A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

Spell-Checking System Using NLP Techniques

CT052-3-M-NLP-092019-LGR

Natural Language Processing

APUMF1808BDSBA

Student:

SEYED HOSSEIN SHAHSAHEBI

TP053352

Lecturer:

PROF. DR. R LOGESWARAN A/L N RAJASVARAN

SUBMISSION DATE:

October 30, 2019

SpellCheckDoc:
Extensive Spell-Checking System

By Seyed Hossein Shahsahebi

Tp053352@mail.apu.edu.my

Table of Contents

Table of Figures

Table of Tables

1 Related Works

There are two main factors in the process of writing, which are Spell Checking and Grammar Checking. People may submit an essay or any written piece in any language without considering these aspects, especially when they are in a hurry to submit. The material with several spells and grammatical mistakes may prevent the reader from understanding the idea the writer wants to convey.

Spell checking process mainly divided into two main categories based on the literature (Gupta and Mathur, 2012; Singh *et al.*, 2016), including Non-Word and Real-Word errors. Non-Word errors refers to the misspelled word or a word that does not exist in the dictionary. In other hand, Real-Word is a word that is spelled correctly but its place in the sentence is not correct.

Spell checkers (Nejja and Yousfi, 2015; Singh *et al.*, 2016; Mandal and Hossain, 2017) tries to deal with these two issues by using different methods and algorithms for corpus cleaning, preprocessing, tokenization and etc. More simply, spell checkers need to perform two main tasks, which are error detection and suggestion prediction (Singh *et al.*, 2016).

Author in his article (Singh *et al.*, 2016), discussed two main methodologies to detect the error which need to be criticized. Dictionary lookup and N-grams technique introduced as the two main methods of detecting the misspelled word, which are mentioned in other studies as well (Gupta and Mathur, 2012). However, Singh claims that the dictionary is a data in a row and column format, which is a very restricted perspective as it can be stored in many different structures such as a set, list, key-value or even a text format. In other hand, he penalized the dictionary lookup and believes N-gram is sufficient for detecting the error. First of all, by assuming N as 1, the technique will be called Unigram and it can be considered as dictionary lookup as it will not detect any Real-Word error using unigram. In this situation, we assume the dictionary as a complete text file and words are stored next to each other. So, the dictionary lookup and bigram can be considered the same as the dictionary can have any format based on the implementation.

Before creating the dictionary, the corpus needs to be cleaned and preprocessed. Many different techniques are introduced throughout the literature. For cleaning purposes, extraction, stop-words removal, and stemming methods are general between many studies (Vijayarani *et al.*, 2015; Nayak and Kanive, 2016; Mandal and Hossain, 2017). Stop-word is a commonly used word in the text such as “an”, “the”, and etc. Removing these words may be handy for detecting Non-Word, but it causes a problem in Real-Word error detection and N-gram technique. Stemming is the process of converting a word into its root. There are many stemmer algorithms that are introduced and discussed in the literature (Vijayarani *et al.*, 2015).

After creating the cleaned and preprocessed corpus and detecting available errors in the input, providing suggestions is the next step. Minimum edit distance is the most popular method to calculate the difference between the existing words and the word with error (Gupta and Mathur, 2012; Singh *et al.*, 2016; Mandal and Hossain, 2017). In this process, Levenshtein algorithm calculate the distance between two words by three conducting three operations on the characters and count the number of modifications. These operations are deletion, insertion, and substitution. Damerau Levenshtein is another similar minimum edit distance algorithm (Nejja and Yousfi, 2015). The difference between these two algorithms is that the Damerau, includes the swap action. In other word, swapping two adjacent character in Damerau has a cost of one, where the same action in simple Levenshtein costs two actions (deletion for the first character and insertion for the first character after the second one).

In case there are more than one suggestion with the similar edit distance to the erroneous word, frequency can be used to remove ambiguity. In other words, among the similar distance, the word with higher frequency in the corpus will be selected (Singh *et al.*, 2016).

2 Dataset, Coding Techniques, and Libraries

In this section, we reveal the dataset that we used as a corpus for our study. Primary coding techniques are discussed and all third-party libraries and package that we benefited from are going to be introduced.

2.1 Dataset

Due to the increasing popularity and importance of the term "Cloud" in computing and daily developments, the primary corpus selected from the cloud-related book to initiate the dictionary with the ability to grow.

Our dictionary is created based on the book "Cloud Computing Bible" (Sosinsky, 2010) as a corpus of cloud-related words and phrases. The book has 531 pages in total which makes it a proper stand-alone corpus for this study.

In order to have a more generic overview about the text corpus, Table 1 shows the more detailed information about this corpus before applying any pre-processing methods.

Number of All Words	195,960
Number of Unique Words	10,978
Number of Non-Stop Word	116,627
Number of Unique Non-Stop Words	10,818

Table 1: Corpus details before preprocessing

Table 1 illustrated useful information about the corpus which indicates the rich repository of words. However, table 2 shows the most repeated words in the raw corpus. As it shows, there are several words with no special meaning and value that are repeated many times.

The fact depicted by Table 2, points to the need for preprocessing method. There is a need for a cleaning method to deal with several meaningless, worthless, and stop words. Our corpus cleaning and preprocessing methods are described in depth in Section 3.1, where we also showed the same information of Table 1 and Table 2 after applying the preprocessing methods. Also, the free version of this e-book is available online on pdfdrive[1].

Word	Frequency	Word	Frequency	Word	Frequency
…	8,679	12/1/10	1,058	System	728
Cloud	1,882	’	931	Computing	672
Service	1,508	Applications	855	Storage	630
Services	1,150	Google	827	Application	622
l	1,120	Web	812	Windows	571
pm	1,062	Data	741	Chapter	524

Table 2: Frequency of the most repeated tokens in the raw corpus before preprocessing. Red text implies a meaningless or worthless word.

2.2 Coding Techniques

In this section, the methodology and steps we took to implement the system will be introduced briefly. Each of the section we name here, will be elaborated in depth in Section 3. Introducing the implementation steps, prepares us to go through the required packages and libraries in Section 2.3 to deal with these steps and take the best decision accordingly.

The whole project is created in two different components. The first component is the core, where encapsulate the process of corpus cleaning, database transactions, suggestion selections, and other related core operations from the application level code which is a user of the core. The second component create the GUI and establish the connection between end-users and system core. This component is responsible to match the core component findings to the graphical representation of the SpellCheckDoc project.

Figure 1 show the flowchart of the scenario and this section introduces the first component steps briefly, which are elaborated in Section 4.2. An in-depth elaboration of second component is provided in Section 4.3.

[1] https://www.pdfdrive.com/cloud-computing-bible-e26793540.html

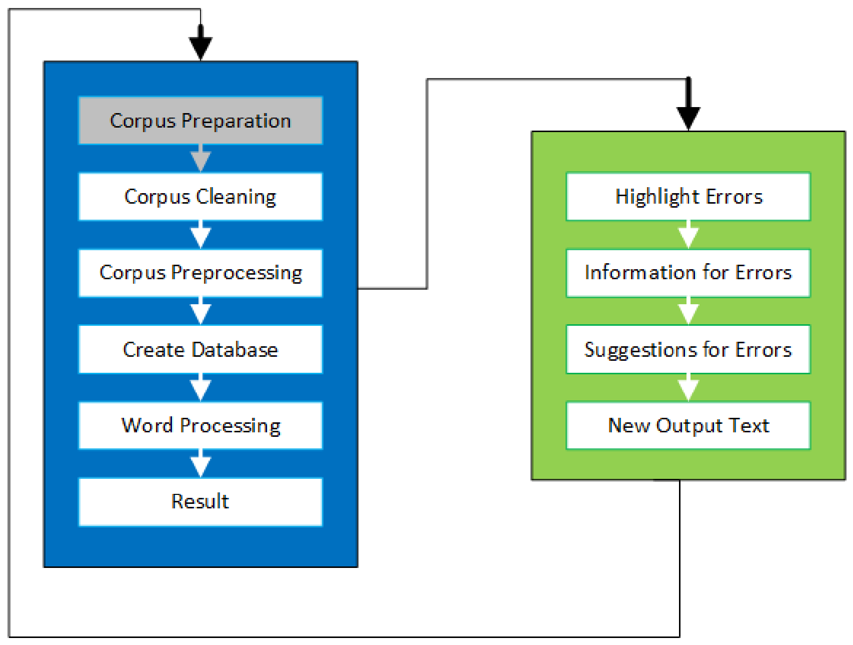


Figure 1: Components Diagram. Blue box represents the core component and green box is GUI component

Left box (blue) shows the core component of this project which receives an input from the initial corpus or the GUI input text. This component is responsible for analyzing each word and provide suggestions for words encountered an error because of spelling or bigram issues. The result of this component will be returned back to the GUI component. On the other hand, right box (green) is responsible for receiving the user's text and show him the possible errors in the text. It provides a rich environment for user to see the errors and use handlers to prune the detected errors.

2.2.1 Corpus Preparation

As we discussed earlier, the corpus for this study is created based on a cloud-related book which is accessed as a PDF e-book. In order to enable us to process its words, we need to convert this PDF e-book to a text file so we easily access to the content.

In Figure 1, this step is denoted as grey box, which refers to the fact that when GUI enters the cycle, this step will be put aside as the text will be provided by user in a text format and there is no need to convert any other file format to text. Technically, this step is not a part of core component and will be treated as a plugin.

2.2.2 Corpus Cleaning

The file which is converted from PDF format to the text file, or the raw input provided by user, may contain inappropriate content which needs to be pruned. This step mainly focuses on preparing the input text for the next step by removing empty spaces, lines, and other related processes based on the literature review. More in-depth explanation is provided in Section 4.2.1.

2.2.3 Corpus Preprocessing

After the text is cleaned and there is no unexpected token in the input text, core component tries to replace some general-purpose tokens such as URLs, dates, and times to a fixed token and prepare each eligible word in the input text for the next level. Further explanation on this step is provide in Section 4.2.2.

2.2.4 Create Database

This step receives a list of eligible words from preprocessing step. In this phase, eligible words will be stored in a database to be referenced later as a dictionary or bigram repositories.

In the first time that GUI is not in the cycle and we are initiating the system, every eligible word from the corpus are going to be stored in the database. After initiation and when GUI enters the cycle, words that exist in database will be returned, otherwise if the word is correct, it will be added to the database. This feature is elaborated in Section 4.2.4.

2.2.5 Words Processing

This section does not have any responsibility on the system initiation. However, when system is already initialized and GUI is functioning in the application's ecosystem, this step is responsible for analyzing the word against dictionary and bigram repository in order to detect possible errors and provide suggestions for replacement if applicable. All phases of word processing are discussed in Section 4.2.3.

2.2.6 Result

After every word in the input text is processed, the result of each word's analyzing process will be prepared and returned to the consumer of the core component.

2.3 Libraries, Packages, and Frameworks

In order to achieve the desired functionality, we had to use different sets of libraries and packages for core component. Also, to provide our rich GUI to the user, we used different frameworks and libraries to enhance the user experience.

This section goes through all the small-, medium-, and large-scaled third-party tools that we used in this project and elaborate each and every of them to illustrates the reason why the tool is used and what are its accommodations.

