https://doi.org/10.31436/iiumej.v21i2.1408

WORD SEGMENTATION OF OUTPUT RESPONSE FOR SIGN LANGUAGE DEVICES

NOR FARAHIDAH ZA'BAH*, AHMAD AMIERUL ASYRAF MUHAMMAD NAZMI, AMELIA WONG AZMAN

Department of Electrical and Computer Engineering, Kulliyyah of Engineering, International Islamic University Malaysia, Kuala Lumpur, Malaysia

**Corresponding author: adah510@iium.edu.my*

(Received: 22nd March 2020; Accepted: 11th June 2020; Published on-line: 4th July 2020)

ABSTRACT: Segmentation is an important aspect of translating finger spelling of sign language into Latin alphabets. Although the sign language devices that are currently available can translate the finger spelling into alphabets, there is a limitation where the output is stored in a long continuous string without spaces between words. The system proposed in this work is meant to be used together with a text-generating glove device. The system used text input string and the string is then fed into the system, one character at a time, and then it is segmented into words that is semantically correct. The proposed text segmentation method in this work is by using the dynamic programming and back-off algorithm, together with the probability score using word matching with an English language text corpus. Based on the results, the system is able to properly segment words with acceptable accuracy.

ABSTRAK: Segmentasi adalah aspek penting dalam menterjemahkan ejaan bahasa isyarat ke dalam huruf Latin. Walaupun terdapat peranti bahasa isyarat yang menterjemahkan ejaan jari menjadi huruf, namun begitu, huruf-huruf yang dihasilkan disimpan dalam rentetan berterusan yang panjang tanpa jarak antara setiap perkataan. Sistem yang dicadangkan di dalam jurnal ini akan diselaraskan bersama dengan sarung tangan bahasa isyarat yang boleh menghasilkan teks. Sistem ini akan mengambil rentetan input teks di mana huruf akan dimasukkan satu persatu dan huruf-huruf itu akan disegmentasikan menjadi perkataan yang betul secara semantik. Kaedah pembahagian yang dicadangkan ialah segmentasi yang menggunakan pengaturcaraan dinamik dan kaedah kebarangkalian untuk mengsegmentasikan huruf-huruf tersebut berdasarkan padanan perkataan dengan pengkalan data di dalam Bahasa Inggeris. Berdasarkan hasil yang telah diperolehi, sistem ini berjaya mengsegmentasikan huruf-huruf tersebut dengan berkesan dan tepat.

KEY WORDS: Segmentation, Sign Language, Algorithm

1. INTRODUCTION

Sign language is the main language used by deaf people around the world. However, there is always a gap between them and people without disability, since communication is a problem between them. Sign language is a necessity for people with hearing problems to communicate and interact with others. For non-disabled people, learning sign language is optional. Even if they learn sign language, it is difficult for them to be able to grasp it well as they do not use it frequently unless they are around the deaf community. This is the reason why a sign language device is needed. The device may help to improve communication and reduce the language barrier between them.

In order to solve this problem, numerous research and studies were performed to make a sign language device possible. Although there are many research papers on the topic, there are still many challenges and still there are no perfect solutions for translating sign language into other languages due to various constraints.

One of the biggest challenges in making a sign language device is the segmentation issue [1]. This issue is faced differently based on the types of input for the device. For example, camera-based sign language devices must be able to detect the hand sign made by the user correctly to ensure a proper translation [1]. Glove-type devices [2][3], on the other hand, are based on digital text output but face issues to segment the text and sentences properly. Therefore, it is difficult for these devices to determine what kind of output to produce and what the user intends to do, without a proper segmentation system.

Currently, a glove-type device that can produce character output based on hand signs is being researched. However, it is not working properly as it faces the text segmentation issue. All the alphabet character outputs from the device are stored in a long continuous string of text without spaces. For example, if the user intends to say "*word of the day*", the output that will be produced is '*wordoftheday*'. Therefore, in order to split this string, word segmentation needs to be performed to ensure that the output is correct and will be understood by humans [4]. There are a number of segmentation systems proposed using different approaches and algorithms to improve the accuracy of the output due to the challenges faced in many areas such as language translation and spelling correction. Table 1 shows a summary of some previous work on segmentation.

