



PROJECT REPORT ON LEXICAL SIMPLIFICATION

A Project Work Submitted in Partial Fulfillment of the requirements for

The Course

ALGORITHMS FOR INTELLIGENCE WEB AND INFORMATION RETRIEVAL

By

SHREYA PRABHU – PES1201800128

RAGHAV ROY – PES1201800342

RAKSHA RAMESH – PES1201800345

Under the supervision of

Prof. Nagegowda K S

Associate Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

PES UNIVERSITY

RR CAMPUS

TABLE OF CONTENTS:

1. INTRODUCTION.....	3
2. OVERVIEW OF THE PROJECT.....	5
3. ALGORITHM PSEUDO CODE.....	9
4. RESULTS AND DISCUSSION.....	13
5. CONCLUSION AND FUTURE SCOPE.....	15
6. REFERENCE & SOURCES.....	16

1. INTRODUCTION

In our daily life, we come across a lot of sentences that are so convoluted that it becomes difficult to comprehend. This is where text simplification comes into the picture.

Text simplification is an operation used in natural language processing to modify, enhance, classify or otherwise process an existing corpus of human-readable text in such a way that the grammar and structure of the prose is greatly simplified, while the underlying meaning and information remains the same. Text simplification is an important area of research, because natural human languages ordinarily contain large vocabularies and complex compound constructions that are not easily processed through automation. In terms of reducing language diversity, semantic compression can be employed to limit and simplify a set of words used in given texts.

Text simplification is within the field of NLP, and 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 text simplification come from these other fields. Another very similar process is text summarization. Text simplification is different from 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 while simplification focuses on making the content easier to comprehend.

As the amount of unstructured text data has exploded in recent decades, due to the advent of the Internet, so has the need, and interest, in text simplification research. TS is a diverse field with a number of target audiences, each with a specific focus. One of the most prominent target audiences for text simplification are foreign language learners, for whom various approaches to simplifying text have been pursued, often focusing on

lexical but also sentence-level simplification. Text simplification is also of interest to people suffering from dyslexia, and the aphasic, for whom particularly long words and sentences, but also certain surface forms such as specific character combinations, may pose difficulties. Application of text simplification for those suffering from autism focuses on reducing the amount of figurative expressions in a text or reducing syntactic complexity. Reading beginners (both children and adults) are another group with very particular needs, and text simplification research has tried to provide this group with methods to reduce the amount of high-register language and non-frequent words.

There are primarily two approaches to implement text simplification, the extractive method and abstractive method.

The extractive approach involves creating a frequency distribution of all the words, after preprocessing, and calculates sentence weights for each sentence. We can summarize the original text by either selecting N number of sentences with the highest sentence weights or by selecting all sentences above a certain threshold sentence weight.

Abstractive Approach Abstractive text simplification involves generation of new and novel text, which is lexically and/or syntactically simpler than the original. Abstractive approaches have mostly focused on lexical or phrasal substitutions for sentence-level simplification. This process focuses solely on simplifying the vocabulary of a text, instead of additional simplification tasks of grammatical or syntactic simplification.

Lexical substitution is the task of identifying a substitute for a word in the context of a clause. For instance, given the following text: "After the *match*, replace any remaining fluid deficit to prevent chronic dehydration throughout the tournament", a substitute of *game* might be given.

The goal of Lexical Simplification is to replace complex words (typically words that are used less often in language and are therefore less familiar to readers) with their simpler synonyms, without infringing the grammaticality and changing the meaning of the text.

Another approach to text simplification apart from lexical simplification is syntactic simplification. Syntactic text simplification approaches modify a sentence's structure in order to make it easier to comprehend, whereas lexical text simplification approaches mainly apply localised modifications to words based on their local lexical context (often a sentence).

The aim of our project is to use Lexical substitution to simplify text. **Lexical Simplification** uses contextual word substitution for simplifying text and therefore making it more comprehensible and easier to understand.