The order of the following headings is based on component, step, and importance. In other words, tools that are used in core components will be introduced first. Among the tools we used in the core component, tools that are used in the first step are higher in the list. Finally, in all tools that are used in a single step, those with more importance to the project have higher priority.

2.3.1 Python

Python programming language, is a powerful language which provides many Natural Language Processing related libraries and packages (Bird, Klein and Loper, 2009). This behavior, facilitates us to use open-source libraries which are tuned by community to handle a certain task. Available libraries reduce the implementation time and redirect our focus to the design part.

We are using Python 3.7.3 in this study which is Anaconda[2] distribution of Python.

2.3.2 PDF Miner

In order to convert the initial corpus from PDF to text, we are using pdfminer.six[3] library. This library is the best library to handle PDF document which forked the main PDF Miner library to add Python2 and Python3 compatibility (Stackoverflow, 2014). The required code to convert our PDF corpus to a text format is available in Section 7.3

[2] https://www.anaconda.com/distribution/
[3] https://pypi.org/project/pdfminer.six

2.3.3 NLTK

Natural Language Toolkit is one of the best NLP libraries written in Python. It provides a rich set of different methods to process and analyze the text. Being open source gave this library the opportunity of having a big community behind the project which helps the library to grow faster and includes more functionalities (Shaikh, 2018).

Word tokenizer, **sentence tokenizer**, and **stop words** are some of NLTK functionalities that we are using in our project. Using these methods, we will be able to prune the input text and prepare it for further analysis by tokenizing the large text into smaller portions and being aware of stop words in the text.

Pos tag is another module of NLTK that we are using in our project. This module enables us to detect the position of each word in a sentence such as noun, verb, and etc.

We used **Word Net Lemmatizer** module of NLTK in order to lemmatize each word. This module receives the word and its position in the sentence. Then based on the word's position it tries to convert the word to its root. This module is an endpoint which enables us to access to the wordnet[4] which is a powerful lexical database (Machine Learning Plus, 2019).

2.3.4 RE

RE is a built-in package in Python which enables us to write regular expressions and process the text. Regular expression is a fast and powerful method to search the text and fetch the matched phrases or perform special action on them (Friedl, 2006).

2.3.5 Collections

Collections is another built-in package in python which provides a container to store data in a special data structure. In this project we are using the **Counter** module of this package which let us to calculate the frequency of each word in the corpus.

2.3.6 Redis

Redis is a python library to use the actual Redis[5] in-memory database. Redis is an open-source and in-memory key-value database or caching system. It helps to store the dictionary in memory

[4] https://wordnet.princeton.edu
[5] https://redis.io

and have a fast access to the dictionary and bigram. Also, it has the capability to become a permanent database by storing an image of active instance into the disk.

2.3.7 Jellyfish

Jellyfish[6] is a python package which enables us to have access to multiple ready to use minimum distance algorithms including Levenshtein distance. This library is one of the most popular libraries for minimum distance calculation with over one thousand Github stars. Another reason to use this library among other available libraries is the fast speed due to the benchmark[7].

2.3.8 Flask

Flask[8] is a lightweight micro-framework for writing web application in Python. Flask and Django are the most popular web framework for Python. However, the reason for using Flask over Django is its simplicity and providing a lightweight environment to develop a web application hence access to required modules and functionalities.

And the reason for using Web to demonstrate the project is accessibility and simplicity. Users are able to use web application everywhere and on any device without installing a software except browser.

2.3.9 jQuery

jQuery[9] is a JavaScript web library which provides extensible power to manipulate web page and communicate with server using AJAX. Its simplicity and popularity are the reason of being chosen as the JavaScript library.

2.3.10 Bootstrap

We use Bootstrap as our CSS framework to improve the user experience and designing a responsive web page with intuitive design. Bootstrap is the most popular CSS framework based on the analysis of the most popular web frameworks[10].

[6] https://github.com/jamesturk/jellyfish
[7] https://github.com/robertgr991/fastDamerauLevenshtein
[8] https://palletsprojects.com/p/flask
[9] https://jquery.com
[10] https://www.ostraining.com/blog/webdesign/bootstrap-popular

3 NLP Techniques

In this section, we are introducing the techniques that we are using in this project. A proper justification for each selected technique will be provided and benefits and limitations of these techniques are discussed.

3.1 Extraction/Tokenization

Like many other studies, we try to divide our corpus to its lines using sentence tokenization and then divide each line into its words using word tokenization. By performing these operations, we will have access to all single words in our corpus which gives us more power to apply further operations ang get the most out of the word.

All input text will be converted to a list of lines and each member of this list contains a list of words. So, the words are still in a sentence format and we also have the ability to access to each of them directly.

This behavior enables us to use other word-level NLP techniques and this data structure is a unified format throughout our spell-checking process.

3.2 POS Tag

Part-of-Speech Tagging is the process of capturing the position of each word in the sentence. So, this step will use our unified data structure described above to access to sentence-level words and fetch their position in the sentence. This technique provides more than 30 word’s positions.

It also enables us to have two set of bigrams, where the first one is word bigrams in the corpus and the second one is positions bigrams based on the POS Tag and the corpus.

These advantages enable us to provide another spell error called RW-Ignored which refers to the absence of the word bigram and presence of positions bigram.

Although we did not implement decision tree in our study, this technique can help to form a decision tree based on the N-grams positions and build a powerful responsive spell-checking rules.

3.3 WordNet Lemmatization

We are using lemmatization over stemming because of its position awareness. Stemming will try to convert the word to its root without any specification. However, lemmatization considers the word position (e.g. verb, noun) and tries to convert the word to its root (Machine Learning Plus, 2019).

Using lemmatization enables us to have a more extensive dictionary by adding more words in case the word and its root are really different. In addition to this feature, we will not check a word against our dictionary if it can be converted using the lemmatizer. As the lemmatizer is able to convert the word to its root, it proves that the word is an existing word and can skip Non-Word error.

The limitation of WordNet lemmatization is a limited set of positions that it accepts as an input. In other words, from over 30 positions that we receive from POS Tag, we need to narrow it down to four main positions (e.g. verb, noun, adjective, adverb) and ignore the rest. Also, a recent project performed a benchmark over the WordNet lemmatization and it had a poor result. However, this project is still in its infancy level (LemmInflect, 2019).

3.4 Dictionary Lookup

Words will be stored in the dictionary to give us simple and quick access to serve the GUI and other non-graphical consumers of the core component. We can build the dictionary in our desired format to enhance the efficiency and simplicity of the system.

3.5 Bigram

Bigram is one of the most popular NLP techniques for spell-checking process. Regarding to other technologies we are using; we will have bigram of words in the corpus and bigram of word’s positions in the corpus.

Bigram tries to implement a context-wise processes and checking by looking back in the corpus for previous adjacent and it is simple to use and implement.

However, Bigram is not enough to perform an efficient context-wise text processing. Higher level N-grams can restrict the rules more and techniques with look forward pointer can be useful. These options will be considered for the future updates.

3.6 Minimum Edit Distance / Levenshtein

Minimum Edit Distance is the most popular techniques to calculate the similarity between two single words. This method is quite popular and easy to implement and use.

However, we will integrate it with bigram technique to provide more sensible suggestions and prevent further errors.

3.7 Frequency

In case we face the similar edit distance after integrating bigrams, the word’s frequency in the main corpus will be used to select the most appropriate suggestion out of the list of suggestion already selected.

This method is straightforward and easy to implement.

4 Formulation and Design

This section is responsible to elaborate on each and every functionality resides in the system, including core component and GUI. In each sub-section, we mention where the functionality resides in the system and how it helps to implement this project.

Before starting the elaboration, Figure 2 illustrates the study methodology where each section is discussed either implicitly or explicitly through the following sub-sections. The implementation and result are discussed in-depth in Section 5.1 and Section 5.2 respectively.

Further explanation of Figure 2 is provided in the following sub-sections which are designed to convey the whole system formulation and design to the readers.

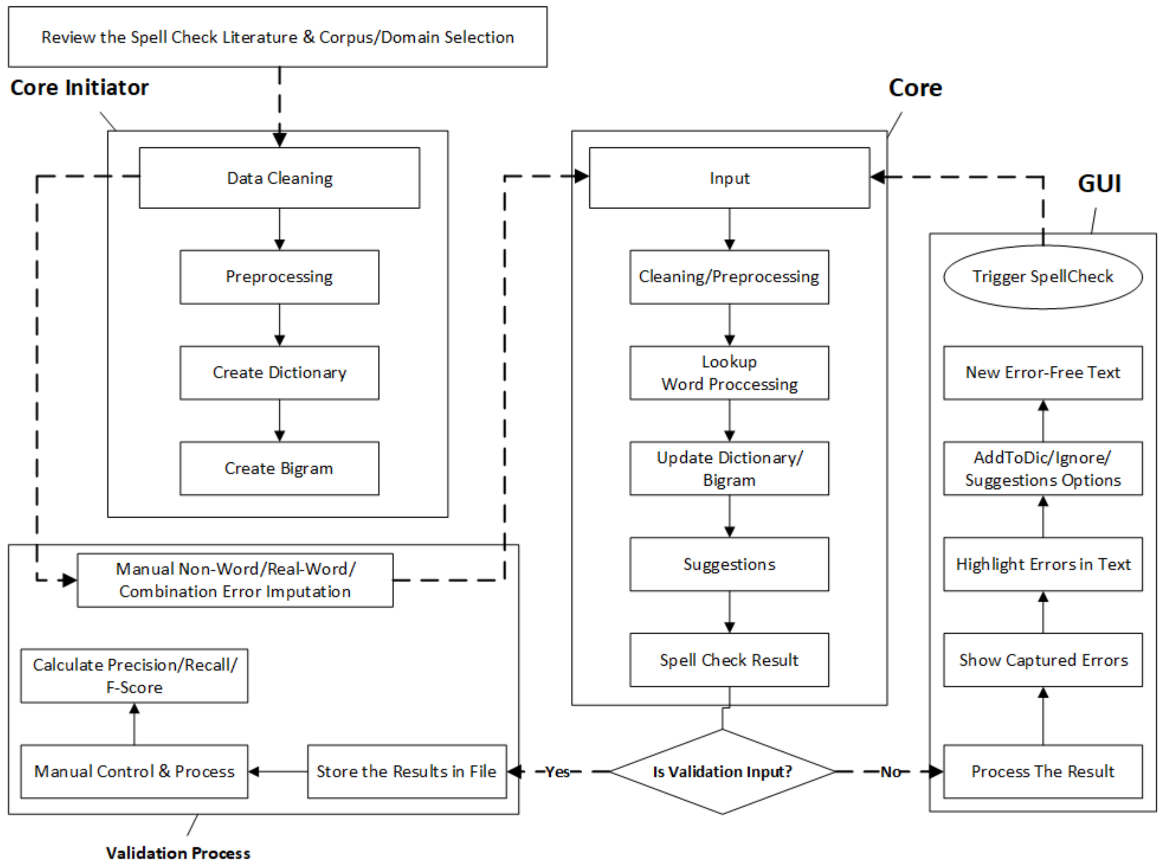


Figure 2: Study Methodology and Project flowchart. Steps of each component and interaction between components including test and analysis

In each sub-section there may be a reference to the next sections or sub-sections. The philosophy behind this behavior is to satisfy reader's curiosity if you need to jump to the details and usage back and force. Otherwise, you can consider those definitions with reference as a black box which exist and we can use them until you reach their related sections.

