1.0 INTRODUCTION

The usage of computers has increased tremendously in various word processing tasks, therefore the need for an accurate and reliable spell-checking system has prompted the large amount of research conducted on this regard. Currently, there are two main spell-checking approaches, which includes offline spell correction and online spell correction (Gupta, et al., 2019). Offline spell correction approach performs spell-checking and correction after the completed enquiry is passed to the search engine or application. On the other hand, an online spell correction approach performs the spell-checking and correction when the user is the midst of typing. Although both approaches have produced interesting series of studies, the current project will focus on offline spell-checking approach.

The topic on spell checking has a long history in the field of computing, dating back to the 1960s. The remarkable study conducted by Damerau (1964) has demonstrated the usage of a spell-checking system and has also identified the common spelling errors made by humans. The study by Damerau (1964) revealed that approximately 80% of the spelling error fall into one of the following categories, which are: (1) Single letter is misspelled; (2) Single letter is missed; (3) An extra letter has been unintentionally inserted; (4) Location of two adjacent letters are transposed. In other words, the errors above are expected to occur when one misread, unintentionally mistype the words, or is simply unaware of the correct spelling for the intended word. Such errors that occur when users produce an error at the word-level are also known as non-word errors. Recent study by Chan (2016) also summarized the common errors as 4 main types of errors as shown in the table below.

Table 1.1 Common errors

Types of error	**Example of the error**
Deletion Error	higer > higher
Insertion Error	recommend > recommend
Substitution Error	abd > and
Permutation Error (Transposition)	heigth > height

Similar to what Damerau (1964) has highlighted, the four main errors are deletion error, insertion error, substitution error, and permutation error. Deletion error occurs when a single letter is missing in a string while insertion error occurs when an extra letter is added into a string. Furthermore, substitution error occurs when adjacent letters were mistyped. Lastly, permutation error occurs when the sequence of the letters is unintentionally swapped.

Meanwhile, another type of error which can also occur in many word-processing tasks is real-word error. Such errors occur when the error produces a word with correct spelling but will be deem inappropriate once the context of the sentence is considered. Regardless of the type of errors, two crucial steps that must be taken in order for a spell-check system to be complete, which are error detection and error correction. Nevertheless, it is also important to note that the techniques used to detect and correct for non-word and real word errors can vary due to the nature of the errors.

2.0 RELATED WORKS

Due to the long history of this field, numerous studies have been dedicated on this topic. Therefore, the common techniques used in past research will be summarized discussed in the following sections. The next few sections will be divided into the discussion of techniques used in error detection and followed by the discussion of error correction. The tools and technologies available will also be included in the discussion.

2.1 ERROR DETECTION

For the purpose of error detection, the two most common techniques used are dictionary lookup and *N*-gram analysis. As the name suggests, dictionary lookup method stores a dictionary that consists of words are deemed correct in memory and perform the spell-check using the stored dictionary. Specifically, when a body of text such as a sentence or paragraph is submitted into the spell-checking system, the words within the submitted text will be compared against the words within the dictionary (De Amorim and Zampieri, 2013). Following the comparison, the words that are not present within the dictionary will be flagged as incorrect. By looking at such method, it is expected that dictionary-lookup method can fall short when some uncommon words were present in the submitted body of text. In other words, there may not necessarily be any mistake with the rare words but are nonetheless flagged as they are not in the dictionary. Although larger dictionary can be used to improve the reliability, but this method also requires larger storage requirements and longer search time and consequently impose a huge burden on computing power. To overcome this issue, usage of dictionaries on specific fields such as economy or medical dictionaries may help to improve the reliability while reducing the need for storing large amount of words. In this case, only words that are commonly used in each field will be stored so that the reliability and computing power can be balanced. For the purpose of detecting non-word errors such as the errors delineated by Demerau (1964), the dictionary-lookup method is straightforward and easy to implement.

However, this method may not be feasible when it comes to the detection of real word errors. Real word errors are difficult to detect using dictionary-lookup method as the error words exist within the dictionary, but they disturb the overall meaning of the sentence (De Amorim and Zampieri, 2013). In order to overcome this issue, the addition of a confusion set alongside the dictionary-lookup method is proposed. Based on the research by Carlson et al. (2001), a confusion set is a set of words that are commonly confused with one another (i.e. there, their, theyre). The study by Carlson et al. (2001) used 265 confusion sets while Pedler

and Mitton (2010) used up to 6000 confusion sets. As a result, Pedler and Mitton (2010) reported 70% detection rate for real word errors when confusion set is used along with the dictionary-lookup method.

