Logo, company name

Description automatically generated

**GROUP ASSIGNMENT**

**A SPELL-CHECKER SYSTEM TO**

**DETECT NON-WORD AND REAL-WORD**

**SPELLING ERRORS**

**STUDENT NAMES** : INTAN, BOFENG, MARZIEH, BOSHRA

**STUDENT ID** :TP067076, TP066667, TP067265, TP066719

**PROGRAMME** : MSC IN ARTIFICIAL INTELLIGENCE

**MODULE NAME** : NATURAL LANGUAGE PROCESSING

**MODULE CODE**  : CT052-3-M-NLP

**LECTURER NAME** : DR. HAMAM MOKAYED

**HAND-IN DATE** : 12/8/2022

**Table of Contents**

Abstract………………………………………………………………………………………………..………………………………3

1. Introduction……………………………………………………………….………….……3
2. Methodology………………………………………………………………………………6
   1. Minimum Edit Distance………………………………………………………………7
   2. N-gram……………………………………………………………………..…………8
   3. Laplacian Smoothing………………………………………………………………….9
   4. Evaluation Metrics…………………………………..………………………………..9
      1. Recall Measurements……………………..……………………………………9
      2. Precision Measurements……………………..…………….…………………10
3. Implementation……………………………………………..…………………….………11
   1. Experiment Setting………………………………………………………..…………11
      1. PyQt………………………………………………………………..…………11
   2. Standard Information Block…………………………………………………………12
   3. Corpus Pre-processing………………………………………………….……………13
   4. Main Module………………………………………………………...………………13
   5. Non-word Error Module…………………………………………..…………………13
   6. Real-word Error Module………………………………………….…………………14
   7. Highlight Module……………………………………………………………………17
   8. Correction Action Module………………………………………...…………………18
   9. Spelling Check Wrapper Module……………………………………………………18
   10. Text Edit Module………………………………………..……………………18
4. Experimental Results………………………………………….…………….……………19
   1. GUI Screenshots…………………………………………..…………………………19
   2. Qualitative Analysis……………………………………...…………….……………24
5. Discussion……………………………………………………...………………..…….…24
6. Conclusion………………………………………..………...…………………………….25

References…………………….……………………………….…..…………………………25

**Abstract**

Edit distances have been utilised for decades in applications like spelling correction. The great majority of edit distances are variants of the Levenshtein distance, which solely account for single-character edits. Non-word errors and real-word errors are the most frequent sorts of typographical errors an any language. Researchers have been working hard on the first error, while the second has gotten less attention. In this paper, we use Minimum Edit Distance (MED) and n-gram algorithms in our spellchecker system, followed by Laplacian smoothing. Our experimental results show that our system can detect and correct both error types, with an overall accuracy of 93%.

*Key words: Edit Distance, Spelling Correction, n-gram, MED Method*

1. **Introduction**

In this modern age of technology, humans have become increasingly dependent on electronic devices to compose, keep, and transmit text documents. Spelling errors made by humans when using these devices become inevitable and has therefore become a popular research topic in the field of Natural Language Processing (NLP) so as to analyse and revert those spelling errors either automatically or by listing down suggestions to the user. There are many possible reasons behind spelling mistakes made by humans such as vocabulary ineptitude, misconception, or mistyping of keys. Spelling errors can be categorised into *real-word errors* and *non-word errors* (sharma and Gupta, 2015). A real-word error is a when a word which exists in a language is used incorrectly in a specific context. On the other hand, a non-word error is when the word does not exist in a specific language.

The Minimum Edit Distance (MED) and N-gram technique are widely-used methods implemented in numerous research on effective spelling error detection and correction systems. Ahmad et al. (2018) reported the results and analysis of two experiments done using a large-scale Bengali N-gram model trained on an online newspaper corpus: Context-aware spelling correction and trending topic detection. With this volume of data comes a broad variety of difficulties, which are illustrated in the presentation. This model was evaluated using the perplexities of many years. In a test employing a context-aware spell checker, they received a F grade with an 86.6% failure rate. In another experiment, researchers were able to determine the most popular topics at a certain moment in time.

According to another research by Gledec et al. (2019), N-gram systems for the Croatian language have been created using an innovative approach for the creation of large-scale n-gram systems. Since 1993, Hascheck, a Croatian online academic spellchecker, has been extensively utilised as the world's largest text repository for the collection of n-grams and is freely available worldwide. As opposed to the public Google n-gram systems, n-gram filtering is based on dictionary criteria as opposed to cutoff criteria. The Croatian N-gram system is equivalent in size to some of the largest Google Version 1 n-gram systems, while requiring 12 years of gathering to reach this stage. The Croatian n-gram system is dynamic since it depends on a continually used service. Heaps' law enables a system dynamics model of n-gram count behaviour, yielding surprising results.