4.1 Core Initiator

This module has the responsibility of preparing and initiating the system so the end user or GUI can start functioning. In other words, it receives the corpus and creates the dictionary, bigram, and other required databases in order to make core component functioning properly. As Figure 2 depicts, validation process uses some random lines of the cleaned corpus as the primary sources for the validation purposes.

This module uses other modules actions to prune the initial corpus and prepare a clean corpus of text. After data cleaning and preprocessing steps, the text will be converted into the list of meaningful lines and each line will be converted to a list of meaningful words. These processes will be done using the same core component's corpus cleaning and preprocessing modules. These modules are elaborated in-depth in Section 4.2.1 and Section 4.2.2 respectively.

In addition, it uses dictionary creator module which refers a one stop action for word processing and creating the dictionary and bigram databases. The steps taken in word processing is discussed in Section 4.2.3 and the details about the database module is provided in Section 4.2.4.

The code written to initiate the core is available in Section 7.4 and Section 7.8 includes the database wrapper action written to initiate the system.

4.2 Core Component

The core component, as Figure 2 demonstrates, has the responsibility of checking the text spelling and report existing errors in the text in word-level format. In the following sub-sections, each step of the core component is elaborated in-depth.

4.2.1 Cleaning Step

The input of this step is a raw text which can be messy or well-organized. However, this module provides an output where all inappropriate words and lines are removed and each sentence is in its separated line. In other words, the output contains multiple lines where each line contains single pruned sentence.

Removing meaningless lines is the first action of this module. In this action, the corpus will be converted to array of its lines. Next, each line will be split into its words using a custom pattern. For each word, the eligibility of the word to be real will be checked against a custom pattern. In addition to eligibility, based on the corpus monitoring we are ignoring the word if it is equal to special URL[11] or if the line begins with a letter ‘l’. The mentioned URL is the address that we downloaded the corpus from and it is repeated many times in the text. While converting PDF to a text file, graphical shapes in list representation turn into ‘l’ in the text file. After removing ineligible words from the line, if the line is not empty or number of real characters[12] in the line are more than 40% of the whole number of characters in that line, the line will be reserved otherwise, the whole line will be removed from the output. The output of this action is a list of eligible lines based on the above processes.

Attach incomplete lines is the next action which receives worthless-free words and lines as an input. During PDF to text conversion, line terminations are not translated correctly and some lines are divided into multiple lines. This line division issues made some words that have dash in between to have extra space after dash in the end of the line. To fix this behavior, first the action tries to remove this extra space from the end of lines which end with ‘-’. Finally, the text will be separated into actual sentences using sentence tokenizer action of NLTK and a list of sentences with more than 5 characters will be returned as an output.

Finally, the desired text which contains corpus sentences in each line will be returned or could be stored in a file.

Imputing bigram symbols is the next action which create a desired output to be used for handling bigrams. The input can be a file which is created by cleaning action above or an input text where each line contains a single sentence. This action tries to replace some general-purpose dynamic phrases with a fixed phrase to be handled easier without loss of meaning. This action will add “|STARTSENT|” and “|ENDSENT|” at the beginning and at the end of each sentence.

[11] www.it-ebooks.info
[12] Alphabets, Apostrophe, and -

Moreover, it replaces different format of dates, times, and URLs with “|DATE|”, “|TIME|”, and “|URL|” tokens respectively using written patterns. Finally, the action stores the output in a file or it can return the output directly.

The actual code written to manage these actions can be accessed in Section 7.6.

4.2.2 Preprocessing Step

This module contains multiple actions which try to prepare the input to be used for another consumer action. Therefore, there is no separated action as a preprocessing step and these actions are used in many steps by other modules and actions such as the actions, we mentioned in the cleaning step. However, all used actions will be elaborated when a specific action or step uses it in its related sub-section. To have an overview of this module, its Python code is accessible in Section 7.7.

4.2.3 Lookup / Word Processing Step

Lookup or Word Processing module is responsible for checking a word to find any possible spelling error defined in the project.

Load raw text is a single action of this module which perform several actions required for the direct user of core component. This action receives a raw text or primary file path which contains the raw text. It applies corpus cleaning and bigram imputation to prune the raw text and prepare the ready-to-process output. After all, this action uses “**fetch line words**” action from preprocessing module. This action receives a list of pruned lines, remove punctuations from each line, divide the line into words using word tokenize module of NLTK, fetch the position of each word in the sentence/line using pos tag module of NLTK, and returns a list of tuples for each line. The output contains a list of lines and each line contains a list of tuples where each tuple contains the word and its position in the sentence.

Validate word is another important action of this module which receive a tuple representing the word and its position and a tuple representing previous word and its position. These tuples can be accessed from the output of the previous action. The responsibility of this action is to check the word for any possible error defined in our project by using sub-actions of this module. For each provided word, this action uses our preprocessing module and **lemmatizer** module of WordNet from NLTK to find the lemmatization of the word based on its position (e.g. watched as a verb will be converted to watch).

This action first checks the existence of the provided word in the dictionary using "**word exists**" sub-action. This action receives the word and its lemmatization as input and try to fetch it from the dictionary using dictionary module which is elaborated in Section 4.2.4. If the word exists in the dictionary it returns true, otherwise if the word and its lemmatization are really different (e.g. not differ in just s or es) it means the word is a real word which WordNet was able to detect it. If they differ, action tries to store this new word in our dictionary and return true, otherwise it returns false, which means that the word does not exist in our dictionary and it triggers ***Non-Word*** error.

If previous sub-action returns true, the main action tries to check the position of the provided word based on the provided previous word and our bigram database which is created using the corpus. "**Word in real place**" sub-action receives the word and previous word tuples as an input. It tries to load the bigram info of the provided word and look for the provided previous word in the list of word's previous words fetched from bigram database. If these actions are successful, the sub-action returns true, which means that the word is in a correct place, otherwise it returns false and it triggers ***Real-Word*** error.

If previous sub-action returns true, the main action tries to check the case of the word (e.g. can be in uppercase). "Word in correct case" sub-action receives the word and check if the provided words match the correct case based on the representations of the word that we had in our corpus. This sub-action is mostly responsible for detecting uppercased words error (e.g. aws should be AWS) or lowercased words error (e.g. BOOK should be book/Book). It returns true if the provided word's case matches the corpus representations, otherwise it returns false and triggers ***Case*** error.

If ***Real-Word*** has been triggered, "**Word can be real**" sub-action performs an additional check to see if the word can be in the correct place or not. This sub-action search to see whether the provided previous word's position exists in the provided word's previous position fetched from our bigram database or not (e.g. adjective can come before noun). If this sub-action returns false it keeps the error as *Real-Word*, otherwise it changes the error to ***RW-Ignored***, which means the word can be considered in a correct place based on its position in the sentence (e.g. adj), although the bigram does not exist due to our corpus.

If the triggered error is ***Non-Word***, ***Real-Word***, or ***RW-Ignored***, the main action uses our suggestion module, which is described in Section 4.2.5, to fetch suggestions from the dictionary if applicable.

All written code to manage these actions and lookup module are accessible in Section 7.10.

4.2.4 Dictionary Module

This module provides required set of actions to interact with the Redis database. Actions are self-explanatory and are accessible in Section 7.9. These actions are responsible for storing and retrieving various data to and from our database. However, the most important part is the database design which we will cover in this section. The schematic that is going to be explained, designed based on the Redis database limitations and powers and the way we need to access the data.

The whole database is divided into four sections to store different data specified by their key. These keys and separations are as follow:

- **common_words:** The key stores a set of most common words repeated in the main corpus.
- **lencat_{*LENGTH*}:** These keys pattern store a set of words that match the length of the word (e.g. lencat_3 contains “big”). These keys are used for the suggesting functionality where we select the suggestions based on the provided word’s length. In addition to the actual word, its lemmatized format will be stored in the relevant key.
- **bigram_{*WORD*}_frequency:** These keys store the frequency of each word in the main corpus. We use these keys in the suggestion module where edit distance of several words is the same and we need to sort them based on their frequency.
- **bigram_{*WORD*}_pos:** These keys store each word’s positions in the text based on the corpus (e.g. noun, plural noun, …). These keys are used in word validation action to check if the word is in a correct place.
- **bigram_{*WORD*}_prev_pos:** These keys store a set of previous words’ positions of each word based on the corpus. These values are used by word validation action to check if a word position can be considered as a correct place.

- **bigram_{*WORD*}_prev_words:** These keys store a set of previous words of each word based on the corpus. These values are used by word validation action to check whether a word is in a correct place or not.
- **{*WORD*}:** These keys have the same name as the word or its lemmatized version and store the case of the word based on its representations in the corpus. These keys are used to check whether a word is written in a correct case or not. The value of these keys has to be one of three available options. These options are as follow:
    - **0:** The word can be written either in uppercase and lowercase format.
    - **1:** The word can be written only in uppercase format.
    - **2:** The word can be written only in lowercase format.

4.2.5 Suggestion Module

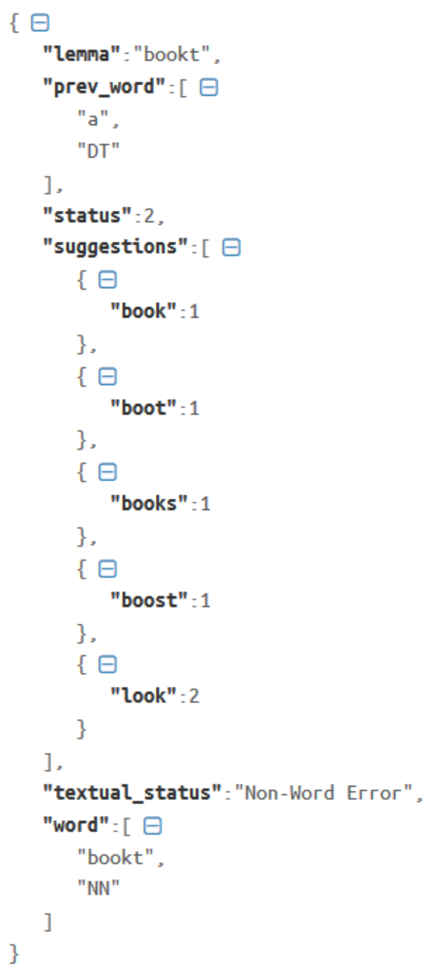
This module is responsible for providing available suggestions of a problematic word. "**Get suggestions**" is the only action of this module which tries to return maximum of five suggestion with lower than edit distance of three by default.

This action receives the word and the previous word as inputs. Then it tries to load the words in dictionary that differ from the provided word by a maximum of two characters (e.g. words with length of 2 to 5 will be fetched for the provided word "big"). The Levenshtein minimum edit distance algorithm will be executed on the fetched words and word with maximum edit distance of 2 will be selected in this phase. Among the selected words, those that have provided previous word in their previous words fetched from the database, will be selected. After all these filtering, the list of available words will be sorted based on their edit distance at first and then their frequency. A list of the five top words will be returned as an output.

The written code to manage suggestion module is attached in Section 7.11.