Table 1: Summary of some previous work on segmentation

Reference	Application	Input	Technique
G. A. Rao, P. Kishore [1]	Sign language translation	Video captured with camera	Used Gaussian filtering for pre-processing and segmentation. Then, hand and head contour are segmented with morphological subtraction.
Peter Norvig [4]	Word segmentation, spelling correction	Text input, URL	Proposed a general method of word segmentation based on Google n-gram dataset using python programming.
W. Jiang [5]	Word segmentation	Text input from user	Proposed a method of word segmentation using dynamic programming to improve performance. Divide and conquer algorithm is introduced so there will be no maximum length character limit for the input.
S. N. Nawaz [6]	Word segmentation	Urdu text	Proposed word segmentation model using Conditional Random Field (CRF) algorithm.
D. Tanaya and M. AdrianI [7]	Language transcription	Text in Javanese script and Latin character	Used a dictionary-based model with Maximal Matching Algorithm modified with additional steps. Performed character statistics on the dictionary to improve the output score.
S. Srinivasan, S. Bhattacharya [8]	Web-domain and hashtag segmentation	Capturing URL from website and twitter hashtag from twitter API	Used multiple corpora from different sources to improve the accuracy of segmentation and used dynamic programming to implement inference algorithm.
Y. Bassil, M. Alwani [9]	Spelling correction for Optical Character Recognition (OCR)	Optical Character Recognition (OCR) output text	Proposed an error correction algorithm based on Google spelling suggestion. The output from OCR will be fed to Google search engine and triggers "did you mean" feature if there is at least one error. Google search is capable of segmenting words properly in case the OCR output text has spacing error.
S. P. Panda and A. K. Nayak [10]	Speech recognition system	Voice recorded in *.wav* file	Proposed a model that performs segmentation at syllable boundaries using vowel offset point (VOP) identification and zero-crossing rate (ZCR) technique.

Reference	Application	Input	Technique
Y. Bassil, M. Alwani [11]	Error Correction for Automatic Speech Recognition (ASR)	ASR output text	An error correction algorithm based on Bing spelling suggestion. The output of ASR will be fed to Bing search engine as an input.
S. S. Chawathe [12]	Word segmentation	Sanskrit text	Proposed a dictionary-based model with dynamic programming to match the input with the dictionary word with the inclusion of Aho-Corasick algorithm

2. METHODOLOGY

2.1. Overview

For the proposed word segmentation system, three main components are required as shown in Fig. 1; the text input, the processing algorithm that contains the text dictionary, and the output. While the input and output are simple and self-explanatory, the processing algorithm involves some steps and procedures in order to process the data and produce the intended output. This will be discussed in the next section.