It has been recommended in another study by Singh & Singh (2019) that the bigram, trigram, and confusion set (CS) techniques were coupled in order to identify Hindi real-word mistakes. When computing the left bigrams, right bi-grams, and trigrams, the immediate left, immediate right, and the test word in Hindi are all considered. Using the Levenstein edit distance method, a list of the most ambiguous phrases is compiled. The last step is to calculate a composite score for the whole CS using bigram and trigram probability values. Using the derived composite score, a list of suggested alternatives for the incorrect term is created. A 2000-word Hindi text file was used to analyse and validate the proposed method, yielding outstanding outcomes. There is a range of 0.70 to 0.75 for precision, recall, and F-scores, with a total range of 0.80 to 0.85.

Vartani Spellcheck is a context-aware tool for correcting Hindi text that utilises a state-of-the-art transformer – BERT – and the Levenshtein distance algorithm, also known as Edit Distance. The researchers used a lookup dictionary and context-based named entity recognition to scan the text for misspellings. This approach was evaluated using a massive corpus of Ramayana text prepared by Tesseract OCR. Because of the inflectional nature of Indic languages like Hindi, Levenshtein Distance was selected over Longest Common Subsequence. Consequently, despite the fact that halants and vowels have a very similar look, even a little change in either might result in a misspelling. Levenshtein Distance (LD) is a better way for determining the similarity of two words in languages with such complicated morphological and inflectional structures. LD is used to discover the lowest number of character insertions, deletions, or substitutions necessary to turn an input string S1 into an output string S2. Assume that they have an input string S1 that must be transformed into an output string S2 using the MED. With an accuracy of 81%, the results demonstrate a significant improvement over previously established context-sensitive error correction systems for Hindi. In contrast, they illustrated how Vartani Spellcheck may be used to offer autocorrection recommendations while typing continuously in a text editor (Pal & Mustafi, 2020).

In this paper, we constructed a spellchecker system by implementing the abovementioned spelling error detection methods; MED and N-gram. By building a simple Graphical User Interface (GUI) and using these techniques, our system was able to detect non-word and real-word errors found in a text entered by a user, provide a list of suggested words to replace the detected spelling error, as well as display the MED value of those suggested words.

The rest of this paper is organised as follows; Section 3 covers the methodology used in our algorithm, Section 4 covers the implementation of our methodology, Section 5 covers the results of our experiment, Section 6 covers some discussion related to our system performance, and Section 7 concludes the whole paper.

1. **Methodology**

In our spellchecker system, we implement Minimum Edit Distance (MED) and n-gram methods, followed by Laplacian smoothing. Figure 1 shows the flowchart of our spellchecker algorithm.

Diagram

Description automatically generated

Figure 1. Flowchart of proposed spellchecker algorithm

* 1. **Minimum Edit Distance**

Edit Distance (ED) refers to the number of editing operations which is done on a word, causing it to transform into another word. There are four editing operations taken into consideration; *Insertion*, *Deletion*, *Substitution* and *Transposition* (Ristad and Yialinos, 1998). Table 1 shows the editing operations and their ED value.

|  |  |
| --- | --- |
| **Editing Operation** | **Edit Distance Value** |
| Insertion | 1 |
| Deletion | 1 |
| Substitution | 2 |
| Transposition | 2 |

Table 1. Editing Operations and their Edit Distance Values

The Minimum Edit Distance (MED) technique is a spelling error detection method whereby the spellchecker system will suggest a list of correct words with minimum edit distance with regards to the incorrectly spelt word, depending on the MED value specified by the algorithm. The following pseudocode represents the computation of MED (Ristad and Yialinos, 1998),

Initialization

D(i, 0) = i

D(0, j) = j

Recurrence Relation:

For each i = 1…M

For each j = 1…N

D(i, j) = min

Termination:

D(N, M) is distance

In our algorithm, we detect non-real words and suggest corrections up to MED value of 2, and real words and suggest corrections with MED value of 1.