4.2.6 Core Component Result

After performing all required actions on each word, the data shown in figure 3 will be returned back to handler. So, handler decides what errors and which errors to show.



```
{ ⊟
   "lemma":"bookt",
   "prev_word":[ ⊟
      "a",
      "DT"
   ],
   "status":2,
   "suggestions":[ ⊟
      { ⊟
         "book":1
      },
      { ⊟
         "boot":1
      },
      { ⊟
         "books":1
      },
      { ⊟
         "boost":1
      },
      { ⊟
         "look":2
      }
   ],
   "textual_status":"Non-Word Error",
   "word":[ ⊟
      "bookt",
      "NN"
   ]
}
```

Figure 3: Example output of core component for the word bookt which has the position of noun in the processed sentence

4.3 GUI Component

GUI component send the user provided text to the core component and receives the core result. It processes the result and shows proper notification in a graphical representation to the user.

4.3.1 Result Process

In this section the result returned from the core component will be analyzed and the result for the problematic words will be kept and sent to the UI handler. It also ignores the consecutive errors if the first problem is *Non-Word* or if the first and the second problems are *Real-Word* or *RW-Ignored* errors.

This section is also responsible to receive and patch the user's spellcheck request and send it to the core component. The codes written for this process is attached in Section 7.12.

4.3.2 Results Demonstration

After results are processed and returned to the UI, this section is responsible to show the errors to the user. User is able to see all the notifications and easily identifies detected errors in the text. By clicking each highlighted error, he will be given a description of the error and any available suggestions to correct the error. Also, he will be given an option to ignore the error for his current session or add a *Non-Word* error to the dictionary. In this section, user will be able to see the dictionary most repeated words by default and ability to search desired word in the dictionary.

The codes written for this section is all in CSS, HTML, and JavaScript. These codes are available in Section 7.13, 7.14, 7.15, and 7.16. Also, more graphical representations of this section are shown and discussed in Section 5.1.

4.4 Validation Process

After the system has been implemented, we will take a semi-manual approach to evaluate the model and calculate precision, recall, and F1-Score of the designed system in three mode. The primary input will be fetched from the cleaned corpus and some errors will be introduced to these files manually and then it will use the core component to check the spell of the supervised input. These three options are:

- ***Non-Word*** errors where some misspelled tokens will be created inside the text. We use three different approach to validate each file (same text)
    - A file with random imputed errors
    - A file with consecutive errors
    - A file with errors to the critical spots
- ***Real-Word*** errors where some correct words will be placed in a wrong spot based on the database. Similar validation approach exists for this type.
- ***Combination*** errors produce a combination of Non-word and Real-Word issues in the same time and using the same validation approach mentioned above.

Accuracy factors such as recall, precision, and F1-Score will be calculated using the result from each option based on the three executed validation process.

5 Implementation and Results

This section is discussing the implementation phase and shows various features of the system using the designed GUI. In addition, it will show the validation process result done to evaluate the system and calculate its accuracy. Also, shortcoming of the prepared project will be discussed in this section.

5.1 Implementation

The methodology of implementation of different system components pretty covered in section 4. In the mentioned section we demonstrated the way the system works and how we designed and implemented core initiator, core component, GUI component, and model validation process. In this section, we try to depict various features that described earlier using the implemented GUI.

Figure 4 shows the primary interface of the application. In the very right side, user will be able to see the most common words in the dictionary as the page loads. Also, he will be able to search any word to see whether the word exists in our dictionary or not. In the most left side, user will be given a text input to write or paste his raw text that need to be checked against spelling errors. By clicking the green play button, user initiates a request to check the provided text against our dictionary for any possible spelling error. If any issues or suggestions found, it will be listed in the middle section to enable user to have an overview of detected problems. The bottom section, illustrated the status bar. In the status bar, user will see the proper details based on his actions. For instance, while user starts typing or pasting the raw text, status bar shows the number of words. Upon the completion of writing and clicking the play button, user will be informed about the process and when the results are fetched, the user can see the number of processed words and the number of seconds application took to check the provided text. Also, details about suggestions number will be provided in this section. After all, the status bar shows the number of words resides in the dictionary.

Figure 5 illuminates the GUI after user receive his suggestions based on the provided text and initiation request. Each type of error introduced earlier, will be illustrated in specific color. As shown, *Non-Word* error is depicted as red and *Case* error demonstrated as blue. It took almost five seconds for the application to process the text and provide four suggestions. As shown in the status bar, although 14 words have been typed, 18 words are processed to provide the result. These four additional tokens are “|STARTSENT|” and “|ENDSENT|” tokens added to the beginning and end of each sentence respectively. By clicking a box in the suggestion section or a highlighted word in the text box, details information about the error in addition to suggestions if any, will be shown to user. These behaviors are shown in Figure 6 to Figure 9.

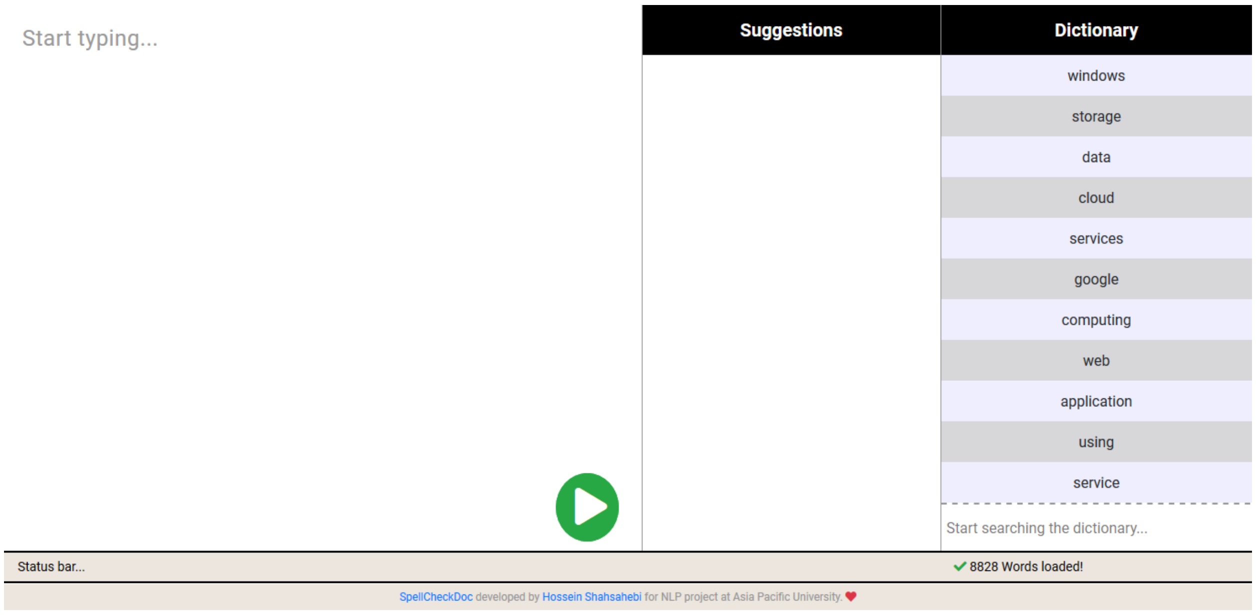
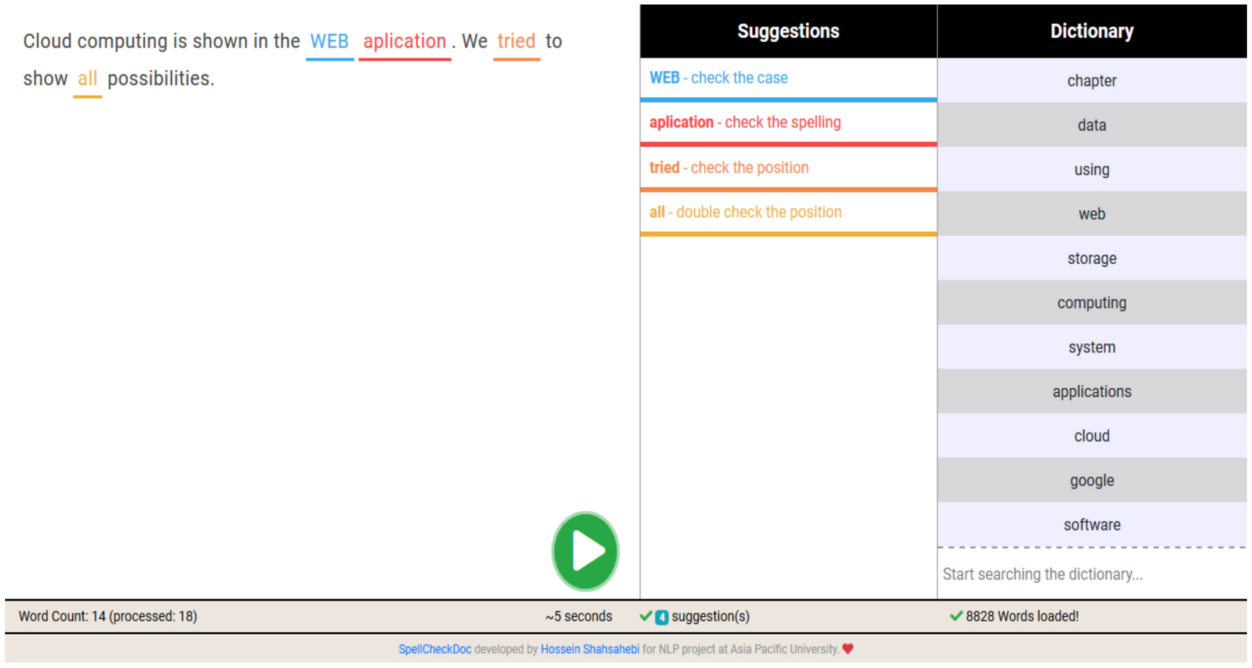
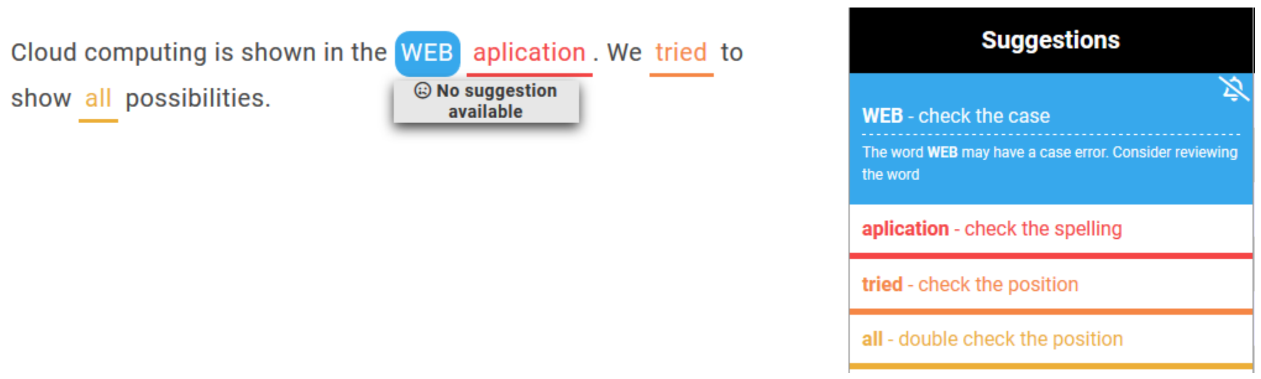


Figure 4: Primary GUI of the application when user open the web application



Cloud computing is shown in the WEB aplication . We tried to show all possibilities.
Suggestions
Dictionary
WEB - check the case
aplication - check the spelling
tried - check the position
all - double check the position
chapter
data
using
web
storage
computing
system
applications
cloud
google
software
Start searching the dictionary...
Word Count: 14 (processed: 18)
~5 seconds
✓ 4 suggestion(s)
✓ 8828 Words loaded!
SpellCheckDoc developed by Hossein Shahsahebi for NLP project at Asia Pacific University.

