

**GROUP ASSIGNMENT**

**Module Code : CT052-3-M-NLP**

**Module Name : Natural Language Processing**

**Intake Code : APDMF2207DSBA**

**Lecturer’s Name : Mr. Raheem Mafas**

**Group Members : Wong Zheng Qie (TP069702)**

**Lim Chun Cheak (TP068620)**

**Justin Ng Zheng Xin (TP069603)**

# **Table of Contents**

[**Table of Contents** i](#_Toc134979761)

[**1.0** **Introduction of Candidate Techniques** 1](#_Toc134979762)

[**2.0** **Available Python Libraries & Packages** 2](#_Toc134979763)

[**3.0** **Edit Distance** 5](#_Toc134979764)

[**4.0** **ASC System Design** 7](#_Toc134979765)

[4.1 GUI Design 9](#_Toc134979766)

[**5.0** **Implementation** 10](#_Toc134979767)

[**6.0** **Results** 10](#_Toc134979768)

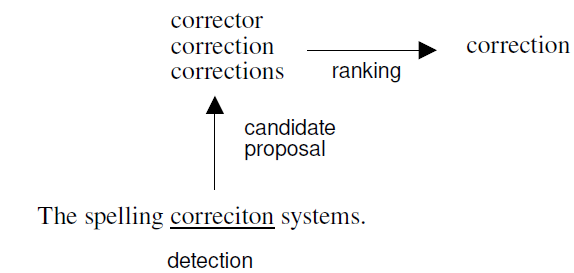
[**7.0** **Conclusion** 10](#_Toc134979769)

[**References** 10](#_Toc134979770)

# **1.0 Introduction of Candidate Techniques**

In the field of natural language processing (NLP), spelling errors are known as mistakes or variations in how words are spelled during written text. Some of these errors can come in the form of typographical errors (typos) or with a mixture of different languages (Singh and Singh, 2020). Generally, spelling errors can be classified into two categories, namely: real-word errors and non-word errors. Real-word errors occur when valid words in the dictionary are spelt wrongly, resulting in a different word. For instance, if the intended word in a text is “got” but typed “git” instead. Non-word errors on the other hand, are invalid words that are not in a dictionary (Hládek et al., 2020; Singh and Singh, 2020).

Spelling errors can pose challenges in NLP as misspelt words make text more difficult to read and process. On top of that, incorrect spellings can cause misunderstandings and even lose informational value of a text (Mishra and Kaur, 2013). As such, to address the problems of spelling errors in NLP, automatic spelling correction (ASC) systems were introduced to aid in the detection of incorrect words, followed by suggesting a list of suitable candidate replacement words based on their likelihood ranking (Hládek et al., 2020). The correct word spelling can then be selected interactively or automatically. Figure 1 below illustrates the process flow of an ASC system.



ASC systems predominantly features in search engines, text editors, as well as web browsers and applications (Hládek et al., 2020). The task of detecting and correcting spelling errors in ASC’s system are based on Spelling Correcting techniques, with each technique being more effective for detecting and correcting different types of error (Mishra and Kaur, 2013).

In recent years, researchers have been developing a variety of techniques for Spelling Correcting techniques such as *Minimum Edit Distance, Similarity Keys, Rule-Based Technique, phonetic algorithms, N-gram Based Technique, Probabilistic Technique* and *Neural Nets* (Mishra and Kaur, 2013). However, literature by Aouragh et al. (2015), Pande (2017) and Hládek et al. (2020) suggests that the minimum edit distance is the most applied technique for spelling correction with candidate generation. As a result, the minimum edit distance will be selected as the candidate technique for tackling spelling errors in this study, and a detailed description of its variation techniques will be explored in Section 3.0 below.

# **2.0 Available Python Libraries & Packages**

This study will utilize python programming language to develop the Spelling Correction component in the ASC system. Some of the available NLP python libraries to be implemented in the application of spelling correction via edit distance technique includes Natural Language Toolkit (NLTK), SpaCy and Genism (S. K. Singh and Sachan, 2019; Srinivasa-Desikan, 2018). The NLTK library contains most of the packages and tasks needed for text preprocessing, such as tokenization, spelling correction, Part-of-Speech (POS) tagging, stop word elimination and stemming (S. K. Singh and Sachan, 2019). Subsequently, SpaCy is another library that provides text preprocessing similar to NLTK but with more advance features like dependency parsing, named entity recognition and lemmatization; making SpaCy more appropriate for developing real-world applications as compared to NLTK, which is widely used for research or academic purposes (Srinivasa-Desikan, 2018). Gensim on the other hand, is used to construct a Word2Vec model for word embedding, which aids in the creation of word vectors to measure the similarity between words and suggest the most likely correction for a misspelt word based on its similarity to other words in the corpus (Pande, 2017).

