and convert this into something like the natural language expression "The red book is on the table." The end result of such a generation process is relatively clear: a sentence must be produced which is grammatically correct and which has made appropriate choices of words (or lexemes). [10] Coming to the output produced in the example mentioned in the Lexical Simplification phase, the substitution performed for ‘reincorporation’ wasn’t grammatically correct. Hence, we need to use the sentence generation process to convert“have move back to an original place at offshore” “have moved back to an original place at offshore” As correctly identified, the problem in the first sentence was the tense of the verb “move”, as it was directly replaced from the meaning and needed modification. Sentence generation thus identifies such words which require subtle modifications into order to fit into the existing sentence, for the sentence as a whole to be correct grammatically.

CHAPTER 5

IMPLEMENTATION

**5.1 Word Sense Disambiguation**

from nltk.corpus import wordnet as wn

def lesk(context\_sentence, ambiguous\_word, pos=None, stem=True, hyperhypo=True):

max\_overlaps = 0; lesk\_sense = None

context\_sentence = context\_sentence.split()

for ss in wn.synsets(ambiguous\_word):

# If POS is specified.

if pos and ss.pos is not pos:

continue

lesk\_dictionary = []

# Includes definition.

lesk\_dictionary+= wn.synsets(ambiguous\_word)[0].definition()

# Includes lemma\_names.

lesk\_dictionary+= ss.lemma\_names()

# Optional: includes lemma\_names of hypernyms and hyponyms.

if hyperhypo == True:

lesk\_dictionary+= list(chain(\*[i.lemma\_names() for i in ss.hypernyms()+ss.hyponyms()]))

if stem == True: # Matching exact words causes sparsity, so lets match stems.

lesk\_dictionary = [stemmer.stem(i) for i in lesk\_dictionary]

context\_sentence = [stemmer.stem(i) for i in context\_sentence]

overlaps = set(lesk\_dictionary).intersection(context\_sentence)

if len(overlaps) > max\_overlaps:

lesk\_sense = ss

max\_overlaps = len(overlaps)

return lesk\_sense

**5.2 Identifying the tense of the input sentence**

def get\_tense(original,verb,suffix):

tense='none'

pos=original.index(verb)

pos\_tagged3=nltk.pos\_tag(original)

if suffix=='ing':

if original[pos-2]=='have' or original[pos-2]=='has' and original[pos-1]=='been':

tense="present\_prefect\_continous"

if original[pos-2]=='had' and original[pos-1]=='been':

tense="past\_perfect\_continous"

if original[pos-3]=='will' and original[pos-2]=='have' and original[pos-1]=='been':

tense="future\_perfect\_continous"

if original[pos-2]=='will' and original[pos-1]=='be':

tense="future\_continous"

if original[pos-2]=='was' or original[pos-2]=='were':

tense="past\_continous"

else:

tense="present\_continous"

elif pos\_tagged3[pos][1]=='VBN':

if original[pos-1]=='have' or original[pos-1]=='has':

tense="present\_perfect"

if original[pos-1]=='had':

tense="past\_perfect"

if original[pos-1]=='will' and original[pos-1]=='have':

tense="future\_perfect"

else:

tense="past\_perfect"

elif pos\_tagged3[pos][1]=='VB':

if original[pos-1]=='will':

tense="simple\_future"

else:

tense="simple\_present"

elif pos\_tagged3[pos][1]=='VBD':

tense="simple\_past"

return tense

**5.3 Rulebase**

**5.3.1 Rules for Adjectives**

if part\_of\_speech == 's' or part\_of\_speech == 'a':

#single\_phase meaning

if meaning.find(' ')==-1:

new=replace(original,word,meaning)

else:

pos\_tagged=nltk.pos\_tag(original)

adj\_pos=original.index(word)

noun\_pos=0

if pos\_tagged[adj\_pos-1][1]=='NN' or pos\_tagged[adj\_pos-1][1]=='NNS':

noun\_pos=-1

new=replace(original,word,meaning)

elif pos\_tagged[adj\_pos+1][1]=='NN' or pos\_tagged[adj\_pos+1][1]=='NNS':

noun\_pos=1

if pos\_tagged[adj\_pos+2][1]=='NN' or pos\_tagged[adj\_pos+2][1]=='NNS':

noun1=pos\_tagged[adj\_pos+1][0]

noun2=pos\_tagged[adj\_pos+2][0]

new=replace(original,noun2,meaning)

new=replace(original,noun1,noun2)

new=replace(original,word,noun1)

else:

noun=pos\_tagged[adj\_pos+1][0]

new=replace(original,noun,meaning)

new=replace(original,word,noun)

else:

new=replace(original,word,meaning)