Figure 5: GUI schematic after providing a text and detecting all possible errors that can be caught by the application


Cloud computing is shown in the WEB aplication . We tried to show all possibilities.

☹ No suggestion available

Suggestions

WEB - check the case

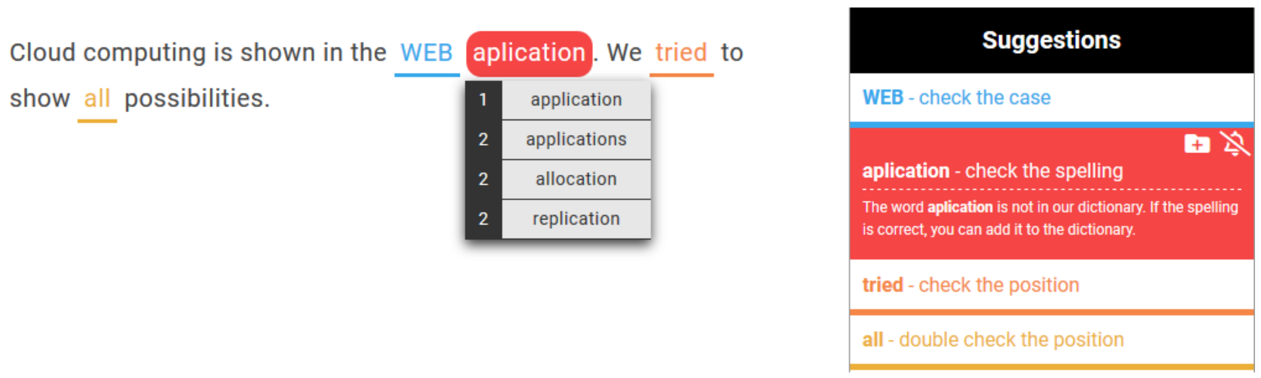
The word **WEB** may have a case error. Consider reviewing the word

aplication - check the spelling

tried - check the position

all - double check the position

Figure 6: Suggestion details for the Case error


Cloud computing is shown in the WEB aplication. We tried to show all possibilities.

1	application
2	applications
2	allocation
2	replication

Suggestions

WEB - check the case

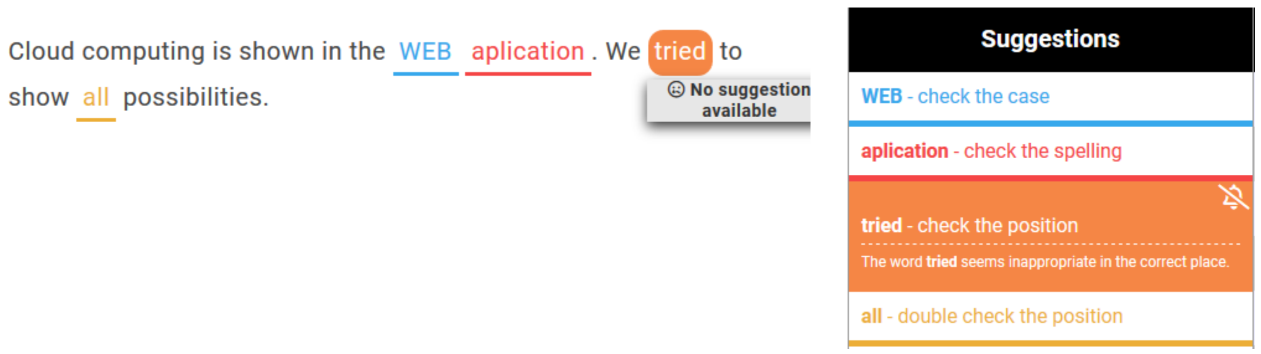
aplication - check the spelling

The word **aplication** is not in our dictionary. If the spelling is correct, you can add it to the dictionary.

tried - check the position

all - double check the position

Figure 7: Suggestion details for the Non-Word error


Cloud computing is shown in the WEB aplication . We tried to show all possibilities.

☹ No suggestion available

Suggestions

WEB - check the case

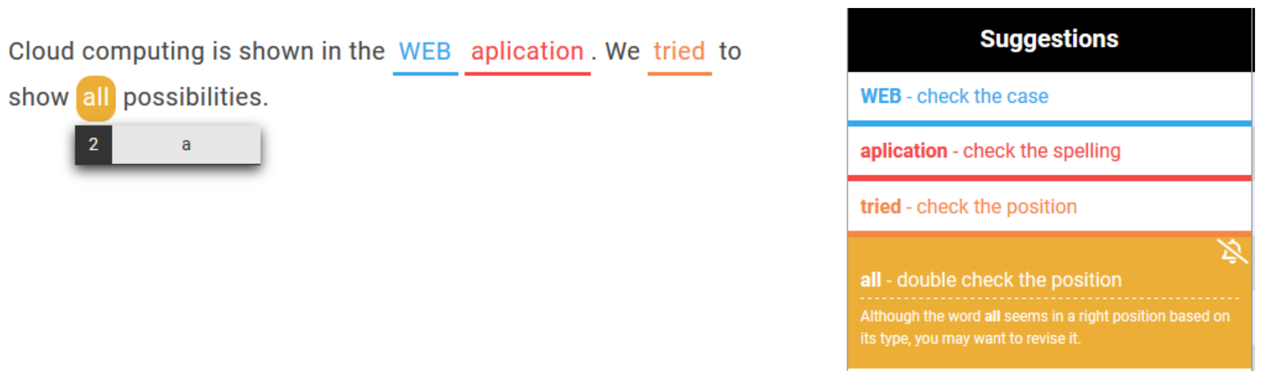
aplication - check the spelling

tried - check the position

The word **tried** seems inappropriate in the correct place.

all - double check the position

Figure 8: Suggestion details for Real-Word error


Cloud computing is shown in the WEB aplication . We tried to show all possibilities.

2	a

Suggestions

WEB - check the case

aplication - check the spelling

tried - check the position

all - double check the position

Although the word **all** seems in a right position based on its type, you may want to revise it.

Figure 9: Suggestion details for the RW-Ignored error

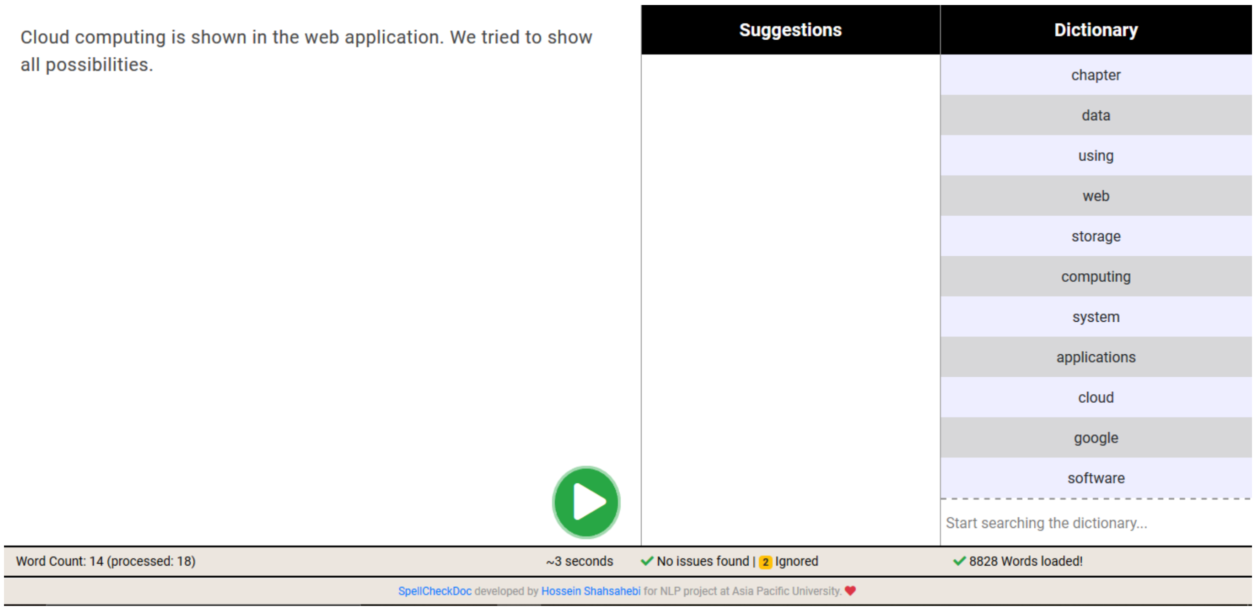
Figure 6 shows the suggestion details for the *Case* error where the case of the word does not match to our corpus. In this figure, “WEB” should not be written in uppercase letters based on the corpus representations of this word. This type of error is demonstrated in blue color and details are provided in the suggestion section. At this version, application does not provide any suggestion for *Case* error instances. Also, for this type of error, user is able to ignore the notification in order to remove the highlights from the desired word.

Figure 7 illustrates the notification for a *Non-Word* error in a red color. This means that the word is either misspelled or does not exist in the dictionary. User has three options to deal with this *Non-Word* error. Ignoring the word removes the word highlighting and its suggestion section box and will ignore all future “aplication” that come immediately after the word “web”. Adding the word to dictionary will remove the highlighting and the suggestion section box and will detect the word “aplication” as a correct word permanently. Choosing a suggested word from the dropdown list will remove highlighting and its suggestion section box and it will replace the word “aplication” with the selected word from the dropdown list. Currently, only Non-Word errors have the ability of being added to the dictionary. Adding the word to the dictionary will also adds the bigram rule to the bigram database based on the preceding word.

Figure 8 show a *Real-Word* error as there is no “tries” word in the corpus that comes after the word “we”. The only option for the user in this case is ignoring the notification in addition to changing the word manually. Also, no suggestion is provided for this error, although words such as “try” and “trying” already exist in the dictionary. This is another delightful feature of the system. The reason that those two mentioned words are not provided as a suggestion is preventing another *Real-Word* problem. In other words, bigrams like “we try” and “we trying” do not exist in our bigram database either and if user replace the word with these two words and re-run the spell checker, it will produce the same error again.

Figure 9 denotes the last type of error which is *RW-Ignored* error. This error represents that although there is no bigram like “show all” in our bigram database, there is a rule that “Determiner” position can come after “Verb” position. Therefore, the error message for this error focus on the double check instead of simply telling user to check the word. Also, the suggestions that we are providing for words with this error, should not produce a *Real-Word* error or reproduce a *RW-Ignored* error. So, suggested words must have the previous word in their previous words in the bigram database. In other word, in this image, the bigram “show a” resides in the bigram database and that is why this suggestion is provided and other words are not.

After replacing the “aplication” with the suggested “application”, ignoring “tried” and “all” words, and changing the word “WEB” to “web” manually, Figure 10 depicts the GUI. As it shows, no more highlights and suggestion boxes will be shown to the user. Also, the status bar mentions that the SpellCheckDoc did not find any issues with the text after ignoring two rules that were ignored by the user earlier.