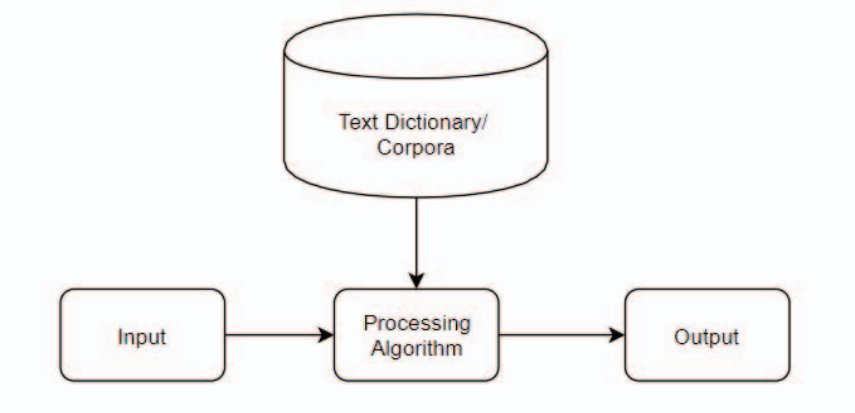


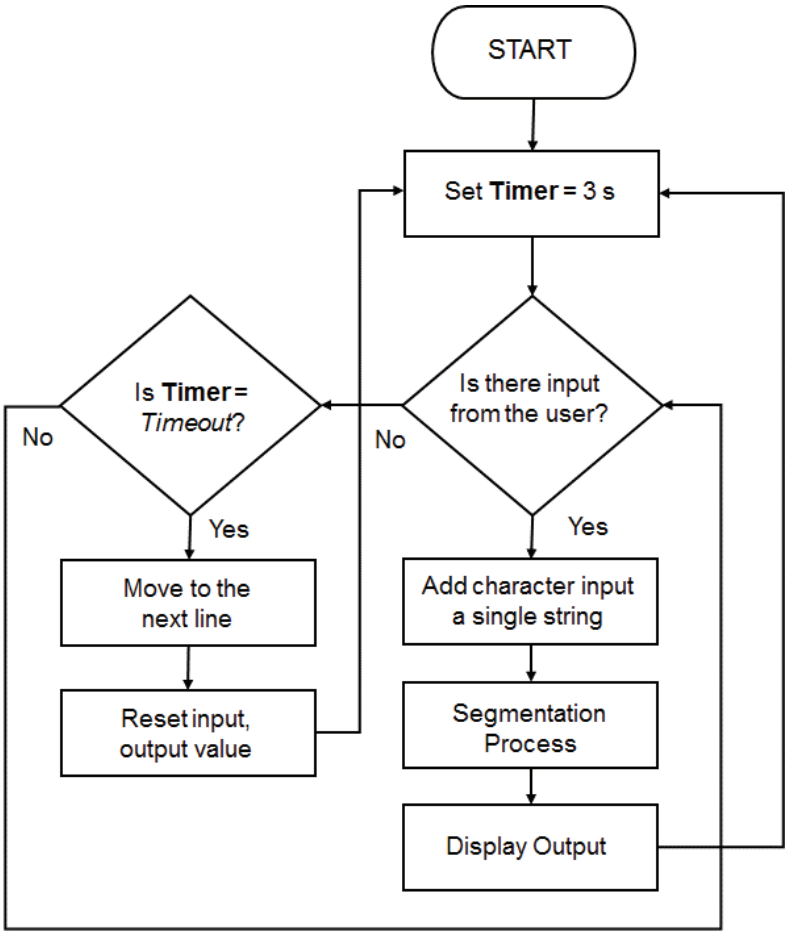
Fig. 1. Block diagram for the word segmentation model

For the input, a text-generating glove device is used to produce the alphabet characters to be fed into the system. However, in this work, a keyboard input will be used to emulate the glove device to produce the alphabet characters when the wearer makes a proper hand sign. Since the glove can only produce alphabet characters, all the non-alphabet keyboard keys such as "Spacebar", "Enter", "Backspace" and all non-alphabet keys will be ignored. The flowchart for the model is illustrated in Fig. 2. Since the "Enter" key is ignored, the string will be continuously fed into the system every time a character is added.

Secondly, for the processing of the input, Python programming language will be used to write and implement the segmentation algorithm. Also, for this work, a pre- installed dataset called text corpus will be used as the word library for the word matching mechanism. This text corpus is used to provide a vocabulary of words to the processor and "guide" the system with the correct words for segmentation. This is similar to the work done by Jiang [5] and Jenks [13] on the general word segmentation algorithm.

For the output, it will be directly shown on display as soon as a character is added into the string. When the user stops entering an input for more than 3 seconds, the system will reset the input in order to allow the user to enter the next line of strings.

https://doi.org/10.31436/iiumej.v21i2.1408



Set Timer to 3 seconds
User enters characters into the system using the keyboard
Add Character into a single input variable
Segmentation Process starts – where the input will be matched with the words stored in the text corpus.
Display the output on the screen, and the Timer is set to 3 seconds again
If user enters another character in less than 3 seconds, the previous output is reset and the character is added to the first string of characters and segmentation process starts again

Fig. 2. Flowchart of the segmentation system

2.2. Segmentation Algorithm

There are some steps involved in implementing the word segmentation system, which involves the division of characters into possible words and then matching the words with the words contained in the text corpus. This part takes place after the input string is being fed into the processor.