* 1. **N-gram**

N-gram is a type of probabilistic language model which is used to calculate the probability of a word to be correct given the word(s) placed before it (Wei-Jen et al., 2005). It follows a chain rule of joint probability as shown in Equation (1), where the probability of words (w) 1,2…n to be in that order equals to

|  |  |
| --- | --- |
|  | (1) |

However, using Markov’s assumption for simplification, each component in the product is approximated as shown in Equation (2),

|  |  |
| --- | --- |
|  | (2) |

The considered amount of words occurring before the word depends on the value N specified. In the case where N =2, it is called a bi-gram model. This model considers more the context in which the word is used for, as the single word occurring before that word is taken into account. A Maximum Likelihood Estimate is used to estimate bigram probabilities as shown in Equation (3),

|  |  |
| --- | --- |
|  | (3) |

where c is the count of the word.

In the case where N=1, it is called a unigram model. This model does not at all consider the context in which the word is used in, as it only considers the word standing on its own. Therefore, to detect non-real words, we apply a unigram, while to detect real-words, we use a bi-gram. To detect and classify non-real word errors, the bigram model will find that the probability of the word appearing in the corpus will equal to zero (Wei-Jen et al., 2005).

* 1. **Laplacian smoothing**

There may be instances where a term in the chain of probability equals to zero, resulting in the joint probability to equal zero. This is undesirable and due to the nature of the conditional probability, Laplacian smoothing is implemented to any zero probability values (Mimi et al., 2022)

* 1. **Evaluation Metrics**
     1. **Recall Measurements**

We compute recall scores for correct words and wrong words separately (Starlander et al., 2002). The correct words recall score is calculated using Equation 4 by dividing the number of correct words in the text recognized by our application (true positives) by total number of correct words in the text (sum of true positives and false negatives),

|  |  |
| --- | --- |
|  | (4) |

The wrong words recall score is calculated using Equation 5 by dividing the number of wrong words in the text detected by our application (true negatives) by total number of wrong words in the text (sum of true negatives and false positives),

|  |  |
| --- | --- |
|  | (5) |

The recall scores indicate how representative our model is of the test dataset, which also represent the effectiveness of the model.

* + 1. **Precision Measurements**

We compute precision scores for correct words and wrong words separately (Starlander et al., 2002). The correct words precision score is calculated using Equation 6 by dividing the number of correct words in the text recognized by our application (true positives) by total number of correct words recognized by our application (sum of true positives and false positives),

|  |  |
| --- | --- |
|  | (6) |

The wrong words recall score is calculated using Equation 7 by dividing the number of wrong words in the text detected by our application (true negatives) by total number of wrong words detected by our application (sum of true negatives and false negatives),

|  |  |
| --- | --- |
|  | (7) |

The precision scores indicate how precisely our model is functioning. Whether the model is able to recognize correct words as correct and detect wrong words as wrong.

Based on the Recall and Precision measurements, we then compute F1-Scores for both correct words and wrong words using Equation 8 and 9 respectively,

|  |  |
| --- | --- |
|  | (8) |

|  |  |
| --- | --- |
|  | (9) |

The overall accuracy of our application is calculated using Equation 10,

|  |  |
| --- | --- |
|  | (10) |

1. **Implementation**
   1. **Experiment Setting**
      1. **PyQt5**

PyQt is a framework consisting of sets of Python bindings of over a thousand classes that can be implemented and ran on multiple operating systems such as but not limited to Windows, macOS and Linux and others. This framework is an amazing integration of Qt and Python that allows its user to enjoy its ease of use and simplicity (riverbankcomputing, 2022). Table 2 shows PyQt5 methods and classes which our system has utilised.

|  |  |
| --- | --- |
| *PyQt5 Class* | Function |
| *SpellCheckWrapper* | This wrapper has been used to generate a suitable GUI for the project, which displays a list of suggestions for a misspelled word, add a word to the dictionary and more. |
| *SpellCheckHighlighter* | This PyQt class is a syntax highlighter. Its goal is to automate the highlighting of misspelled words that is inputted by the user. |
| *SpecialAction* | The class specialAction takes in a parameter called QAction, which contains menus, status text, icons and more, to be utilized in a given interface. QAction is used to generate a word suggestion list for the misspelled words. |
| *mousePressEvent* | This action allows the user to right-click to fire a specified event, in this case is the list of suggestion words. |
| *contextMenuEvent* | The context menu displays the word suggestions of the user. |
| *createSuggestionsMenu* | This class allows for the creation of a menu out of the suggestion list with the help of specialAction. Once a word is selected, the correctWord method is fired. |
| *correctWord* | This method replaces the selected misspelled word with a suggestion. |

Table 2. PyQt5 Classes and Functions in proposed system (Adapted from qt.io (2022))

* 1. **Standard Information Block**

There are two types of errors, which are non-word error and real-word error. For the non-word errors, we need to correct them because they do not exist in English. Therefore, we call this process correction.