2. OVERVIEW OF THE PROJECT

The project was implemented in Python using the following libraries: pandas, ujson, nltk, gensim. The most important of which was nltk.

The Natural Language Toolkit, or more commonly called **nltk**, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

From this library, the following modules were used for the implementation of our project:

The Brown University Standard Corpus of Present-Day American English (or just Brown Corpus) is an electronic collection of text samples of American English, the first major structured corpus of varied genres. This corpus first set the bar for the scientific study of the frequency and distribution of word categories in everyday language use.

WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. **Synsets** are interlinked by means of conceptual-semantic and lexical relations.

The implementation comprises of the following steps:

1. **Input corpus:**

In this step, the input text is accepted from the user for simplification.

2. Create the Frequency Corpus:

- a. A frequency dictionary generated from NLTK's brown corpus that contains every word and the number of occurrences.
- b. A bigram frequency dictionary that contains all bigram combinations from the brown corpus and their frequency of occurrence.

3. Perform text preprocessing:

After accepting input, the pre-processing of the input text is being performed. In this phase, the main purpose is to perform tokenization of the sentence. In tokenization sentence is parsed into separate words

4. Perform POS Tagging:

In this step, the parts-of-speech of the tokenized words is identified. Each word is assigned with a tag containing abbreviations of parts-of-speech. Depending on parts-of-speech the list is separated into a list of Adverbs, Adjectives, Nouns.

5. Complex words identification:

In this step, the lists created in the previous step are merged. The words in the merged list are compared in sentences to find the position of each word. A complete set of complex words and their position in the sentence is created.

6. Suitable Synonym identification:

This step focuses on finding the suitable simple synonym for complex words. Here, synset is used for synonym identification. Alternatives are then stored and kept ready to perform replacement according to their position. The alternative candidates are also converted to the correct tense form according to the POS tag.

7. Perform synonym replacement:

After finding synonyms of complex words, the replacement of words in sentences takes place depending upon the relative position of the complex word.

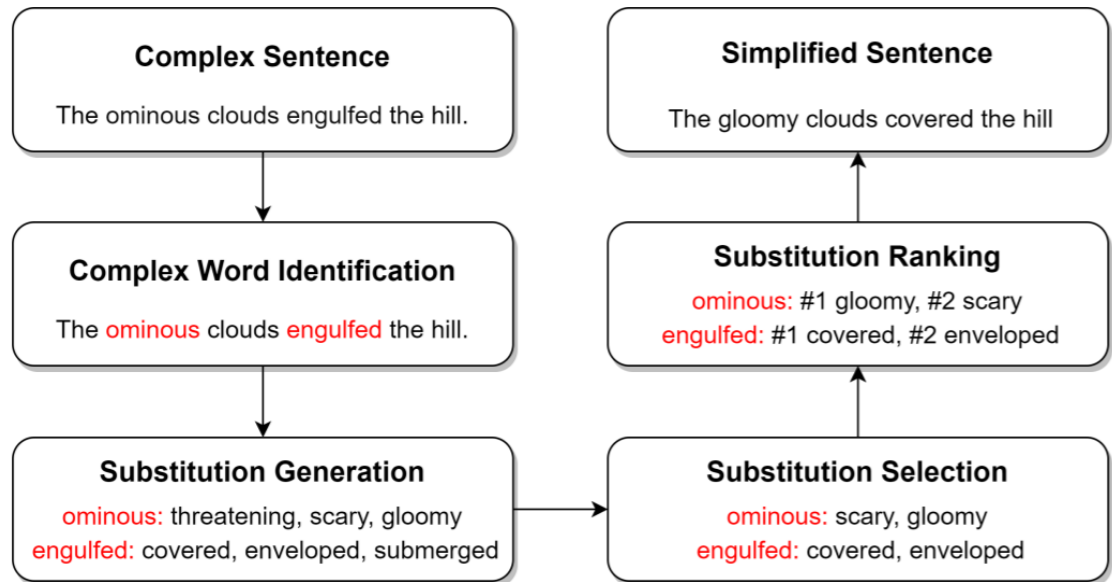
The actual execution of the project comprised of the following steps:

1. The input sentences are tokenized.
2. The frequency of the 40 percentile of the most frequent words in the input is set as the threshold.
3. Any word whose frequency is lesser than the threshold is considered 'difficult' and kept in the list 'difficultWords'.
4. For each word in difficultWords(after checking whether the word can be replaced (based on the part of speech embodies), a set of replacement candidates is chosen from Wordnet along with their frequency in the Brown Corpus.
5. For each word in difficultWords, from the set of candidates, a subset best candidates is made with only the candidates who fit in the same context of the difficultWord:
 - If the original text is X-difficultWord-Y, bigrams X-candidate and candidate-Y should exist.
 - If the bigrams exist, that candidate is put inside best_candidates for the difficultWord along with its corresponding bigram score(average of the 2 sub-bigram frequencies in the Brown corpus).

Replacement Strategies (models):

1. For each word in difficultWords, out of the best_candidates list, the candidate with the highest bigram score is chosen as the replacement .
2. For each word in difficultWords, out of the best_candidates list, the candidate with the highest frequency in the Brown Corpus after checking whether the word fits the context (bigrams X-candidate and candidate-Y must exist).
3. For each word in difficultWords, out of the best_candidates list, the candidate with the highest frequency in the Brown Corpus is chosen as replacement.

A diagrammatic view of the process:



3. ALGORITHM PSEUDO CODE

Algorithm 1: Generate Frequency Dictionary

Result: Frequency Dictionary

initialize freqDict;

for *sentence in Brown corpus sentences* **do**

for *word in sentence* **do**

 freqDict[word] +=1;

end

end

return freqDict;

Algorithm 2: Find Bigram Score

Result: Bigram Score

initialize leftWord, rightWord, freqDict and replacement;

score \leftarrow 0 **if** '*leftWord + replacement*' **in** *freqDict* **then**

 score += freqDict['*leftWord + replacement*']

end

if '*rightWord + replacement*' **in** *freqDict* **then**

 score += freqDict['*rightWord + replacement*']

end

return score;

Algorithm 3: Lexical Simplification

Result: Lexically simplified sentence

tokenize sentences;

find difficultWords;

for *word in difficultWords* **do**

 replacements \leftarrow *findReplacements()* **for** *candidate in replacements* **do**

 bestCandidates \leftarrow *findBestCandidates()* **end**

end

return replacedSentences;

CODE SNIPPETS:

Function for Generating frequency dictionary:

```
text_simplification.py M X corpus_frequency.csv steps.txt M input1.txt M input2.txt M output10.txt M out
LexicalTextSimplification-test > text_simplification.py > Simplifier > _init_
14 def generate_brown_frequency_dictionary():
15     """ Create frequency distribution of BROWN corpora. """
16     brown_frequency_dictionary = FreqDist()
17     for sentence in brown.sents():
18         for word in sentence:
19             brown_frequency_dictionary[word] += 1
20
21     corpus_frequency_distribution = pd.DataFrame(list(brown_frequency_dictionary.items()), columns = ["Word", "Frequency"])
22     corpus_frequency_distribution.sort_values("Frequency")
23     corpus_frequency_distribution.to_csv('corpus_frequency.csv')
24     return brown_frequency_dictionary
25
```

Function for checking if the word fits context a function to return bigram score:

```
38
39 def check_if_word_fits_the_context(self, context, token, replacement):
40     """ Check if bigram with the replacement exists.
41     Check for word preceding and succeeding the replacement in the bigram dictionary. """
42
43     if len(context) == 3:
44         if (context[0] + ' ' + replacement).lower() in self.bigrams_brown_frequency_dictionary.keys():
45             return True
46         else:
47             return False
48     else:
49         return False
50
51 def return_bigram_score(self, context, token, replacement):
52     """ Return the averaged frequency of left- and right-context bigram. """
53     score = 0
54     if (context[0] + ' ' + replacement).lower() in self.bigrams_brown_frequency_dictionary.keys():
55         score += self.bigrams_brown_frequency_dictionary[(context[0] + ' ' + replacement).lower()]
56     if (replacement + ' ' + context[2]).lower() in self.bigrams_brown_frequency_dictionary.keys():
57         score += self.bigrams_brown_frequency_dictionary[(replacement + ' ' + context[2]).lower()]
58     return score / 2
```