2.2.1. Dynamic Programming

The process of matching strings with words inside the text corpus is performed using dynamic programming. Dynamic programming is a well-known method used in programming to solve complex problems by breaking them into smaller and simpler sub-problems. In the case of word segmentation, a long continuous string entered by the user is split into two substrings by matching the string with the proper word inside the text corpus [5]. The first string will be the matched word and the latter will become the character strings that are not yet processed. Then, the system recurs multiple times until the last character of the string, then splitting it into possible words, as shown in Fig. 3. If no word can be found due to typing error, it will either separate the string into single character or simply return the input as the output.

https://doi.org/10.31436/iiumej.v21i2.1408

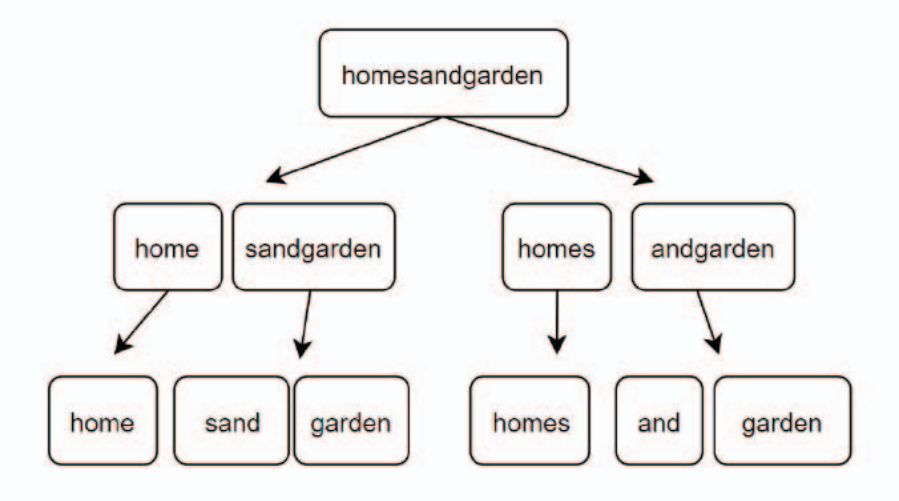


Fig. 3. Dynamic programming example

2.2.2. Word Matching Algorithm Using Probability Score

After the division of characters, the process of matching strings with words inside the text corpus is done using a word matching algorithm. Basically, when the user enters a string, the system will try to match the string with the words contained in the dictionary. Using a word dictionary is one of the common ways for creating a word segmentation model [7]. If the model finds a matched word, then that part of the string will be segmented. Then, the process is iterated until it reaches the end of the string.

The matching algorithm will also rely on the probability of the word as well as the whole sentence. As shown in Fig. 4, the frequency is stored together with the word in the dictionary. If there are multiple ways of segmenting the string, it will prioritize the word with higher probability. Also, longer words will be prioritized compared to shorter words. For example, the string "strawberry" will be segmented as "strawberry" instead of "straw" and "berry" although these two words are stored in the dictionary, and "berry" has a higher probability compared to "strawberry". This is because the system will consider the total probability of the string, which is the multiplication of probability for each word [4]. In theory, it can be calculated by using Equation 1 [4] shown below.

$$P(W_{1:n}) = \Pi_{k=1:n} P(W_k | (W_{1:k-1}) \quad (1)$$

Equation 1 is highlighted in Norvig [4] states that, based on the context of a particular word, the probability of a sequence of words $[P(W_{1:n})]$ is the product of the probabilities of all the preceding words $[\Pi_{k=1:n} P(W_k | (W_{1:k-1})]$ within the same context. It means that, for example, using a 5-word sequence – "Honesty is the best policy" - the probability of this word sequence would be the product of each word given the four previous words.