In our algorithm, *process correction* is used to handle detected non-word errors, and *process suggestion* is used to handle detected real-word errors. Process correction indicates a strong need for the word to be replaced as the system detects a non-existent word, and process suggestion indicates a lower need to replace the word as it depends on the context. The compulsoriness of replacing a word is illustrated by the colour of the underline under the words; blue for real-word errors, and red for non-word errors. These two processes can be represented using the information block below (qt.io, 2022).

For the correction,

A list of suggestions: [word\_info\_1, word\_info\_2, …]

For each word information,

word\_info\_i = corrected\_word\_i + ‘non-word correction with MED = ’, n

where, n represents the med between the i-th wrong word and the i-th corrected word.

For the suggestion:

A list of suggestions: [word\_info\_1, word\_info\_2, …]

For each word information: word\_info\_i = suggested\_word\_i + , ‘real-word correction’]

* 1. **Corpus Pre-processing**

During the pre-processing stage, we convert all uppercase letters to lowercase letters, remove all punctuation marks, special symbols from the text. With exception to periods, all numbers, punctuations, and line breaks are replaced with space. ‘<SOS>’ is added at the beginning of a sentence, and ‘<EOS>’ after each period. Afterwards, we remove the extra space in the sentence and divide the sentence into words by the space.

* 1. **Main Module**

This module contains the title, size and layout of our application, and is also represents the entrance of our program. The text editor is the main environment for interaction.

* 1. **Non-word Error Module**

This module is for non-word errors’ checking and suggesting.

To check whether a word is non-word error, we first compute the unigram language model. The unigram language model is equivalent to a vocabulary dictionary containing all the unique words in the dataset. If the input word is not in this vocabulary dictionary, we consider it as a non-word error. Otherwise, it is a correct word.

Algorithm 1 below shows the algorithm for non-word error checking.

Graphical user interface, text, application, email

Description automatically generated

To give word suggestions for a non-word error, we first generate all candidate words existing in the vocabulary dictionary which are one ED away from that word. If such candidates are not found, we enlarge the searching space to candidates which are two ED away.

Algorithm 2 below shows the algorithm for non-word error suggestion.

Text

Description automatically generated with medium confidence

Finally, a list of suggested words are obtained for a non-word error. These suggestions are wrapped to build a standard information block.

* 1. **Real-word Error Module**

Before introducing the module for real-word error checking, we first discuss the bigram probability score. We use this probability to measure whether there is a real-word error in the current word group.

To calculate the bigram probability, we first obtain the unigram language model and the bigram language model. Bigram probability is calculated using Equation (3).

The numerator denotes the frequency of the word group in the bigram dictionary, while the denominator denotes the frequency of the first word of the word group in the unigram dictionary. In the case where the word group or first word is non-existent, the probability value will equal to zero.

After we obtain the values of the numerator and the denominator, we perform Laplace Smoothing to smooth the probability distribution, where a value of 1 is added to the numerator, and (length of vocabulary dictionary + 1) are added to the denominator.

Finally, the bigram probability is calculated accordingly. If any numerator or denominator is 0, we return 0.0 as the probability. The algorithm for computing bigram probability is as shown in Algorithm 3 below.

Text, letter

Description automatically generated

To check whether a word has a real-word error, we first compute the bigram probability of the input word group. Then, we get the candidates of the first word in the word group which are 1 ED away from the first word. We traverse the list of candidates and concatenate the first word with each candidate. Finally, we compute the bigram probability of each combination. If there is a combination with a larger probability than the input word group, the system detects it as a real-word error. Otherwise, it will be a correct word group.

The algorithm for checking for real-word errors is as shown in Algorithm 4 below.

Text

Description automatically generated

Generation of list of suggestions for real-word errors are performed the same as in the non-word error module, with the difference being we save all the combinations which have higher probabilities than the input word group. Therefore, we get a list of candidates that are more appropriate. We then use this list of candidates as the suggestions.

The algorithm for suggestions for real-word errors is as shown in Algorithm 5 below.

Text

Description automatically generated

* 1. **Highlight Module**

In this module, we highlight the errors found in the input text. Red wavy lines will be drawn under non-word errors, blue wavy lines will be drawn under real-word errors. This function is called each time a character is typed into the GUI. After calling this function, the input sentences are split into words with the same order using a regular expression. Any number or special character in the sentences will be ignored.