Function to check whether a word is replaceable:

```
60 def check_if_replacable(self, word):
61     """ Check POS, we only want to replace nouns, adjectives and verbs. """
62     word_tag = pos_tag([word])
63     if 'NN' in word_tag[0][1] or 'JJ' in word_tag[0][1] or 'VB' in word_tag[0][1]:
64         return True
65     else:
66         return False
67
```

Function to convert tense of the replaced word:

```

9 def convert(word_from, word_to):
10     """ Analyses POS tags and converts words to a desired form. """
11     if tag(word_to)[0][1] == 'VBD':
12         converted = conjugate(word_from, 'past')
13     elif tag(word_to)[0][1] == 'VBN':
14         converted = conjugate(word_from, 'past')
15     elif tag(word_to)[0][1] == 'VBZ':
16         converted = conjugate(word_from, 'present', '3sg')
17     elif tag(word_to)[0][1] == 'VBP':
18         converted = conjugate(word_from, 'present', '1sg')
19     elif tag(word_to)[0][1] == 'VB':
20         converted = conjugate(word_from, 'infinitive')
21     elif tag(word_to)[0][1] == 'NN' and tag(word_from)[0][1] == 'NNS':
22         converted = singularize(word_from)
23     elif tag(word_to)[0][1] == 'NNS' and tag(word_from)[0][1] == 'NN':
24         converted = pluralize(word_from)
25     else:
26         converted = word_from
27     return converted
28

```

The final simplification function:

```

94
95 def simplify(self, input):
96     simplified0 = ''
97     simplified1 = ''
98     simplified2 = ''
99
100     sents = sent_tokenize(input) # Split by sentences
101     '''Top 40 % least frequency score (rarer) words of the input corpus are taken as difficult words'''
102     top_n = int(40/100*(len(input)))
103     freq_top_n = sorted(self.brown_frequency_dictionary.values(), reverse=True)[top_n - 1]
104     for sent in sents:
105         self.steps.write(sent + '\n')
106         tokens = word_tokenize(sent) # Split a sentence by words
107         #Store all difficult words
108         difficultWords = [t for t in tokens if self.brown_frequency_dictionary[t] < freq_top_n]

```

```

111     all_options = {}
112     for difficultWord in difficultWords:
113         replacement_candidate = {}
114
115         '''Collect WordNet synonyms for each difficult word,
116         | along with their brown corpus frequency.'''
117
118         for option in self.generate_wordnet_candidates(difficultWord):
119             replacement_candidate[option] = self.brown_frequency_dictionary.freq(option)
120
121         '''store all these candidates in all_options '''
122
123         all_options[difficultWord] = replacement_candidate
124     all_options_list = [(k, v) for k, v in all_options.items()]
125     self.steps.write('all_options:')
126     self.steps.write(str(all_options_list) + '\n')
127
128     ''' Populate best candidates dictionary if it is a bigram, and add bigram score '''
129     best_candidates = {}
130     for token_id in range(len(tokens)):
131         token = tokens[token_id]
132
133         best_candidates[token] = {}