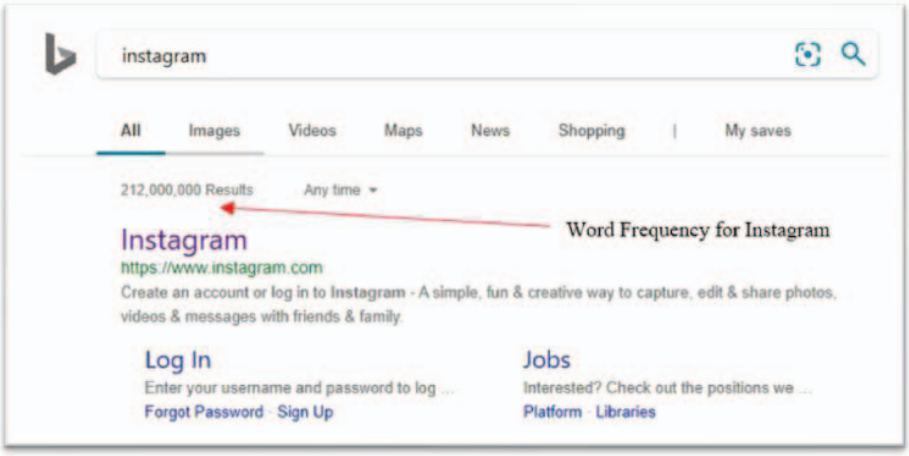


Fig. 4. The word '*instagram*' and its frequency is obtained using a search engine

2.2.3. Back-off Algorithm

For every word segmented, the algorithm will check whether there is any bigram probability for the next word to occur. A bigram dataset is used in the system to ensure better accuracy of the segmentation. However, since there are chances that the probability of the bigram is zero, a back-off algorithm is included in the system to ensure no error in probability calculation due to multiplication by zero [4]. The back-off algorithm will simply ignore the bigram probability and continue the segmentation using unigram probability, or the probability for every single word. However, one of the disadvantages in implementing this is that the system will take a longer time to match each word in the unigram database, whereas if the bigram probability score is included, the computation time may be shorter.

2.2.4. Text Corpus

Text corpus (plural: corpora) is a huge collection of structured text stored in electronic form. It is used as a dataset for the word matching process so that the string input can be segmented into the correct output. In this work, the text corpus is massively important because the matching word will solely rely on it for segmentation purposes. In order to obtain a highly accurate output, a huge file of text corpus or multiple collection of corpora is required. Therefore, the corpus is taken from the Google n-gram dataset where it contains billions of words that are collected by Google's users around the world, which is stored in multiple ways; a single word (unigram), 2-word phrase (bigram), 3-word phrase (trigram) and so on. Since the file size of the n-gram corpus is so big that it may consist of hundreds of gigabytes of memory, this work only considers the unigram and the bigram database. However, it is not difficult since all this data is provided by Norvig [4] and is also being used for other word segmentation models. Fig. 5 shows some of the words contained in the unigram and bigram dataset.

This text corpus also contains the word frequency for each word for the purpose of increasing accuracy to determine the correctness of a word for a certain input. As shown in Fig. 3, "home" and "homes" are both valid words that are the candidates for the output. Here, the probability is calculated to ensure that the word with the higher probability is chosen. The term "homes and" generally has higher word frequency compared to "home sand", hence the former is chosen.