Aside from using dictionary lookup and confusion sets, *n*-gram analysis can also be used in a spell-checking system. An *n*-gram is sequence of strings or words depending on whichever figure *n* is set to. Instead of comparing each word in the dictionary, *n*-gram analysis conducts the inspection through a frequency matrix computed for all *n*-grams. In simpler terms, *n*-gram models predict the occurrence of a word based on the occurrence of the previous word(s). Furthermore, the number of previous word(s) will follow the "*n*-1" rule. For instance, if *n* is set to 2, it is a Bigram model. The bigram model will predict the occurrence of a word based on the previous word. Additionally, if *n* is set to 3, it will be a trigram model. A trigram model will predict the occurrence of a word based on the last two words (*n = 3* -1). Generally, *n*-gram analysis performs better than dictionary-lookup method in the detection of real-word errors as the overall context of the sentence is considered because the prediction of words is based on the previous word(s). Nevertheless, the implementation of such method can also be more complex than the dictionary-lookup (Islam and Inkpen, 2009; Xu et al., 2011).

2.2 ERROR CORRECTION

After the errors are detected, the spell-checking system moves on to the next step, which is error correction. In general, at this stage, several candidate words that are deemed likely to be the intended word will be presented for the user to choose. Furthermore, error correction can be further broken down into two sub-tasks which are the generation of the candidate words and the ranking of the candidate words (Gupta and Mathur, 2012). During the generation of candidate words, the words are selected from a list of valid *n*-grams. Then, the ranking is conducted by computing the similarity between candidate words and the error word.

Furthermore, the higher the similarity, the higher the candidate word will be ranked. In order to compute the similarity between strings, there are several common approaches that were used by past studies. The common methods include the usage of edit distance (Wagner and Fisher, 1974; Hulden, 2009), rule-based methods (Yannakoudakis and Fawthrop, 1983), *n*-gram analysis (Ahmed, DeLuca, Nurnberger, 2008), and neural networks (Hodge and Austin, 2003). All of the methods above perform error correction by computing the similarity between the error string and candidate strings.

2.3 EDIT DISTANCE

Edit distance measures the amount of operation cost required to convert one string to another string. The common edit operations include insertion, deletion, and substitution. As of now, there are several variations of edit distance used. Although each variation may have different edit operations and compute the edit cost differently, all edit distance-based methods assumes that candidate words with the highest similarity with the error word as the most likely correction. Levenshtein distance and Damerau's levenshtein distance are two of the most popular types of edit distance algorithm. Additionally, the Longest common sequence and Hamming Distance are also popular among the studies (Vogler, 2013; Loo, 2019). The illustration of each edit distance is explained below.

2.3.1 MINIMUM EDIT DISTANCE AND LEVENSHTEIN'S DISTANCE

The classic minimum edit distance and Levenshtein's distance assume that users usually make minimal mistakes therefore the selection of ranking of the candidate corrections will be performed by selecting strings that require the minimum amount of edit operations to convert from the error word to the candidate word. Both Minimum edit distance and Levenshtein's distance have the same three edit operations, which are deletions, insertions, and substitution. The main difference between the two is the cost of edit operation, specifically the cost of substitution. Minimum edit distance assumes that all edit operations have the cost of 1. Meanwhile, Levenshtein Distance states that the cost of substitution should be 2 because it involves the deletion and insertion of a character. The first example below converts the string "Insta" into "Inka". By using minimum edit distance, the total cost is 2.

$$\text{Insta} \xrightarrow{delete\ "s"} \text{Inta} \xrightarrow{substitute\ "t"\ to\ "k"} \text{Inka}$$

On the other hand, the total cost is 3 for the same conversion when Levenshtein's distance is used to calculate the edit distance.

$$\text{Insta} \xrightarrow{delete\ "s"} \text{Inta} \xrightarrow{substitute\ "t"\ to\ "k"} \text{Inka}$$

2.3.2 DAMERAU-LEVENSHTEIN DISTANCE

Slightly different from the Levenshtein's distance, the Damerau-Levenshtein's distance included transposition as one of the edit operation, so that the positions of the letters in a string can be changed. For example, the edit distance to convert "yx" into "xzy" is 2 which involves 1 transposition and 1 insertion via Damerau-Levenshtein's distance.

$$\text{yx} \xrightarrow{swap\ "y\ and\ x"} \text{xy} \xrightarrow{insert\ "z"} \text{xzy}$$

2.3.3 LONGEST COMMON SEQUENCE

The Longest common sequence measures the edit distance between strings by measuring the amount of common words that are in the same order. Furthermore, it also only allows for deletion and insertion of characters. As illustrated below, the edit distance is 3 to convert "Insta" into "Inka" using Longest Common Sequence where 2 deletions and 1 insertion are used.