```



```

128     ''' Populate best candidates dictionary if it is a bigram, and add bigram score '''
129     best_candidates = {}
130     for token_id in range(len(tokens)):
131         token = tokens[token_id]
132
133         best_candidates[token] = {}
134         if token in all_options:
135             for opt in all_options[token]:
136                 if token_id != 0 and token_id != len(tokens): # if not the first or the last word in
137                     if self.check_if_word_fits_the_context(tokens[token_id - 1:token_id + 2], token,
138
139                         best_candidates[token][opt] = self.return_bigram_score(tokens[token_id - 1:token_id + 2], token)
140     # self.steps.write('best_candidates:' + str(best_candidates) + '\n')
141     best_candidates_list = [(k, v) for k, v in best_candidates.items()]
142     self.steps.write('best_candidates:')
143     self.steps.write(str(best_candidates_list) + '\n')
144
145     '''Generate steps0 - take the word with the highest bigram score'''
146     output = []
147     for token in tokens:
148         if token in best_candidates:
149             if token.istitle() is False and best_candidates[token] != {}:
150                 # Choose the one with the highest bigram score

```

```

145     '''Generate steps0 - take the word with the highest bigram score'''
146     output = []
147     for token in tokens:
148         if token in best_candidates:
149             if token.istitle() is False and best_candidates[token] != {}:
150                 # Choose the one with the highest bigram score
151                 best = max(best_candidates[token], key=lambda i: best_candidates[token][i])
152                 self.steps.write('best v1:' + str(token) + ' -> ' + str(best) + '\n')
153                 output.append(best)
154             else:
155                 output.append(token)
156         else:
157             output.append(token)
158     simplified0 += ' '.join(output)
159
160     '''Generate steps1 - take the word with the highest frequency + check the context'''
161     output = []
162     for token_id in range(len(tokens)):
163         token = tokens[token_id]
164         if token in all_options and len(all_options[token]) > 0 and token in difficultWords and token.
165             if token_id != 0 and token_id != len(tokens):
166                 # Choose most frequent and check if fits the context
167                 best_filtered = {word: all_options[token][word] for word in all_options[token] if

```

```

160     '''Generate steps1 - take the word with the highest frequency + check the context'''
161     output = []
162     for token_id in range(len(tokens)):
163         token = tokens[token_id]
164         if token in all_options and len(all_options[token]) > 0 and token in difficultWords and token.
165             if token_id != 0 and token_id != len(tokens):
166                 # Choose most frequent and check if fits the context
167                 best_filtered = {word: all_options[token][word] for word in all_options[token] if
168                     self.check_if_word_fits_the_context(tokens[token_id - 1:token_id + 2]
169                     and self.check_pos_tags(tokens, token_id, word)}
170             if best_filtered != {}: # if not empty
171                 best = max(best_filtered, key=lambda i: best_filtered[i])
172                 self.steps.write('best v2:' + str(token) + ' -> ' + str(best) + '\n')
173                 output.append(best)
174             else:
175                 output.append(token)
176         else:
177             output.append(token)
178         else:
179             output.append(token)
180     simplified1 += ' '.join(output)
181

```



```

181
182     '''Generate steps2 - take the synonym with the highest frequency'''
183     output = []
184     for token in tokens:
185         # Replace word if in is difficult and a candidate was found
186         if token in all_options and len(all_options[token]) > 0 and token in difficultWords and token:
187             best = max(all_options[token], key=lambda i: all_options[token][i])
188             self.steps.write('best v3:' + str(token) + ' -> ' + str(best) + '\n')
189             output.append(best)
190         else:
191             output.append(token)
192     simplified2 += ' '.join(output)
193
194     return simplified0, simplified1, simplified2

```

4. RESULTS AND DISCUSSION

For our test cases, we used three large input files that were passed to the models. The models processed the input files and each model simplified the input text differently.

The file steps.txt indicates all generated replacement candidates and the ones used specifically by each of the different models.