Cloud computing is shown in the web application. We tried to show all possibilities.

Word Count: 14 (processed: 18) ~3 seconds ✓ No issues found | 2 Ignored ✓ 8828 Words loaded!

SpellCheckDoc developed by Hossein Shahsahebi for NLP project at Asia Pacific University.

Figure 10: GUI schematic after fixing and ignoring the reported errors.

Moreover, as shown in Figure 5 and Figure 10 the processing time of the same text reduced from five seconds to three seconds. The system has the ability to reduce the processing time when user checks the same text for several times as it will cache some fetched information and does not require to re-fetch repetitive information.

5.2 Results

The methodology and implementation of the validation process had been already discussed in-depth in Section **Error! Reference source not found.**. In this section, the result of performing the mentioned process will be discussed and the output of model evaluation will be presented.

5.2.1 Non-Word Errors Validation

We repeated the validation process on three text derived from the main corpus. Two of these instances returned the exact number of errors and one of them returned more errors than expected. The reason for returning more than expected errors was the presence of symbols “ and ”, where the preprocessing module was unable to catch and it was retrieved as Unicode and checked against the dictionary. Table 3 shows the details about the validation.

		SpellCheckDoc Non-Word Analysis			
		Non-Word	Correct Words	Precision	Recall
Text 1[13]	Non-Words	10	0	1	1
	Correct Words	0	84		
Text 2[14]	Non-Words	8	0	1	0.67
	Correct Words	4	64		
Text 3[15]	Non-Words	10	0	1	1
	Correct Words	0	78		
Average				1	0.89

Table 3: Validation process to evaluate Non-Word detection

[13] The text has 94 words and can be found in Section 7.2.1
[14] The text has 76 words and can be found in Section 7.2.2
[15] The text has 88 words and can be found in Section 7.2.3

5.2.2 Real-Word Errors Validation

Similarly, we add some real-word errors to a file containing one or multiple lines of the main cleaned corpus. The validator will check the code to detect possible errors. Detections and expectation are compared and Table 4 shows the details about the findings.

		SpellCheckDoc Real-Word Analysis			
		Real-Word	Correct Words	Precision	Recall
Text 1[16]	Real-Words	13	0	1	1
	Correct Words	0	70		
Text 2[17]	Real-Words	12	1	0.92	1
	Correct Words	0	69		
Text 3[18]	Real-Words	12	0	1	1
	Correct Words	0	82		
Average				0.97	1

Table 4: Validation process to evaluate Real-Word detection

The reason for the issue in the second text is also related to the cleaning process, where a word between "you" and "your" was removed because of its pattern.

5.2.3 Combination of Real-Word and Non-Word Errors Validation

Similarly, in this section we impute combinational errors to each text and run the validator to check the findings and compare it to the bigram and dictionary databases and the main corpus.

Table 5 shows the details resulted from validating the combined error imputation.

[16] The text has 83 words and can be found in Section 7.2.4
[17] The text has 82 words and can be found in Section 7.2.5
[18] The text has 94 words and can be found in Section 7.2.6

		SpellCheckDoc Combination Analysis			
		Error	Correct Words	Precision	Recall
Text 1[19]	Error	17	0	1	1
	Correct Words	0	79		
Text 2[20]	Error	21	0	1	1
	Correct Words	0	53		
Text 3[21]	Error	19	1	1	0.95
	Correct Words	0	58		
Average				1	0.98

Table 5: Validation process to evaluate both Real-Word and Non-Word detection

5.2.4 Final Evaluation

As we discussed earlier, these validations are resulted from imputing errors manually to the text that are fetched from the primary corpus, which is already processed and stored in the database. Therefore, the model is highly overfitting and the result can be drastically decreased if any unseen word or bigram provided to the core. The possible solutions to deal with these issues are listed in the next section. However, based on the previous evaluation, the final evaluation of the system based on the discussed validation approach is provided in the Table 6.

Functionality	Precision	Recall	F1-Score * 100
No-Word Error	1	0.89	94.17%
Real-Word Error	0.97	1	98.47%
Combination Error	1	0.98	98.99%
Average	0.99	0.95	96.95%

Table 6: Final evaluation of the system

[19] The text has 96 words and can be Section 7.2.7
[20] The text has 74 words and can be found in Section 7.2.8
[21] The text has 78 words and can be found in Section 7.2.9

5.3 Shortcomings

This project is still in its alpha version and it needs more tuning. The following issues need to be address in the future versions:

- Enhance the cleaning and preprocessing modules to handle more patterns and exceptions.
- Integrate N-grams where N > 2 to empower the rule-based checking
- Apply a decision tree and other machine learning algorithm to enable the dictionary to learn from the patterns and process rules with a bigger insight.
- Prevent ignoring punctuation to be able to suggest a proper place when needed
- Assess the system to decrease the memory usage
- Provide graphical suggestions and errors for customized words in the GUI

6 References

Bird, S., Klein, E. and Loper, E. (2009) *Natural language processing with Python*. O'Reilly. Available at: https://books.google.com.my/books/about/Natural_Language_Processing_with_Python.html?id=KGIbfiiP1i4C&source=kp_book_description&redir_esc=y (Accessed: 29 October 2019).

Friedl, J. E. F. (2006) *Mastering regular expressions*. O'Reilly. Available at: https://www.oreilly.com/library/view/mastering-regular-expressions/0596528124/ (Accessed: 29 October 2019).

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. Available at: www.ijarcsse.com.

LemmInflect (2019) *LemmInflect: A python module for English lemmatization and inflection.* Available at: https://github.com/bjascob/LemmInflect (Accessed: 30 October 2019).

Machine Learning Plus (2019) *Lemmatization Approaches with Examples in Python*. Available at: https://www.machinelearningplus.com/nlp/lemmatization-examples-python/ (Accessed: 29 October 2019).

Mandal, P. and Hossain, B. M. M. (2017) 'Clustering-based Bangla spell checker', in *2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. IEEE, pp. 1–6. doi: 10.1109/ICIVPR.2017.7890878.

Nayak, A. S. and Kanive, A. P. (2016) 'Survey on Pre-Processing Techniques for Text Mining', *International Journal Of Engineering And Computer Science*, 5(6), pp. 16875–16879. doi: 10.18535/ijecs/v5i6.25.

Nejja, M. and Yousfi, A. (2015) 'The Context in Automatic Spell Correction', *Procedia Computer Science*. Elsevier Masson SAS, 73(Awict), pp. 109–114. doi: 10.1016/j.procs.2015.12.055.

Shaikh, R. (2018) *Gentle Start to Natural Language Processing using Python*, *twoardsdatascience*. Available at: https://towardsdatascience.com/gentle-start-to-natural-language-processing-using-python-6e46c07addf3 (Accessed: 29 October 2019).

Singh, S. P. *et al.* (2016) ‘Frequency based spell checking and rule based grammar checking’, in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. IEEE, pp. 4435–4439. doi: 10.1109/ICEEOT.2016.7755557.

Sosinsky, B. (2010) *Cloud Computing Bible*. Indianapolis, IN, USA: Wiley Publishing, Inc. doi: 10.1002/9781118255674.

Stackoverflow (2014) *Extracting text from a PDF file using PDFMiner in python? - Stack Overflow*. Available at: https://stackoverflow.com/questions/26494211/extracting-text-from-a-pdf-file-using-pdfminer-in-python (Accessed: 29 October 2019).

Vijayarani, S. *et al.* (2015) ‘Preprocessing Techniques for Text Mining -An Overview’, *International Journal of Computer Science & Communication Networks*, 5(1), pp. 7–16.

7 Appendix

7.1 Online Material access

As the whole written codes have a high volume, you can use this link https://github.com/hshahsahebi/SpellCheckDoc to have an access to the Github repository and explore the code with a better UI.

To have access to the recorded video of the functionality of the project, you can visit this link. Also, for further information about projects updates refer to the Github page.

7.2 Validation Texts

The following texts are used as a validation input text which certainly contains errors.

7.2.1 Non-Word Errors Validation Text 1

Some modelng technologies allow for the reduction of the model to executeble code. Many core Web servies are loked out of the iPhone platform. Note SNIA (Storage Networking Industry Asociation; http://www.snia.org/) has coined the term Data Storage as a Service (DaaS) to describe the delivery of storage on demand to clients over a distributed system. Other fetures of CDMI are access kontrols, usage accounting, and the ability to advertise containers so that applications see these containers as if they are volumes (LUNs with a certain size). The consept of presense as it relates to identiti is also introduced.

7.2.2 Non-Word Errors Validation Text 2

FIGURE 11.4 is powerful and particularly easy to use. People discount future risk and favor instant gratification. A fature fone ik mor capeble dan ap “damb pon,” kat often it has a screen that is limited to text or very low-end graphics. This means your users always have access to the latest software versions. In Chapter 17, “Using Webail Services,” you learn about perhaps the most popular category of cloud-based applications in use and e-mail outside of simple Web browsing.

7.2.3 Non-Word Errors Validation Text 3

Po learn more about these other programst, I refer you to survey books that cover these types of programs from a functional category basisk. Phe latter three services are described in this chapteri. PREST stands for Representational State Transferf, and it owes its original description to the work of Roy Fielding, who was also a co-developer of the HTTP protocol. Create and share content with Google Earth, Maps, and Maps for mobilep. Citrix GoToMeeting (http://www.gotomeeting.com/) is one of the best-known collaboration software products because the company heavily advertises the service. PIGURE 17.7 Yahooo!

7.2.4 Real-Word Errors Validation Text 1

second The and much larger deployment moves titles its and the ability to use the Watch Instantly online service to AWS. sosonline backup.com/ spideroak. cloud computing is a Additionally, stateless system, as is the Internet in general. OpenID is supported by the OpenID Foundation openid.net/foundation/), a organization that promotes the technology. Available wherever books sold are. In message the descriptions, the message types are declared. application Any or process that benefits from economies of scale, commoditization of, and conformance to programming standards benefits from the application of cloud computing.

7.2.5 Real-Word Errors Validation Text 2

Also described in this chapter are syndication services. RIM BlackBerry a has leadership position in the messaging a very large market share. You a see page on which you your name account, provide passworda , select a payment option and .An AMI can be for public use under distribution a free license, for with operating systems such as Windows, or shared by an user with other users who are given the privilege of access.

7.2.6 Real-Word Errors Validation Text 3

cloud In computing, anyone be can a giant any at time. This type of graph you allows to add the appropriate server type your to infrastructure while performing a cost analysis of the deployment. An architectural layer containing ADCs is described as an Application Delivery Network (ADN), and is considered to provide WAN optimization services. The in settlement June 2010 requires Twitter to add more layers of security and undergo audits biannually. Chapter 5: Understanding Abstraction and Virtualization Note Not all CPUs support machines virtual, and many that do require that you enable this support in the BIOS.

7.2.7 Combinational Errors Validation Text 1

This is analogus to the goals to used create the Amazon Simple Storage System The servic is meant two be low touch, in that it abstracts many of the common concerns of database administrators for hardware requirements, software maintenance, indexing, and performance optimization. Microsoft Exchange and Windows Mobile use a technology calld ActiveSync (http://www.microsoft.com/ that has gone through several development cycles. It is a standard of OASIS and an XML standard for passing autentication and authorization between an identity provider and the service provider. Twitter provides a unique way of accesing the knowledge of a community in real timing.