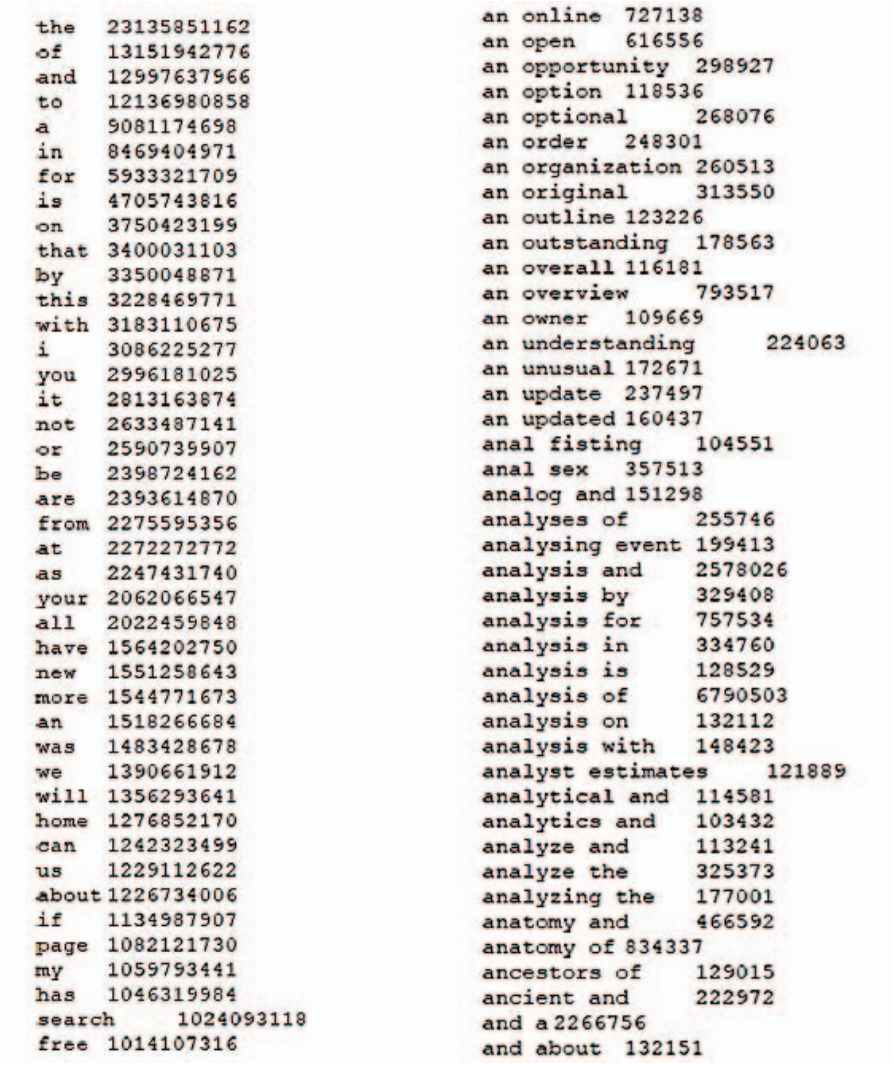


Fig. 5. Unigram and Bigram count inside the text corpus [4]

3. RESULT AND ANALYSIS

3.1. Python Programming

Python coding was developed specifically for word segmentation systems. It involves the process of loading the text corpora -unigram as well as bigram text- to improve the accuracy of word segmentation. When the user starts the program, it will automatically load all these three text files into the program and store every word and its frequency in the dictionary. This is completed in a couple of seconds when the program is executed. Then, the user can start entering the text using a keyboard.

Since this work is specifically made for the sign language device, it is programmed to accept only lower-case alphabet characters and numbers. All other buttons and symbols will be ignored, including the spacebar, enter, and delete buttons. If the user presses these keys, nothing will happen. However, for required functions such as 'moving to next line' or spacing between words, a delay time of 3 seconds is used as described in Fig. 2. Fig. 6 illustrates the graphical interface of the segmentation program. When the program is executed, two windows will be opened; one for the main windows for displaying the input and output and the other is to display the timer.