**5.3.2 Rules for Nouns**

elif part\_of\_speech == 'n':

noun\_pos=original.index(word)

pos\_tagged=nltk.pos\_tag(original)

noun\_nltk\_pos = pos\_tagged[noun\_pos][1]

noun=meaning

#single\_phase meaning

if meaning.find(' ')==-1:

# singular-plural conversion

if noun\_nltk\_pos=='NN':

noun=to\_singular(meaning)

elif noun\_nltk\_pos=='NNS':

noun=to\_plural(meaning)

# article check

if original[noun\_pos-1] == 'a' or original[noun\_pos] == 'an':

import inflect

p = inflect.engine()

prev=p.a(meaning)

prev=prev.split(' ')

original[noun\_pos-1]=prev[0]

new=replace(original,word,noun)

#phrase\_meaning

else:

meaning\_list=meaning.split(' ')

meaning=remove\_stopwords(original,meaning\_list,word)

meaning\_list=meaning.split(' ')

# article check

if original[noun\_pos-1]=='an' or original[noun\_pos-1]=='a':

import inflect

p = inflect.engine()

prev=p.a(meaning\_list[0])

prev=prev.split(' ')

original[noun\_pos-1]=prev[0]

new=replace(original,word,meaning)

**5.3.3 Rules for Verbs**

elif part\_of\_speech == 'v':

verb=word

verb\_pos=original.index(verb)

pos\_tagged=nltk.pos\_tag(original)

verb\_nltk\_pos = pos\_tagged[verb\_pos][1]

# no suffix for base form(simple present)

if verb\_nltk\_pos=='VB':

verb\_suffix=""

else:

verb\_suffix=get\_suffix(verb)

#tense of verb

verb\_tense=get\_tense(original,verb,verb\_suffix)

#single\_word meaning

if meaning.find(' ')==-1:

pos=is\_irregular(meaning)

if pos!=-1:

if verb\_tense=='present\_continous' or verb\_tense=='past\_continous' or verb\_tense=='future\_continous' or verb\_tense=='present\_perfect\_continous' or verb\_tense=='past\_perfect\_continous' or verb\_tense=='future\_perfect\_continous':

meaning=irregular\_verbs.iloc[pos,3]

elif verb\_tense=='present\_perfect' or verb\_tense=='past\_perfect' or verb\_tense=='future\_perfect':

meaning=irregular\_verbs.iloc[pos,2]

elif verb\_tense=='simple\_present' or verb\_tense=='simple\_future':

meaning=irregular\_verbs.iloc[pos,0]

elif verb\_tense=='simple\_past':

meaning=irregular\_verbs.iloc[pos,1]

else:

root=stemmer.stem(meaning)

verb=root+verb\_suffix

meaning=spell\_check(verb)

new=replace(original,word,meaning)

**5.3.4 Rules for Adverbs**

elif part\_of\_speech == 'r':

adverb=word

adv\_pos=original.index(adverb)

full=''

#single\_phase meaning

if meaning.find(' ')==-1:

new=replace(original,adverb,meaning)

#phrase\_meaning

else:

meaning\_list=meaning.split(' ')

answer=lesk(lesk\_raw,adverb)

ans=answer.name()

syn=ans.split('.')

# Get the simplified meaning

if syn[0] != adverb.lower():

if meaning.find("manner")==-1:

meaning="in a manner which is "+meaning

elif syn[0] == adverb.lower():

answer=answer.lemmas()[0].pertainyms()

if len(answer)>0:

answer=answer[0].name()

correct\_answer=lesk(lesk\_raw,answer)

correct\_meaning=correct\_answer.lemmas()[0].name()

found3=search(correct\_meaning)

if found3:

correct\_meaning=correct\_answer.definition()

if correct\_meaning.find("manner")==-1:

meaning="in a manner which is "+correct\_meaning

else:

meaning="in a "+correct\_meaning+" manner"

elif meaning.find("manner")==-1:

meaning="in a manner which is "+meaning

meaning\_list=meaning.split(' ')

last=0

count=0

for i in range(adv\_pos+1,len(original)):

if original[i]=='.' or original[i]=='"' or original[i]=='”':

count+=1

else:

break

if count==len(original)-1:

last=1

**5.3.5 Code for finding Subject, Object and Predicate**

# Find the Subject, Object and Predicate

# left search

sub\_found=0

for i in range(adv\_pos):

x=adv\_pos-i-1

if pos\_tagged[x][0]==',' or sub\_found==1:

break

elif pos\_tagged[x][1]=='NN' or pos\_tagged[x][1]=='NNS' or pos\_tagged[x][1]=='NNP' or pos\_tagged[x][1]=='NNPS' or pos\_tagged[x][1]=='PRP' or pos\_tagged[x][1]=='PRP$':

sub=[pos\_tagged[x][0]]+sub

sub\_found=1

if sub\_found==0:

sub=[pos\_tagged[x][0]]+sub

# right search

obj\_found=0

for x in range(adv\_pos+1,len(original)):