Then, for each word, the non-word error module is ran to check for any presence of a non-word error. If so, the non-word error will be underlined in red colour. If it does meet the requirements of a non-word error, the real-word error module is ran, and presence of any real-word error is checked based on every word’s previous word. If so, the real-word error will be underlined in blue colour. At the end, based on colour, both types of errors will be clearly identified in the text editor.

The algorithm for highlighting errors is as shown in Algorithm 6.

Text

Description automatically generated

* 1. **Correction Action Module**

This module replaces the words selected by the cursor with the words we selected in the correction menu.

* 1. **Spelling Check Wrapper Module**

This module defines an interface class. By defining this class, we do not need to change other parts of the program when we want to use other spelling correction methods.

* 1. **Text Edit Module**

This module is where the logic of our application is defined.

In the GUI, any two types of errors found in the user input will be highlighted.

To correct a non-word error, the user should right-click on the highlight word. This will trigger an event that generates a menu containing a list of suggested words based on the selected word.

Same as for non-word error, the user should right-click on the word detected as real-word error to correct it. Based on the cursor’s position, the system will also consider the word appearing before the highlighted word. This will trigger an event that generates a menu containing a list of suggestions based on the selected word and its previous word.

The user can correct their spelling error by left-clicking on their intended word seen in the list of suggestions. This will trigger a correction event that replaces the original word in the text with the word they have selected.

Our program utilises code which can be referenced from NethumL (2021), Troublemeeter (2019) and AshwiniRangnekar (2018).

1. **Results**
   1. **GUI Screenshots**

To evaluate our spellchecker system, we present here two sample text inputs into the GUI from the user along with error detections and suggested words generated by the system. Figures 2(a)-(f) and Figures 3(a)-(f) shows screenshots of how our spellchecker works in the GUI based on some input text from the user.

Graphical user interface, text, application

Description automatically generated Graphical user interface, application

Description automatically generated

(a) (b)

Graphical user interface, text, application

Description automatically generated

(c)

Graphical user interface, application

Description automatically generated Graphical user interface, text, application

Description automatically generated

(d) (e)

Graphical user interface, text, application

Description automatically generated

(f)

Figure. 2 (a) Input text from user before spelling correction; (b) User right clicks on non-word error ‘indicwted’; (c) User right clicks on non-word error ‘posintion’; (d) User right clicks on non-word error ‘investemnt’; (e) User right clicks on real-word error ‘plan’; (f) Final text after user has replaced all spelling errors using system’s suggestions

Graphical user interface, text, application

Description automatically generated Graphical user interface, application

Description automatically generated

(a) (b)

Graphical user interface, application

Description automatically generated Graphical user interface, text, application

Description automatically generated

(c) (d)

Graphical user interface, text, application

Description automatically generated

(e)

Graphical user interface, text, application

Description automatically generated

(f)

Figure 3. (a) Input text from user before spelling correction; (b) User right clicks on non-word error ‘eventuall’; (c) User right clicks on non-word error ‘coul’; (d) User right clicks on real-word error ‘takes’; (e) User right clicks on non-word error ‘miney’; (f) Final text after user has replaced all spelling errors using system’s suggestions

Two error types were detected by the system; non-word errors underlined in red colour and real-word errors underlined in blue colour. When the user highlights and right-clicks on an error, the user will have an option to choose from a list of suggested words to replace the detected error. In the case of non-word errors, the MED of each suggestion will also be displayed as “(Non-word Error with MED: )” while no MED value for real-word errors suggestions are shown.

* 1. **Quantitative Analysis**

Based on the aforementioned evaluation metrics, we first draw the confusion matrix of our spelling checker in Table 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Prediction** | | **Total Number of Words** |
| **Positive** | **Negative** |
| **Actual** | **Positive** | 397 (TP) | 27 (FN) | 424 |
| **Negative** | 10 (FP) | 79 (TN) | 89 |
| **Total Number of Words** | | 407 | 106 | 513 |

Table 3. Confusion matrix of our spelling checker.