https://doi.org/10.31436/iiumej.v21i2.1408

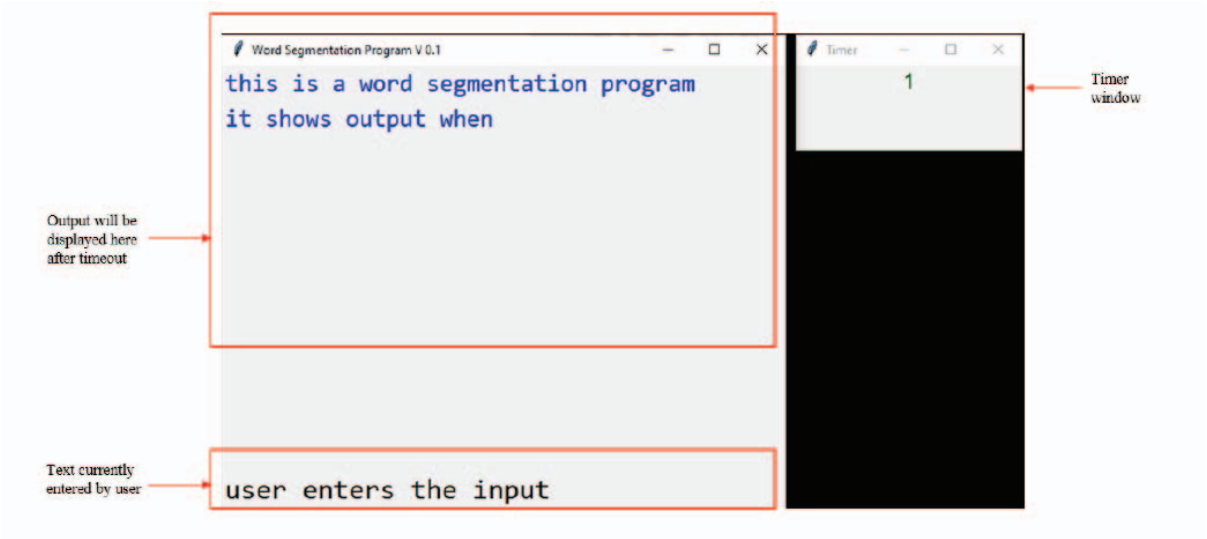


Fig. 6. The graphical interface of the segmentation program

By referring to Fig. 6, when a user enters the string "*thisisawordsegmentationprogram*", the program will automatically split the text and generate a correctly spaced text output. However, as mentioned earlier, the system will accept the input per character and the segmentation process occurs simultaneously as the user continues to key in the character. If the whole process were to be displayed, it would show a list of outputs as illustrated in Fig. 7, where it illustrates that during the segmentation process, the segmentation output may change while user enters the input; such as "word" and "words". This behavior is caused by the change of probability score of the whole sentence during the process. The system will choose the highest probability score for the whole sentence as the chosen output.

3.2. Challenges in the segmentation system

While testing the system, some errors in segmentation may occur due to the following issues.

3.2.1. Incorrect Input

One of the reasons for error in segmentation is the user may enter an incorrect character and due to the continuous mode of the system, the incorrect character will be stored in the string. As the system can neither delete the character in the string nor have any kind of error correction mechanism, a timer is used to tackle this issue as shown in Fig. 8. It shows that a user needs to wait for 3 seconds to allow a timeout, at which the system will reset the input and output of the system and move to the next line where the user can basically start again. However, the incorrect string or word will still be displayed and hence, this might not be an efficient way to fix the incorrect input error.

```
t
th
thi
this
this i
this is
this is a
this is aw
this is awo
this is awor
this is a word
this is a words
this is a words e
this is a word seg
this is a word segm
this is a word segme
this is a word segmen
this is a word segment
this is a word segment a
this is a word segment at
this is a word segmentati
this is a word segmentatio
this is a word segmentation
```

Fig.7. The word segmentation process flow

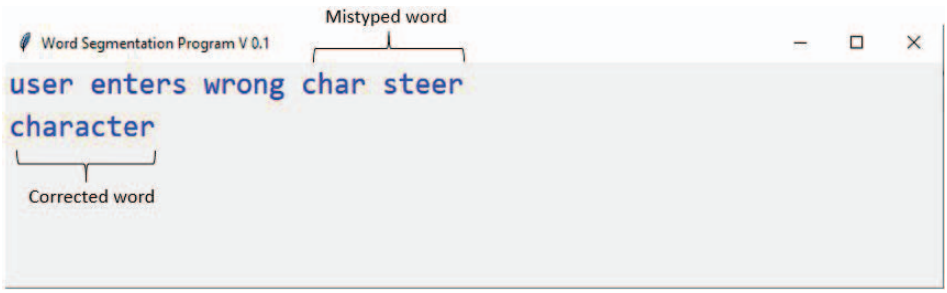


Fig. 8. The output of the segmentation process after timeout

3.2.2. Missing Words in Corpus

Another limitation of the system is the database used in this work. Although it contains 330,000 unigram words and around 250,000 bigram phrases, many highly specific words such as scientific terms and specific words like "*instagram*" are not available because the text corpora used were compiled in 2009. The system will also have issue in detecting uncommon human names and places, as these are too specific.