The following is a snapshot of what the bigram frequencies look like:

```

LexicalTextSimplification-test > bigrams_frequency.csv
You, 2 weeks ago | 1 author (You)
1  ,Bigram,Frequency
2  0,of the,74820247
3  1,in the,53383115
4  2,to the,30419908
5  3,on the,25946885
6  4,for the,20270556
7  5,and the,19710265
8  6,to be,19508465
9  7,is a,17563849
10 8,with the,16078298
11 9,at the,15423651
12 10,from the,13276473
13 11,with a,12180629
14 12,will be,12025933
15 13,is the,11355480
16 14,of a,11285608
17 15,in a,10714844
18 16,for a,10638755
19 17,it is,10181558
20 18,you can,9872981
21 19,by the,9619211
22 20,that the,9477816
23 21.can be,9224680

```

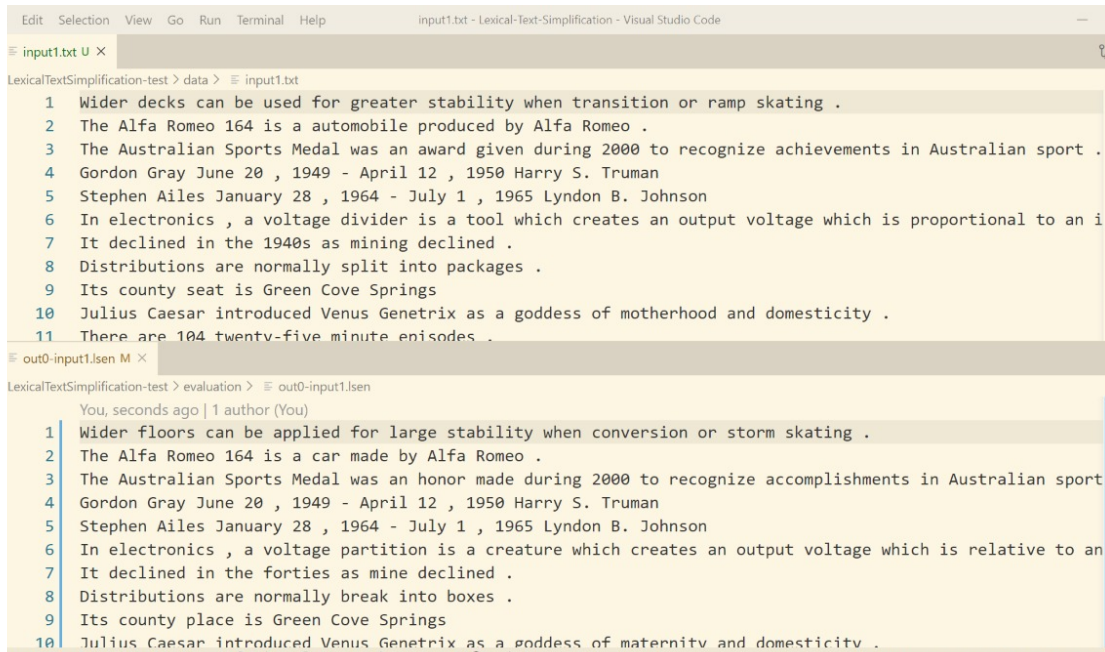
The following is an example of the steps.txt:

```

617 | Madison sought to accomplish this by demonstrating conclusively that the Acts violated the constitution .
618 | difficultWords:['Madison', 'sought', 'accomplish', 'this', 'demonstrating', 'conclusively', 'Acts', 'viol
619 | all_options:[('Madison', {'madison': 0.0, 'james_madison': 0.0, 'capital_of_wisconsin': 0.0, 'president_m
620 | best_candidates:[('Madison', {}), ('sought', {'tried': 263772.5, 'attempted': 24020.0}), ('to', {}), ('ac
621 | best v1:accomplish -> achieve
622 | best v1:demonstrating -> evidence
623 | best v1:violated -> broke
624 | best v1:constitution -> organization
625 | Its current messenger client is Windows Live Messenger .
626 | difficultWords:['Its', 'current', 'messenger', 'client', 'Windows', 'Live', 'Messenger']
627 | all_options:[('Its', {}), ('current', {'stream': 4.392038525928529e-05, 'flow': 5.856051367904705e-05, 'e
628 | best_candidates:[('Its', {}), ('current', {'flow': 779.0}), ('messenger', {}), ('client', {'node': 5066.5
629 | best v1:client -> customer
630 | best v3:messenger -> courier
631 | It comes from Stewart Grand Prix .|
632 | difficultWords:['It', 'comes', 'from', 'Stewart', 'Grand', 'Prix']
633 | all_options:[('It', {}), ('comes', {}), ('from', {}), ('Stewart', {'dugald_stewart': 0.0, 'jimmy_stewart'
634 | best_candidates:[('It', {}), ('comes', {}), ('from', {}), ('Stewart', {}), ('Grand', {'grand': 1757.0}),
635 | difficultWords:['Cities', 'in', 'Brazil']
636 | all_options:[('Cities', {'metropolis': 6.889472197534947e-06, 'cities': 9.128550661733805e-05, 'urban_cen
637 | best_candidates:[('Cities', {}), ('in', {}), ('Brazil', {}), ('.', {})]The southern and western boundarie
638 | difficultWords:['southern', 'western', 'boundaries', 'are', 'based', 'continental', 'shelf']
639 | all_options:[('southern', {'southerly': 0.0}), ('western', {'western_sandwich': 0.0, 'horse_operas': 0.0,