if pos\_tagged[x][0]==',' or pos\_tagged[x][0]=='.' or obj\_found==1:

break

elif pos\_tagged[x][1]=='JJ' or pos\_tagged[x][1]=='JJR' or pos\_tagged[x][1]=='JJS':

obj.append(pos\_tagged[x][0])

elif pos\_tagged[x][1]=='NN' or pos\_tagged[x][1]=='NNS' or pos\_tagged[x][1]=='NNP' or pos\_tagged[x][1]=='NNPS' or pos\_tagged[x][1]=='PRP' or pos\_tagged[x][1]=='PRP$':

obj.append(pos\_tagged[x][0])

obj\_found=1

elif obj\_found==0 and pos\_tagged[x][1].find('V')==-1:

obj.append(pos\_tagged[x][0])

print("Sub= "+' '.join(sub))

print("Preicate= "+adverb+" "+verb)

print("Object= "+' '.join(obj))

temp=[]

CHAPTER 6

RESULTS

CHAPTER 6

APPLICATIONS

CHAPTER 7

TESTING

Testing is the process of finding bugs and defects for analyzing the developed software. Software testing is a verification method that finds errors by applying a controlled set of conditions. It is the most suitable method of validating and verifying the functional and performance requirements. Test results are documented proof that requirements were met and can be repeated. The resulting data can be reviewed for confirmation of capabilities by all concerned.

**Why Testing is Necessary?**

All the products of software development (specifications, source code and test documents are written by human beings. All human beings tend to make errors regardless of how experienced or skilled they are. So a number of faults in system are expected and inevitable. Furthermore, it is not possible for any particular person to understand every aspect of system because of the complexity to develop many of the computer systems. The developed system can be validated and verified to check whether it is working according to specified requirements or not using testing and test cases. If it does not meet the requirements then the defects are further recorded and analyzed so that necessary changes or actions can be taken. One can use recorded defects for analyzing the limitations of the system where one can use it for the purpose of recovery from these defects. As the number of defects increases; the quality of developed software decreases. While developing the software system it is important to minimize the number of defects.

**Software Quality**

Definitions for quality in terms of Software Engineering Terminology found in the IEEE Standard Glossary are:

1. Quality relates to the degree to which a process, system or system component meets specified requirements.
2. Quality relates to the degree to which a process, system or system component meets customer or user needs, or expectations.

Few examples of quality attributes are:

1. Correctness: The degree to which the system performs its intended function is referred to as correctness.
2. Reliability: It is the degree to which the system is expected to perform its required functions.
3. Usability: It relates to the degree of effort needed to learn, operate, prepare input, and interpret output of the system.
4. Portability: It refers to the ability of system to be transferred from one environment to another.
5. Maintainability: The effort required to make changes in the system or software.
   1. **Validation of System**\

The system should accept only the English text as input. The system is tested for following parameters.

1. Even if the system accepts special characters and non-English text, it doesn’t affect the output because the data to be processed is matched with a dictionary, which contains only English text.
2. Tokenization is the first step in pre-processing. Tokenization is very important to perform Part-of-speech tagging of each words so that the required words can be categorized differently.
3. Part-of-speech tagging is an important step in pre-processing. It tags each token with a corresponding tag. Here we have selected the categories of nouns, adverbs, adjectives and verbs. The words which are tagged with part-of-speeches related to these categories will only be scanned for a match in the dictionary and the simplification process would be carried out for those words only.
4. Stemming is the most important step as this process will give the base words for the filtered words. These stems will be matched with the stems of the dictionary words - the pairs of that word and the surrounding words in the sentences will be scanned in WordNet to calculate the number of times they occur together. This helps in solving the problem of Word Sense Disambiguation.

CHAPTER 8

CONCLUSION

Computers will be able to learn from the information available online and apply it to real world problems with the help of better understanding of natural language processing. Computers will become more and more capable of receiving and giving instructions. To smoothen this process, Information simplification is required.

As explained through the various sections of this proposal, we need a full-proof universal Text Simplification system to overcome our linguistic barriers. Our system proves to have achieved not completely but a major part of Text simplification. Our system fulfills three very important objectives-

* The simplified output produced by the system has a simplified structure, simpler vocabulary and is grammatically correct – not accurately but does not affect the reading and understanding process.
* As seen from the results mentioned in the earlier section it is visible that the system works quite efficiently for the parts of speeches that we have selected i.e. Nouns, Verbs, Adjectives and Adverbs.
* The system also solves the tense identification and transformation problem.

We performed a deep study of the work done on Text Simplification and the existing systems about how they work, their limitations and flaws, the scope of improvement upon which we could work. The available Text simplification systems concentrate only on the lexical simplification which is useful only if one wants a substitution for a particular word. However, our system also incorporates the generation of grammatically and semantically correct sentence after the lexical simplification.