7.2.8 Combinational Errors Validation in Text 2

The Droopal is cores the standrad distiribution, with current the version being 6.19; version 7.0 is in preview. Connect client your or network to Talk the Google network, add chatback, or customize the Google Talk gadget. EBS tends to be used in transactianal systems where high-speed data access is required. Figure 20.8 shows the Windows Phone 7 Web site. When this modle is applied to cloud computing, person tend to be overwhelmed by the choice and delay adoption.

7.2.9 Combinational Errors Validation in Text 3

Windos Azore Platfrom alows developres toop stoge thier applications on top of so infrastructure that any application built with the Framework can live locally, in cloud network, or some combination thereof. filefront. It has whatever limit the ISV has placed on it's software. The biggest difference is that because SLS Azure managed in the cloud, there are no administrative controls over the SQL engine. System metrics Notice that in the previus you section, determined what amounts to application-level statistics for your site.

7.3 PDF To Text Code

```python
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
from io import StringIO

def convert_pdf_to_txt(path):
    rsrcmgr = PDFResourceManager()
    retstr = StringIO()
    codec = 'utf-8'
    laparams = LAParams()
    device = TextConverter(rsrcmgr, retstr, codec=codec, laparams=laparams)
    fp = open(path, 'rb')
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    password = ""
    maxpages = 0
    caching = True
    pagenos=set()

    for page in PDFPage.get_pages(fp, pagenos, maxpages=maxpages, password=passwo
rd,caching=caching, check_extractable=True):
        interpreter.process_page(page)

    text = retstr.getvalue()

    fp.close()
    device.close()
    retstr.close()

    fw = open('resources/corpus.txt', 'w+', encoding='utf-8')
    fw.write(text)
    fw.close()
```

7.4 Initiator Module Code

```
import os
import sys
sys.path.insert(0, "../code_repo")
from pdf_to_text import convert_pdf_to_txt as p2t
from corpus_cleaning import Cleaner
from dictionary_creator import create_dictionary, create_bigram

'''
Look in resource folder for corpus.txt
If it does not exist try to convert the pdf format of corpus to text format
'''
if(not os.path.exists('resources/corpus.txt')):
    p2t('resources/corpus.pdf')

cleaner = Cleaner()

if(not os.path.exists('resources/corpus_cleansed.txt')):
    cleaner.clean_corpus(in_file='resources/corpus.txt', out_file='resources/corp
us_cleansed.txt')

cleaner.show_corpus_info(file_path='resources/corpus_cleansed.txt')
```

```python
if(not os.path.exists('resources/corpus_symbolic.txt')):
    cleaner.bigram_preparation(in_file='resources/corpus_cleansed.txt', out_file=
'resources/corpus_symbolic.txt')


if(create_dictionary()):
    print("Dictionary created successfully!")
else:
    print("Dictionary is already created!")

if(create_bigram()):
    print("Bigram created successfully!")
else:
    print("Bigram is already created!")
```

7.5 Validator Module Code

```python
import os
import sys
import random
import json
sys.path.insert(0, "../code_repo")
from lookup import Lookup

source_file = 'resources/corpus_cleansed.txt'
destination_validation = 'resources/validation/'

options_tuple = ('NW', 'RW', 'BO')
validations_set = []
maximum_limit = 100
lookup = Lookup()

def is_init():
    for counter in range (1, 4):
        for opt in options_tuple:
            if(not os.path.exists("{}{}_{}.txt".format(destination_validation, op
t, counter)) and
                not os.path.exists("{}{}_{}.ready.txt".format(destination_validat
ion, opt, counter))):
                return True
    return False


if(not os.path.exists(source_file)):
    print("Cleaned corpus does not exist")
    sys.exit()


if(is_init()):
    print("Initialization...")
    corpus = open(source_file, 'r', encoding='utf-8').read().split("\n")

    random_numbers = [i for i in range(0, len(corpus))]

    for counter in range(0, 3):
        for c2 in range(0, 3):
            valset = []
            word_numbers = 0

            while(word_numbers <= maximum_limit):
                corpus_index = random.choice(random_numbers)

                try:
                    if(word_numbers + len(corpus[corpus_index].split(" ")) > maxi
mum_limit):
                        if(word_numbers > (maximum_limit/2)):
                            break
                        else:
                            continue
                except IndexError:
                    random_numbers.remove(corpus_index)
                    continue

                word_numbers += len(corpus[corpus_index].split(" "))
                valset.append(corpus[corpus_index])
                corpus.pop(corpus_index)
                random_numbers.remove(corpus_index)

            if(c2 == 0):
                validations_set.append(["\n".join(valset)])
            else:
                validations_set[counter].append("\n".join(valset))

    for counter in range (0, 3):
        for c2 in range(0, 3):
            print("Creating file for {}{}_{}.txt ...".format(destination_validati
on, options_tuple[counter], c2 + 1))
            fw = open("{}{}_{}.txt".format(destination_validation, options_tuple[
counter], c2 + 1), 'w+', encoding='utf-8')
```

```
            fw.write(validations_set[counter][c2])
            fw.close()


for counter in range (1, 4):
    for opt in options_tuple:
        file_s = "{}{}_{}.ready.txt".format(destination_validation, opt, counter)
        file_d = "{}{}_{}.result.txt".format(destination_validation, opt, counter
)

        if(not os.path.exists(file_d) and os.path.exists(file_s)):
            print("Spell Check in Progress for {} ...".format(file_s))
            lines = lookup.load_raw_text(file_path=file_s)
            corrections = []
            corrected_words = []

            for line_of_words in lines:
                prev_word = (None, None)

                for word in line_of_words:
                    word_result = lookup.validate_word(word, prev_word)
                    if(opt == 'NW' and word_result['status'] == lookup.CODE_NON_W
ORD_ERR):
                        corrections.append(word_result)
                        corrected_words.append(word_result['word'][0])
                        print("NW Correction added...")
                    elif(opt == 'RW' and word_result['status'] in [lookup.CODE_RE
AL_WORD_ERR,
                        lookup.CODE_RWE_IGNORED]):
                        corrections.append(word_result)
                        corrected_words.append(word_result['word'][0])
                        print("RW Correction added...")
                    elif(opt == 'BO' and word_result['status'] not in [lookup.COD
E_CASE_ERR,
                        lookup.CODE_CORRECT]):
                        corrections.append(word_result)
                        corrected_words.append(word_result['word'][0])
                        print("RW/NW Correction added...")

                    prev_word = word

            fw = open(file_d, 'w+', encoding='utf-8')
            fw.write("This file includes {} records.\n".format(len(corrections)))
            fw.write("Corrected words are as follow:\n")
            fw.write(json.dumps(corrected_words, sort_keys=False, indent=4))
            fw.write(json.dumps(corrections, sort_keys=False, indent=4))
            fw.close()
            print("File {} created.".format(file_d))
```

7.6 Corpus Cleaning Module Code

```
from nltk import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
import string
from collections import Counter
import re
from preprocessing import Preprocessing
from dictionary import Dictionary
from os import path

class Cleaner:

    def __init__(self):
        self.preprocessing = Preprocessing()

    def show_corpus_info(self, file_path = None, passage = None, store = True):

        if(file_path != None and path.exists(file_path)):
            corpus = open(file_path, 'r', encoding='utf-8').read()
        elif(passage != None):
            corpus = passage
        else:
            return False

        corpus = self.preprocessing.replace_urls(corpus, replace_with='')

        all_words = [word.lower() for word in word_tokenize(corpus) if len(word)
> 0]
        unique_words = set(all_words)

        stop_words = stopwords.words("english") + list(string.punctuation + '®')

        all_processed_words = [word for word in all_words if word not in stop_wor
ds]
        unique_processed_words = [word for word in unique_words if word not in st
op_words]

        words_counter = Counter(all_processed_words)
        most_common = words_counter.most_common(20)
        print("Number of words in corpus: {}".format(len(all_words)))
        print("Number of unique words is: {}".format(len(unique_words)))
        print("Number of non stop words in corpus: {}".format(len(all_processed_w
ords)))
        print("Number of unique non stop words is: {}".format(len(unique_processe
d_words)))
        print("The most 30 common words are:\n{}".format(most_common))

        if(store):
            dic_handle = Dictionary()
            dic_handle.store_common_words(most_common)

    def clean_corpus(self, in_file = None, out_file = None, passage = None):

        if(in_file != None and path.exists(in_file)):
            corpus = open(in_file, 'r', encoding='utf-8').read().split("\n")
        elif(passage != None):
            corpus = passage.split("\n")
        else:
            return False

        corpus = self.remove_meaningless_lines(corpus)

        corpus = self.attach_incomplete_lines(' '.join(corpus))
        output = "\n".join(corpus).strip()

        if(out_file != None):
            fw = open(out_file, 'w+', encoding='utf-8')
            fw.write(output)
            fw.close()
            return True
        else:
            return output

    def bigram_preparation(self, in_file = None, out_file = None, passage = None)
:

        if(in_file != None and path.exists(in_file)):
            corpus = open(in_file, 'r', encoding='utf-8').read()
        elif(passage != None):
            corpus = passage
        else:
            return False
        new_corpus = self.preprocessing.impute_bigram_symbols(corpus).strip()

        if(out_file != None):
            fw = open(out_file, 'w+', encoding='utf-8')
            fw.write(new_corpus.strip())
            fw.close()
            return True
        else:
            return new_corpus

    '''
    @input: Array of lines in the corpus
    @output: Array of filtere lines in the corpus
    @description: The function removes empty lines and lines with lower length of
 5
    after removing meaningless/inappropriate format words
    '''
    def remove_meaningless_lines(self, corpus):
        new_corpus = []

        for line in corpus:
            # Split the line into words
            words = self.preprocessing.customized_word_tokenizer(line)

            # Check each word to match a real word, number, date, time and remove
 it if not
            for wk, word in enumerate(words):
                if(not self.preprocessing.is_eligible_word(word)):
                    del words[wk]
                elif(wk == 0 and word == 'l'):
                    del words[wk]
                elif(re.match(r"www.it-ebooks.info", word)):
                    del words[wk]


            # Replace the new filtered line
            # Exclude lines if percentage of real words in the sentence is less t
han 40%
            # Exclude lines where the average length of words is less than 2
            if(len(words) > 0):
                real_words_length = 0
                all_length = 0

                for w in words:
                    all_length += len(w)
                    if(re.match(r"\b[A-Za-z’'-]+\b", w)):
                        real_words_length += len(w)

                if(all_length > 0 and (real_words_length / all_length) > 0.4 and
                    (real_words_length / len(words)) >= 2):
                    new_corpus.append(' '.join(words).strip())

        return new_corpus


    '''
    @input: Textual corpus after removing invalid characters and phrases
    @output: Array of sentences
    @description: Using sentence tokenizer to append multiline sentences together
 and split each sentence
    '''
    def attach_incomplete_lines(self, corpus):
        corpus = re.sub(r"(\-\s)(?!$)", '', corpus)

        sentences = [re.sub(r"(\n+)|(\s+)", ' ', sentence.strip()) for sentence i
n sent_tokenize(corpus)
            if len(re.sub(r"(\n+)|(\s+)", ' ', sentence.strip())) > 5]

        return sentences
```