Then, we compute F1-scores and other evaluation metrics for correct words and wrong words using the equations mentioned above. The results are shown in Table 4.

|  |  |
| --- | --- |
| **Performance Measure** | **Result** |
| Correct Words Recall Score | 0.936 |
| Wrong Words Recall Score | 0.888 |
| Correct Words Precision Score | 0.975 |
| Wrong Words Precision Score | 0.745 |
| Correct Words F1 Score | 0.955 |
| Wrong Words F1 Score | 0.810 |
| Overall Accuracy | 0.928 |

Table 4. Performance Measures and Values

1. **Discussion**

Our spellchecker system scans every word to check for both non-word error and real-word error simultaneously, and was able to detect most errors, giving it a high accuracy of 93%. This feature demonstrates the effectiveness of our system. However, in earlier development stages, the system highlighted some correctly spelled, grammatically sound sentences as incorrect. This was due to the corpus not being inclusive enough to recognize certain words and their proper positions and the correct grammar in sentences. This challenge was overcome by utilizing another corpus with larger content. Another challenge was to integrate the desired interface with the developed algorithm in a way that makes it intuitive and seamless to the user. Deciding on the PyQt framework solved this issue.

1. **Conclusion**

In this paper, a spellchecker that distinguish between real-word errors and non-real word errors was created. The implementation phase was accomplished by utilising two famous techniques in natural language processing when it comes to spellchecking; Minimum Edit Distance (MED) and N-gram. The Python program works by allowing the user to enter a text through the designed GUI and highlighting any misspelled words or contextually incorrect words and displaying a list of suggestion for correct words that could replace the misspelled word. The algorithm yielded an overall accuracy of 93%. In future work, a larger corpus could be used to increase the flexibility of our algorithm and make it more robust.

**References**

Ahmad, A., Rub Talha, M., Ruhul Amin, M., & Chowdhury, F. (2018). Pipilika N-Gram Viewer: An Efficient Large Scale N-Gram Model for Bengali. *2018 International Conference on Bangla Speech and Language Processing, ICBSLP 2018*, 1–5. <https://doi.org/10.1109/ICBSLP.2018.8554474>

AshwiniRangnekar. (2018). GitHub - AshwiniRangnekar/GrammaticalErrorDetection-Correction: Spelling and grammatical error detection and correction using N-grams Model. GitHub. Geraadpleegd op 10 augustus 2022, van https://github.com/AshwiniRangnekar/GrammaticalErrorDetection-Correction

Gledec, G., Soic, R., & Dembitz, S. (2019). Dynamic N-Gram system based on an online Croatian spellchecking service. *IEEE Access*, *7*, 149988–149995. <https://doi.org/10.1109/ACCESS.2019.2947898>

Mimi, R. J., Masud, M. A., Rahman, R., & Dina, N. S. (2022, 24-26 Feb. 2022). *Text Prediction Zero Probability Problem Handling with N-gram Model and Laplace Smoothing.* Paper presented at the 2022 International Conference on Advancement in Electrical and Electronic Engineering (ICAEEE).

NethumL. (2021) GitHub - NethumL/pyqt-spellcheck: Spellchecking QTextEdit widget. GitHub. Geraadpleegd op 10 augustus 2022, van https://github.com/NethumL/pyqt-spellcheck

Pal, A., & Mustafi, A. (2020). Vartani Spellcheck – Automatic Context-Sensitive Spelling Correction of OCR-generated Hindi Text Using BERT and Levenshtein Distance. *ArXiv*.

qt.io (2022). *QSyntaxHighlighterClass.* <https://doc.qt.io/qt-5/qsyntaxhighlighter.html#details>

Ristad, E. S., & Yianilos, P. N. (1998). Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(5), 522-532. doi:10.1109/34.682181

riverbankcomputing (2022). *What is PyQt?* <https://www.riverbankcomputing.com/software/pyqt>

sharma, S., & Gupta, S. (2015). A Correction Model for Real-word Errors. *Procedia Computer Science, 70*, 99-106. doi:https://doi.org/10.1016/j.procs.2015.10.047

Singh, S., & Singh, S. (2019). Handling Real-Word Errors of Hindi Language using N-gram and Confusion Set. *Proceedings - 2019 Amity International Conference on Artificial Intelligence, AICAI 2019*, 433–438. <https://doi.org/10.1109/AICAI.2019.8701394>

Starlander, M., & Popescu-Belis, A. (2002, May). Corpus-based Evaluation of a French Spelling and Grammar Checker. In LREC.

Wei-Jen, L., Ke, W., Stolfo, S. J., & Herzog, B. (2005, 15-17 June 2005). *Fileprints: identifying file types by n-gram analysis.* Paper presented at the Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop.

Troublemeeter. (2019). GitHub - troublemeeter/spelling-correction: spelling corrector by edit distance and n-gram language model. GitHub. Geraadpleegd op 10 augustus 2022, van <https://github.com/troublemeeter/spelling-correction>