Table 1 below summarizes the following NLP Python libraries, packages, tasks, functions, and descriptions for Spelling Correction, in relation to their respective library documentations:

1. NLTK - <https://www.nltk.org/>
2. SpaCY - <https://spacy.io/>
3. Gensim - <https://radimrehurek.com/gensim/auto_examples/index.html>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Library | Packages | Tasks | Functions | Description |
| NLTK | tokenize | Tokenization | *word\_tokenzie* | Used for word tokenization to split text into individual words. |
| *sent\_tokenize* | Used for sentence tokenization to split paragraph text into a list of sentences. |
| metric | Spelling correction | *edit\_distance* | To measure dissimilarity between two sets of strings via Levenshtein edit distance. |
| tag | POS tagging | *pos\_tag* | Assigns a grammatical category (e.g. noun, verb, adjective etc.) to each word in a text. |
| corpus | Stop word elimination | *stopwords* | To delete or filter out meaningless words from the text (e.g. "a", "an", "the", "is", "and” etc.) in order to reduce noise in text and make analyzing of meaningful words easier. |
| stem | Stemming | *PorterStemmer* | Original stemming algorithm developed to remove suffixes from words to reduce words to their stem (root) form, but it only supports English language. |
| *SnowballStemmer* | Newer version of PorterStemmer algorithm that can support various languages such as Arabic, French, Italian, Spanish, and more |
| *LancesterStemmer* | The most aggressive stemming algorithm among the three, as it is an iterative algorithm that applies certain of rules and conditions when stemming, without considering the context |
| SpaCY | spacy | Dependency parsing | *dependency\_parser* | To analyze a sentence's grammatical structure by recognizing word relationships such as subject-verb and object-verb relationships. |
| Named entity recognition | *entity* | To recognize and identify named entities in a text like people, location, company or brand names. |
|  | spacy.lang.xx ("xx" is the language code) | Lemmatization | *lemmatizer* | A more advanced version of stemming that considers the context of the text before reducing words into its base form. Thus, produces more accurate results than stemming |
| Gensim | models | Word embedding model | *Word2Vec* | Generate of word embedding models via neural networks, and create numerical representation of each word (word vectors). |

**Table 1: Summary of NLP python libraries and packages**

# **3.0 Edit Distance**

Edit distance also known as Levenshtein distance (LD) is a measure of similarity between two strings, the source and target strings (Balhaf et al., 2017; Haldar and Mukhopadhyay, 2011). Levenshtein distance can be determined by the number of insertion, deletion and substitution operations (edit operations) required to transform the source string to the target string, where the minimum edit distance representing the closest match between both strings (Haldar and Mukhopadhyay, 2011). Each edit operations in LD can be defined as the following (Mccoy and Frank, 2018):

1. **Insertion:** - Inserting a new character into a string
2. **Deletion:** - Deleting an existing string character
3. **Substitution:** - Substituting new characters from the existing string

Additionally, there are also several variations of edit distance algorithms that differ in terms of allowable edit operations performed. Damerau-Levenshtein distance (DLD) and Jaro-Winkler distance (JWD) are some examples of popular edit distance variations used by researchers (Balhaf et al., 2017; Maarif et al., 2014; Santoso et al., 2019).

Damerau-Levenshtein distance (DLD)

DLD is an improved version of the LD alogrithm, as DLD can resolve spelling errors that involve two switched letters. In other words, DLD supports letter transposition of adjacent characters, on top of the basic edit operations like insertion, deletion and substitution, whereas LD does not (Hládek et al., 2020; Santoso et al., 2019).

Jaro-Winkler distance (JWD)

JWD is an extension of the Jaro distance algorithm, which provides more weight to matching initial characters at the beginning of a string. It does this by calculating the distance between two strings' based on the number of matching characters and transpositions, in order to determine their similarity (Maarif et al., 2014). The JWD algorithm is usually used to detect duplicates in strings and applicable for short string comparison. The outputs of the JWD will return a similarity score between 0 and 1, with 1 indicating that the strings are identical and 0 indicating that they are completely different (Manaf et al., 2019).

Table 2 below summarizes a comparison of each variation with respect to their edit operations.

|  |  |
| --- | --- |
| Edit Distance Variations | Edit Operations |
| Levenshtein distance (LD) | * Insertion * Deletion * Substitution |
| Damerau-Levenshtein (DLD) | * Insertion * Deletion * Substitution * Transposition |
| Jaro-Winkler distance (JWD) | * Transposition |

**Table 2: Edit distance variations**

According to relevant literature review by Maarif et al. (2014), Santoso et al. (2019) and Huang et al. (2020), LD and DLD tend to result in promising accuracies for Spelling Correction systems, particularly DLD since it involves performing an additional edit operation. Hence, DLD will be utilized as the core edit distance technique in this study, as DLD algorithms are an improved version of LD and JWD are more suitable for short stings and duplication detection use cases.