However, there are only a handful of words that are not stored in the corpora and it rarely needs

https://doi.org/10.31436/iiumej.v21i2.1408

to be updated. Nevertheless, a function to add words into the unigram or bigram corpus is added in the system. It is done simply by adding the word "*addunigram*" into the unigram file. When the system detects the user entering *"addunigram"*, it will interrupt the segmentation process, triggering a popup window that allows the user to key in the word and its frequency, as shown in Fig. 9 and 10

Fig. 9. The system does not recognize the word '*instagram*' and by using the command *addunigram*, a pop-up windows appears to key in the word and its frequency

Fig. 10. The system now recognizes the word '*instagram*'

4. CONCLUSION

In this work, a text segmentation algorithm for sign language was implemented using Python programming. The system is able to segment a string of characters correctly for all the input strings tested and thus, it is proven that the system is working properly as a word segmentation system for English words. Although, the system is unable to segment some specific words such as long scientific terms and special nouns, this can be fixed by adding a function to add words into the text corpus together with its frequency. One of the main limitations of this work is that the system is made specifically for a text-generating glove device assuming that it will only generate single alphabet characters. As of now, the system is not able to detect and process hand gestures in sign language that represents phrases such as "*hello*" and "*thank you*". Another observation is that the system can be a bit slow in producing the output if the string is too long (more than 30 characters) as it has to go through the words in the text corpus. Future work is required to address this challenge and in addition, the system may also include creating corpora for other languages that use Latin alphabets, such as the Malay Language.

REFERENCES

[1] G. A. Rao, P. Kishore (2016) Sign Language Recognition System Simulated for Video Captured with Smart Phone Front Camera, International Journal of Electrical and Computer Engineering (IJECE), vol. 6, no. 5, p. 2176

[2] Boon Giin Lee and Su Min Lee (2019) Smart Wearable Hand Device for Sign Language Interpretation System with Sensors Fusion, IEEE Sensors Journal pp (99):1-1

[3] Ahmad Zaki Shukor *et al*, (2015) A New Data Glove Approach for Malaysian Sign Language Detection, Procedia Computer Science, vol. 76, pp 60-67

[4] Norvig, Peter (2009), Natural Language Corpus Data

[5] W. Jiang (2015), Word-Segmentation [http://pypi.org/project/wordsegmentation]

[6] Sadiq Nawaz Khan *et al*, (2018) Urdu Word Segmentation using Machine Learning Approaches, International Journal of Advanced Computer Science and Applications(IJACSA), 9(6) http://dx.doi.org/10.14569/IJACSA.2018.090628

[7] D. Tanaya and M. Adriani (2016), Dictionary-based Word Segmentation for Javanese, Procedia Computer Science, vol. 81, pp. 208–213

[8] S. Srinivasan, S. Bhattacharya, R. Chakraborty (2012), Segmenting web-domains and hashtags using length specific models, In Proceedings of the 21st ACM International Conference on Information and Knowledge Management, Oct 2012, Maui.

[9] Y. Bassil, M. Alwani (2012), OCR Post-Processing Error Correction Algorithm using Google Online Spelling Suggestion., Journal of Emerging Trends in Computing and Information Sciences, vol. 3, no. 1

[10] S. P. Panda, A. K. Nayak (2015), Automatic speech segmentation in syllable centric speech recognition system, International Journal of Speech Technology, vol. 19, no. 1, pp. 9–18

[11] Y. Bassil, M. Alwani (2012), Post-Editing Error Correction Algorithm for Speech Recognition using Bing Spelling Suggestion, International Journal of Advanced Computer Science and Applications, vol. 3, no. 2

[12] S. S. Chawathe (2018), Lexical text segmentation using dictionaries, 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)

[13] Jenks, G. (2017), Python Word Segmentation [http://grantjenks.com/docs/wordsegment]