The future scope as we have mentioned in the Applications section, which includes making the system suitable for second language learners. The system would be very useful for those people who don’t have English as their first language and using our system would help them learn English more comfortably.

The system proposed by us, called ‘Simply5’ stands true on the above objectives, and would definitely benefit a large section of the society helping them learning the English language effectively and at a faster rate.

REFERENCES

[1] R.Chandrasekar, Christine Doran, B.Srinivas “Motivations and Methods for Text Simplification”. Article published in a journal in Research Gate,1998. Copenhagen, Denmark — August 05 - 09, 1996

[2] Carroll, John, et al. "Practical simplification of English newspaper text to assist aphasic readers." Proceedings of the AAAI-98 Workshop on Integrating Artificial Intelligence and Assistive Technology. 1998.

[3] Siddharthan, Advaith. "A survey of research on text simplification." ITL-International Journal of Applied Linguistics 165.2 (2014): 259-298.

[4] Feng, Lijun. "Text simplification: A survey." The City University of New York, Tech. Rep (2008).

[5] Shardlow, Matthew. "A survey of automated text simplification." International Journal of Advanced Computer Science and Applications 4.1 (2014): 58-70.

[6] Chandrasekar, Raman, and Bangalore Srinivas. "Automatic induction of rules for text simplification." Knowledge-Based Systems 10.3 (1997): 183-190.

[7] Siddharthan, Advaith. "Syntactic simplification and text cohesion." Research on Language & Computation 4.1 (2006): 77-109.

[8] Kauchak, David, et al. "Text simplification tools: using machine learning to discover features that identify difficult text." System Sciences (HICSS), 2014 47th Hawaii International Conference on. IEEE, 2014.

[9] Phil Katz,“Supervised Word Sense Disambiguation using Python” Research Gate, 2005.

[10] Jonas EKEDAHL and Koraljka GOLUB “Word sense Disambiguation using WordNet and the Lesk algorithm”. 2004

[11] Robin Keskisärkkä “Automatic Text Simplification via Synonym Replacement” Robin Keskisärkkä, 2012

[12] Biran, Or, Samuel Brody, and Noémie Elhadad. "Putting it simply: a context-aware approach to lexical simplification." Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2. Association for Computational Linguistics, 2011.

[13] Gondy Leroy “User Evaluation of the Effects of a Text Simplification Algorithm Using Term Familiarity on Perception, Understanding, Learning and Information Retention”. Journal of Medical Internet Research, 2013.

[14] Bateman, John A. "Sentence generation and systemic grammar: an introduction." Iwanami Lecture Series: Language Sciences 8 (1997): 1- 45.

[15] Arendse Bernth “EasyEnglish: A Tool for Improving Document Quality”. Proceeding ANLC '97 Proceedings of the fifth conference on Applied natural language processing Pages 159-165

[16] Levison, Michael, and Gregory Lessard. "A system for natural language sentence generation." Computers and the Humanities 26.1 (1992): 43-58.

[17] Hammervold, Kathrine. "Sentence generation and neural networks." Proceedings of the first international conference on Natural language generation-Volume 14. Association for Computational Linguistics, 2000.

[18] S. Rebecca Thomas and Sven Anderson “WordNet-based Lexical Simplification of a Document”. Proceedings of KONVENS, 2012.

[19] Yngve, V. H. A. (1962), Random generation of English sentences, in ‘The 1961 Conference on Machine Translation of Languages and Applied Language Analysis’, Her Majesty’s Stationery Office,London.

[20] Friedman, J. (1969), ‘Directed random generation of sentences’, Communications of the Association for Computing Machinery 12(6)

[21] Zhang, Yaoyuan, et al. "A Constrained Sequence-to-Sequence Neural Model for Sentence Simplification." arXiv preprint arXiv:1704.02312 (2017).

[22] Alu´ısio, S. M., L. Specia, T. A. Pardo, E. G. Maziero, & R. P. Fortes. 2008. Towards brazilian portuguese automatic text simplification systems.

[23] Niklaus, Christina, et al. "A Sentence Simplification System for Improving Relation Extraction." arXiv preprint arXiv:1703.09013 (2017).

[24] [http://www.morganclaypool.com/doi/abs/10.2200/S00700ED1V01Y201 602HLT032](http://www.morganclaypool.com/doi/abs/10.2200/S00700ED1V01Y201%20602HLT032)

[25] https://www.simplish.org

[26] https://www.rewordify.com

PAPERS PRESENTED AND PUBLISHED

* Akshay Dangare, Meghna Pai, Rohit Kothavade, Mohini Kamat. “Simply5 : A Proposal for a Text Simplication System for Effective Learning”. Presented at : 12th INDIACom; 2018 5th International Conference on Computing for Sustainable Global Development.