```

The following is how the lexically simplified sentences look:



```
LexicalTextSimplification-test > data > input1.txt
1 Wider decks can be used for greater stability when transition or ramp skating .
2 The Alfa Romeo 164 is a automobile produced by Alfa Romeo .
3 The Australian Sports Medal was an award given during 2000 to recognize achievements in Australian sport .
4 Gordon Gray June 20 , 1949 - April 12 , 1950 Harry S. Truman
5 Stephen Ailes January 28 , 1964 - July 1 , 1965 Lyndon B. Johnson
6 In electronics , a voltage divider is a tool which creates an output voltage which is proportional to an i
7 It declined in the 1940s as mining declined .
8 Distributions are normally split into packages .
9 Its county seat is Green Cove Springs
10 Julius Caesar introduced Venus Genetrix as a goddess of motherhood and domesticity .
11 There are 104 twenty-five minute episodes .

LexicalTextSimplification-test > evaluation > out0-input1.isen
You, seconds ago | 1 author (You)
1 Wider floors can be applied for large stability when conversion or storm skating .
2 The Alfa Romeo 164 is a car made by Alfa Romeo .
3 The Australian Sports Medal was an honor made during 2000 to recognize accomplishments in Australian sport
4 Gordon Gray June 20 , 1949 - April 12 , 1950 Harry S. Truman
5 Stephen Ailes January 28 , 1964 - July 1 , 1965 Lyndon B. Johnson
6 In electronics , a voltage partition is a creature which creates an output voltage which is relative to an
7 It declined in the forties as mine declined .
8 Distributions are normally break into boxes .
9 Its county place is Green Cove Springs
10 Julius Caesar introduced Venus Genetrix as a goddess of maternitv and domesticity .
```

In the above picture, the changes made in each sentence can be clearly seen.

5. CONCLUSION AND FUTURE SCOPE

In this project we implemented lexical text simplification that simplifies complex words in a sentence based on the context it appears in. We first chose a metric for the selection of the words that would be categorised as “difficult”. These difficult words would then be required to be replaced. The complex words were selected as the words whose frequency was lesser than the 40th percentile of the entire text. For each of the difficult words, a set or replacement candidates are chosen out of which the best candidate is chosen based on the model employed.

The future scope of this project is to implement the part - of - speech tagger ourselves so we have more control over the words considered as replaceable. The performance of the model can be improved by creating our own context reader and coming up with a separate algorithm to find replacement words rather than using the WordNet corpus.

6. REFERENCE & SOURCES

1. <https://www.aclweb.org/anthology/W03-1602.pdf>
2. <https://www.aclweb.org/anthology/W13-4813.pdf>
3. <https://github.com/cocoxu/simplification/tree/master/data/turkcorpus>
4. <https://pypi.org/project/readability/>
5. <https://github.com/feralvam/easse>
6. <http://nlpprogress.com/english/simplification.html>
7. Introduction to Information Retrieval - Christopher Manning, Prabhakar Raghavan,
8. Hinrich Schütze