# **4.0 ASC System Design**

This study’s ASC system is designed to tackle non-word and real word spelling errors related to the investment domain, by obtaining a corpus of 100,000 words from investment textbooks. This ASC system will also satisfy the following design requirements:

* To design a system that prompts users to type a short piece of text up to 500 words.
* To design a system that detects and corrects non-word and real word spelling errors.
* To design a system that highlights misspelt words and generates a list of suggested words according to ranking.
* To design a system that enables users to click on the suggested word manually.

The steps taken to reach this end goal is illustrated in the following process flow of Figure 2.

STEP 1: -

STEP 2: -

STEP 3: -

STEP 4: -

STEP 5: -

STEP 6: -

A picture containing text, diagram, plan, sketch

Description automatically generated

**Figure 1: ASC system process flow** (Hládek et al., 2020)

**Figure 2: ASC system process flow (Singh and Singh, 2020)**

## 4.1 GUI Design

# **5.0 Implementation**

HI!!

# **6.0 Results**

# **7.0 Conclusion**

# **References**

Aouragh, S. L., Yousfi, A., Sidi, U., Abdellah, M. Ben, & Hicham, G. (2015). Adaptating the Levenshtein Distance to Contextual Spelling Correction. *International Journal of Computer Science and Applications*, *12*(1), 127–133. https://www.researchgate.net/publication/273758433

Balhaf, K., Alsmirat, M. A., Al-Ayyoub, M., Jararweh, Y., & Shehab, M. A. (2017). Accelerating Levenshtein and Damerau Edit Distance Algorithms Using GPU with Unified Memory. *2017 8th International Conference on Information and Communication Systems (ICICS)*, 7–11.

Haldar, R., & Mukhopadhyay, D. (2011). Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. In *Web Intelligence & Distributed Computing Research Lab*.

Hládek, D., Staš, J., & Pleva, M. (2020). Survey of automatic spelling correction. *Electronics (Switzerland)*, *9*(10), 1–29. https://doi.org/10.3390/electronics9101670

Huang, G., Chen, J., & Sun, Z. (2020). A correction method of word spelling mistake for English text. *Journal of Physics: Conference Series*, *1693*(1), 1–7. https://doi.org/10.1088/1742-6596/1693/1/012118

Maarif, H. A., Akmeliawati, R., Htike, Z. Z., & Gunawan, T. S. (2014). Complexity algorithm analysis for edit distance. *5th International Conference on Computer & Communication Engineering*, 135–137. https://doi.org/10.1109/ICCCE.2014.48

Manaf, K., Gunawan, R., & Subaeki, B. (2019). Comparison of Carp Rabin Algorithm and Jaro-Winkler Distance to Determine The Equality of Sunda Languages. *In 2019 IEEE 13th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, 77–81.

Mccoy, R. T., & Frank, R. (2018). Phonologically Informed Edit Distance Algorithms for Word Alignment with Low-Resource Languages. *Proceedings of the Society for Computation in Linguistics (SCiL) 2018*, 102–112.

Mishra, R., & Kaur, N. (2013). A Survey of Spelling Error Detection and Correction Techniques. *International Journal of Computer Trends and Technology*, *4*(3), 372–374. http://www.internationaljournalssrg.org

Pande, H. (2017). Effective search space reduction for spell correction using character neural embeddings. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2*, *2*(1), 170–174. http://www.lingholic.com/how-many-words-do-i-need-

Santoso, P., Yuliawati, P., Shalahuddin, R., & Wibawa, A. P. (2019). Damerau Levenshtein Distance for Indonesian Spelling Correction. *Jurnal Informatika*, *13*(2), 11–15. https://doi.org/10.26555/jifo.v13i2.a15698

Singh, S. K., & Sachan, M. K. (2019). SentiVerb system: classification of social media text using sentiment analysis. *Multimedia Tools and Applications*, *78*(22), 32109–32136. https://doi.org/10.1007/s11042-019-07995-2

Singh, S., & Singh, S. (2020). Systematic review of spell-checkers for highly inflectional languages. *Artificial Intelligence Review*, *53*(6), 4051–4092. https://doi.org/10.1007/s10462-019-09787-4

Srinivasa-Desikan, B. (2018). *Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras.* Packt Publishing Ltd.