$$\text{Insta} \xrightarrow{delete\ "s"} \text{Inta} \xrightarrow{delete\ "t"} \text{Ina} \xrightarrow{insert\ "k"} \text{Inka}$$

2.3.4 HAMMING DISTANCE

Hamming distance measure the edit distance between strings by calculating the number of bits that the strings differ. Due to this property, hamming distance can only be applied to strings that are of equal length. Besides, hamming distance only allows for substitution. As shown in the example below, to convert "kittens" into "sitting", the edit distance is 3 where 3 substitutions are required. The table below summarizes the types of edit operations and their respective cost.

$$\text{kittens} \xrightarrow{substitute\ "k"\ to\ "s"} \text{sittens} \xrightarrow{substitute\ "e"\ to\ "i"} \text{sittins} \xrightarrow{substitute\ "s"\ to\ "g"} \text{sitting}$$

Table 2.1 Types of Edit Distance

Variant	Edit operations (cost)			
	Deletion	Insertion	Substitution	Others
Minimum edit distance	1	1	1	-
Levenshtein's Distance	1	1	2	-
Damerau-Levenshtein	1	1	1	Transposition
Longest Common Sequence	1	1	-	-
Hamming Distance	-	-	1	Same length

2.4 RULE-BASED METHODS

Rule based techniques detect and correct errors by also assuming that human make several common spelling or typographic mistakes when typing (Yannakoudakis and Fawthrop, 1983). Therefore, by incorporating such rules into the algorithm, any words that satisfy the condition stated by the rules will be flagged as error. Interestingly, such rules are commonly input as the "reverse" of the common mistakes. So, each correct word resulted from the detection phase will be selected as the candidate word. Additionally, the rule-based methods can also be incorporated into other algorithms. In the study by Ahmed, DeLuca, and Nurnberger, (2008), the researchers improved the detection rate and processing time of an *n*-gram based spell-checker by incorporating rule-based techniques to capture common mistakes.

2.5 N-GRAM ANALYSIS

As mentioned above, *n*-gram analysis predicts the correct words through conditional probability. In this case, selection of candidate words can be done by computing the number of *n*-grams that are shared between the candidate words and the error word (Ahmed, DeLuca, Nurnberger, 2008). The shared *n*-grams between two strings is also known as the similarity coefficient, δ. The table below illustrates the calculation of similarity coefficient between two strings by calculating the number of bi grams ($N = 2$) they share. Based on the table below, the word "statistic" may be selected as one of the candidate words for the error word (sttistic) as it shares high similarity coefficient with the error word. Intuitively, the ranking of candidate words will also be based on the similarity coefficient. The higher the similarity coefficient, the higher the ranking.

Table 2.2: The calculation of similarity coefficient between bigrams

Bi grams	*statistic*	*sttistic*
st	1	1
ta	1	-
at	1	-
ti	1	1
is	1	1
st	1	1
ti	1	1
ic	1	1
Similarity coefficient, δ		6/8 = 0.75

2.6 NEURAL NETWORK

Although neural network is still regarded as a fairly new technique, the findings have shown some promising results. Based on the research by Sakaguchi, Duh, Post, and Van Durme (2017), the researches have used a recursive neural network as a spell checker. In this case, each word stored in a dictionary is represented as one node in a neural network. The findings revealed that the neural network performed better than most conventional spell checkers. However, this method only works well for smaller dictionaries as neural network can be difficult to train and require larger computing power.

2.7 TOOLS AND TECHNOLOGIES

Due to the large amount of work dedicated to this field of research, there are a variety of tools and libraries that can be used for a series operation required to build a spell-checking system. Moreover, as python is a scripting language that is more suited in building applications and programs, most NLP tasks are performed using Python language. Therefore, the libraries discussed will also be focused on the Python language.

One of the popular libraries for NLP tasks is spacy. It is an industrial-strength Natural Language Processing library (Navlani, 2019). It can handle large amounts of texts in natural language. Common NLP tasks such as summarizing, topic modelling, and name-entity recognition can all be achieved using spacy. In addition to the wide range of features, spacy is also known to have much shorter run time. Keras is also another popular NLP library. Specifically, it is an advanced neural network API that is written in Python. In comparison to spacy, Keras is more suited to build models with greater amount of customization. Furthermore, it can also run with TensorFlow, CNTK or Theano as the back end. It is popular among the researchers as it was developed to support rapid experimentation so that the researchers are able to translate their ideas into experimental results with minimal delay. Finally, TensorFlow is a common choice in the researchers and practitioners. It was developed and made free by Google's open source machine learning and neural network library. Having a wide variety of models and algorithms, TensorFlow can easily process a variety of neural networks including deep CNN and LSTM recursive models Although it produces decent performance and scalability, the threshold of implementation is relatively high.

3.0 CONCLUSION

As a summary, the spell-checking system consists of two main steps, which are error detection and error correction. Considering the advantages, using a dictionary alongside *n*-gram analysis may be a more robust method in detecting errors, especially when real word errors are involved. Moving on to error correction, the generation and ranking of the candidate words can be performed through an array of algorithms. Among methods such as rule-based method, neural networks, and *n*-gram analysis, edit-distance based methods may be a more parsimonious solution on this issue.

Moreover, with a simpler system, the results produced will also be more interpretable. In addition to that, the advantages offered by various libraries can be utilized by using a combination of the libraries. In this case, spacy may be used as the main NLP library. Meanwhile, by taking advantage of the greater customizability of both Keras and Tensorflow, they may be used to build spell-checking system tuned for the chosen domain.

REFERENCES

Ahmed, F., Luca, E.W.D. and Nürnberger, A., 2009. Revised n-gram based automatic spelling correction tool to improve retrieval effectiveness. Polibits, (40), pp. 39-48.

Carlson, A.J., Rosen, J. and Roth, D. (2001). Scaling Up Context-Sensitive Text Correction. *In Proceedings of the Innovative Applications of Artificial Intelligence (IAAI)*, Washington, USA, 7th – 9th August.

Chan, T. C. S., 2016. A context¬sensitive spell checker using trigrams and confusion sets. [Online] Available at: http://studentnet.cs.manchester.ac.uk/resources/library/3rd-year-projects/2016/tsz.chan-5.pdf [Accessed 4 February 2019].

Damerau, F.J., 1964. A technique for computer detection and correction of spelling errors. Communications of the ACM, 7(3), pp.171-176.

De Amorim, R.C. and Zampieri, M. (2013). Effective spell-checking methods using clustering algorithms. *In Proceedings of the International Conference Recent Advances in Natural Language Processing* (*RANLP)*, Hissar, Bulgaria, 7th – 13th September.

Gupta, N. and Mathur, P. (2012). Spell checking techniques in NLP: a survey. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(12), pp. 217-221.

Gupta, J., Qin, Z., Bendersky, M. and Metzler, D. (2019). Personalized Online Spell Correction for Personal Search. *In The World Wide Web Conference*, San Francisco, U.S.A, 13th -17th May.

Hodge, V.J. and Austin, J. (2003). A comparison of standard spell-checking algorithms and a novel binary neural approach. *IEEE transactions on knowledge and data engineering*, 15(5), 1073-1081.

Hulden, M., 2009. Fast approximate string matching with finite automata. *Sociedad Española para el Procesamiento del Lenguaje Natural*, 1135-5948.

Islam, A. and Inkpen, D. (2009). Real-word spelling correction using Google Web IT 3-grams. *In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, Singapore, 6th – 7th August.

Loo, M. v. d., 2019. Approximate String Matching and String Distance Functions. [Online] Available at: https://github.com/markvanderloo/stringdist [Accessed 5 February 2020].

Navlani, A. (2019). Tutorial: Text Classification in Python Using spaCy. [Online]. Available from: https://www.dataquest.io/blog/tutorial-text-classification-in-python-using-spacy/ [Retrieved: 7th Feb 2019]

Pedler, J. and Mitton, R. (2010). A large list of confusion sets for spellchecking assessed against a corpus of real-word errors. *In Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC'10),* Valleta, Malta, 17th – 23rd May.

Sakaguchi, K., Duh, K., Post, M. and Van Durme, B. (2017). Robsut wrod reocginiton via semi-character recurrent neural network. *In Thirty-First AAAI Conference on Artificial Intelligence,* California, USA, 4th – 9th February.

Vogler, R., 2013. Comparison of String Distance Algorithms. [Online] Available at: https://www.joyofdata.de/blog/comparison-of-string-distance-algorithms/ [Accessed 5 February 2020].

Wagner, R.A. and Fischer, M.J. (1974). The string-to-string correction problem. *Journal of the ACM (JACM),* 21(1), pp. 168-173.

Xu, W., Tetreault, J., Chodorow, M., Grishman, R. and Zhao, L. (2011). Exploiting syntactic and distributional information for spelling correction with web-scale n-gram models. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Scotland, UK, 27th – 31st July.

Yannakoudakis, E.J. and Fawthrop, D. (1983). An intelligent spelling error corrector. *Information Processing & Management,* 19(2), pp. 101-108.