7.7 Preprocessing Module Code

```
import re
from nltk import word_tokenize, sent_tokenize, pos_tag
import string
import redis
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet


class Preprocessing:

    def __init__(self):
        self.customized_symbols = {
            'date': '|DATE|',
            'time': '|TIME|',
            'url': '|URL|',
            'number': '|NUMBER|',
            'sent_start': '|STARTSENT|',
            'sent_end': '|ENDSENT|'
        }

        self.lemmatizer = WordNetLemmatizer()

    '''
    @input: String representing a sentence or group of words
    @output: Array representing all words in the input string
    '''
    def customized_word_tokenizer(self, phrase):
        singular_tokens = ['.', '?', '!', ',', ';', '-', '_']
        removal_tokens = ['(', ')', '"', "'", '[', ']']

        phrase = re.sub(r"\s+", ' ', phrase.strip())

        for t in singular_tokens:
            patt = '([\\' + t + ']\\s)+'
            phrase = re.sub(patt, t + ' ', phrase)
            patt = '(\\s[\\' + t + '])+'
            phrase = re.sub(patt, ' ' + t, phrase)
            patt = '[\\' + t + ']+'
            phrase = re.sub(patt, t, phrase)

        for t in removal_tokens:
            patt = '[\\' + t + ']+'
            phrase = re.sub(patt, t, phrase)

        phrase = re.sub(r"\s+", ' ', phrase.strip())

        return phrase.split(' ')

    '''
    @input: String representing a word
    @output: Boolean representing whether the string can be considered as a word
or not
    @description: The string shoud be numeric or alphabetic to be considered as w
ord.
    Some symbols that can be mixed with the numbers and alphabets are also consid
ered.
    '''
    def is_eligible_word(self, word):
        word = re.sub(r"^[\#\(\"\'\“]", '', word)
        word = re.sub(r"[\)\"\'\”\;\.\?\!\,]+$", '', word)
        if((re.match(r"^[A-Za-z]+[\-\_]?[A-Za-z]*$", word) or re.match(r"^[0-
9\/\:\-]+$", word) or
            re.match(r"^[A-Za-z]+[0-9]{1, 3}[A-Za-z]*$", word) or
            re.match(r"(\w+(\:\/\/))?([\w\-]+\.)?[\w\-]+\.[\w\-]+[\/\w\+-
\?\{\}\(\)\[\]\&\^\$\#\@\~]*", word))
            and re.match(r"[A-Za-z0-9]", word)):
            return True

        return False

    '''
    @input: Raw text without any bigram symbol
    @output: Text with added required symbols
    @description:
    Step 1: Replace all representations of dates with |DATE|
    Step 2: Replace all representations of times with |TIME|
    Step 3: Replace all representations of links with |URL|
    Step 4: Replace all representations of numbers with |NUMBER|
    Step 5: Add |START_SENT| to the beginning of each sentence
    Step 6: Add |END_SENT| to the end of each sentence
    '''
    def impute_bigram_symbols(self, text):
        text = self.replace_dates(text)
        text = self.replace_times(text)
        text = self.replace_urls(text)
        text = self.replace_numbers(text)

        sentences = [self.customized_symbols['sent_start'] + " " +
            re.sub(r"(\n+)|(\s+)", ' ', sentence.strip()) + " " + self.customized
_symbols['sent_end']
            for sentence in sent_tokenize(text) if len(re.sub(r"(\n+)|(\s+)", ' '
, sentence.strip())) > 5]

        return "\n".join(sentences)

    def replace_dates(self, text, replace_with = None):
        text = re.sub(r"\b\d{1,2}(\/|\-|\_|\s)\d{1,2}(\/|\-
|\_|\s)(\d{4}|\d{2})\b",
            self.customized_symbols['date'], text)
        text = re.sub(r"\d{1,2}(\s*)(jan|feb|mar|apr|may|jun|jul|aug|oct|nov|dec|
january|february|march|april|may|june|july|august|september|october|november|dece
mber)(\s*)(\d{4}|\d{2})",
            self.customized_symbols['date'], text, flags=re.IGNORECASE)
        text = re.sub(r"(jan|feb|mar|apr|may|jun|jul|aug|oct|nov|dec|january|febr
uary|march|april|may|june|july|august|september|october|november|december)(\s|\s?
\,\s?)?\d{1,2}(st|nd|rd|th)?(\s|\s?\,\s?)?(\d{4}|\d{2})",
            self.customized_symbols['date'], text, flags=re.IGNORECASE)

        return text

    def replace_times(self, text, replace_with = None):
        replace_with = self.customized_symbols['time'] if replace_with == None el
se replace_with

        text = re.sub(r"\d{1,2}(\s*(am|pm)|(\:|\-)\d{2}(\s*(am|pm)|(\:|\-
)\d{2}(\s*(am|pm))?))",
            replace_with, text, flags=re.IGNORECASE)

        return text

    def replace_urls(self, text, replace_with = None):
        replace_with = self.customized_symbols['url'] if replace_with == None els
e replace_with

        text = re.sub(r"(\w+(\:\/\/))?([\w\-]+\.)?[\w\-]+\.[\w\-]+[\/\w\+-
\?\{\}\(\)\[\]\&\^\$\#\@\~]*",
            replace_with, text, flags=re.IGNORECASE)

        return text

    def replace_numbers(self, text, replace_with = None):
        replace_with = self.customized_symbols['number'] if replace_with == None
else replace_with

        text = re.sub(r"(\d+(\.\d+|[,0-9]+(\.\d+)?)?)|(\.\d+)",
            replace_with, text, flags=re.IGNORECASE)

        return text

    def is_customized_word(self, word):
        return word in self.customized_symbols.values()

    def fetch_line_words(self, line, escape_puncs = True, escape_symbols = True):
        separators = "\\".join([char for char in string.punctuation])
        separators = "^[\\" + separators + "]$"

        initial_words = word_tokenize(line)
```

```python
        return pos_tag([word for word in initial_words if
            (not escape_puncs or not re.match(separators, word)) and
            (not escape_symbols or not self.is_customized_word(word))])

    def fetch_lemmatized_word(self, word, pos=''):
        wordnet_pos = self.get_wordnet_pos(pos.lower())

        if(wordnet_pos):
            root = self.lemmatizer.lemmatize(word.lower(), pos=wordnet_pos)
        else:
            root = self.lemmatizer.lemmatize(word.lower())

        return root

    def get_wordnet_pos(self, treebank_tag):
        if treebank_tag.startswith('j'):
            return wordnet.ADJ
        elif treebank_tag.startswith('v'):
            return wordnet.VERB
        elif treebank_tag.startswith('n'):
            return wordnet.NOUN
        elif treebank_tag.startswith('r'):
            return wordnet.ADV
        else:
            return ''
```

7.8 Dictionary Creator Module Code

```python
from preprocessing import Preprocessing
from dictionary import Dictionary
import re
import os

def create_dictionary(in_file = None, passage = None):
    if(in_file != None and os.path.exists(in_file)):
        corpus_lines = open(in_file, 'r', encoding='utf-8').read().split("\n")
    elif(passage != None):
        corpus_lines = passage.split("\n")
    else:
        print("Invalid input!")
        return

    d = Dictionary()
    p = Preprocessing()

    if(d.database_exists(d.DB_DICTIONARY)):
        return False


    for line in corpus_lines:
        words = p.fetch_line_words(line)

        for word in words:
            main_word = re.sub(r"[^-A-Za-z0-9]", '', word[0])
            root = p.fetch_lemmatized_word(main_word, word[1])
            d.prepare_word2dic(main_word, root)

    return d.store_prepared_data()

def create_bigram(in_file = None, passage = None):
    if(in_file != None and os.path.exists(in_file)):
        corpus_lines = open(in_file, 'r', encoding='utf-8').read().split("\n")
    elif(passage != None):
        corpus_lines = passage.split("\n")
    else:
        print("Invalid input!")
        return

    d = Dictionary()
    p = Preprocessing()

    if(d.database_exists(d.DB_BIGRAM)):
        return False


    for line in corpus_lines:
        words = p.fetch_line_words(line, escape_symbols=False)
        prev_word = (None, None)

        for word in words:
            d.prepare_bigram2dic(word, prev_word)
            prev_word = word

    return d.store_prepared_data()
```

7.9 Dictionary Module Code

```python
import redis
import re
from preprocessing import Preprocessing

class Dictionary:

    def __init__(self):
        self.redis_handler = redis.Redis(db=1, decode_responses=True)
        self.preprocessing = Preprocessing()

        self.prepared_dic = dict()
        self.prepared_bigram = dict()
        self.prepared_lencat = dict()

        self.bigram_prefix = 'bigram_'
        self.lencat_prefix = 'lencat_'

        self.DB_DICTIONARY = 'dic_exists'
        self.DB_BIGRAM = 'bigram_exists'
        self.DB_LENCAT = 'lencat_exists'
        self.DB_COMMON = 'common_exists'

        self.CASE_UPPER = '1'
        self.CASE_LOWER = '2'
        self.CASE_BOTH = '0'

    def words_really_different(self, main_word, lemma_word):
        pattern = "^{}(es|s)?$".format(lemma_word.lower())
        try:
            if(not re.match(r"^[a-zA-Z]+$", main_word)):
                return False

            if(re.match(pattern, main_word.lower())):
                return False
        except re.error as e:
            print("{}: {}".format(pattern, main_word.lower()))
            raise Exception(str(e))

        return True

    def database_exists(self, keyword):
        return True if self.get_single_word_from_dic(keyword) == '1' else False
    def prepare_word2dic(self, main_word, root_word):
        word2store = root_word.lower()

        prev_case = self.prepared_dic[word2store] if word2store in self.prepared_
dic else None
        current_case = self.get_word_case(main_word, prev_case=prev_case)
        self.prepared_dic[word2store] = current_case

        if(self.words_really_different(main_word, root_word)):
            self.prepared_dic[main_word.lower()] = current_case

        self.prepare_lencat2dic(main_word, root_word)

    def prepare_lencat2dic(self, main_word, root_word):
        word1 = root_word.lower()
        word2 = main_word.lower()

        lencat_index = "{}{}".format(self.lencat_prefix, len(word1))
        if(lencat_index in self.prepared_lencat):
            self.prepared_lencat[lencat_index].add(word1)
        else:
            self.prepared_lencat[lencat_index] = {word1}

        if(word1 != word2):
            lencat_index = "{}{}".format(self.lencat_prefix, len(word2))
            if(lencat_index in self.prepared_lencat):
                self.prepared_lencat[lencat_index].add(word2)
            else:
                self.prepared_lencat[lencat_index] = {word2}

    def prepare_bigram2dic(self, word, prev_word):
        word2look = "{}{}".format(self.bigram_prefix, word[0].lower())
        word_pos = word[0] if self.preprocessing.is_customized_word(word[0]) else
 word[1]
        prev_w = None if prev_word[0] == None else prev_word[0]

        if(prev_word[0] == None):
            prev_p = None
        elif(self.preprocessing.is_customized_word(prev_word[0])):
            prev_p = prev_word[0]
        else:
            prev_p = prev_word[1]

        if(word2look in self.prepared_bigram):
            self.prepared_bigram[word2look]['pos'].add(word_pos.lower())
            self.prepared_bigram[word2look]['frequency'] += 1

            if(prev_w != None):
                self.prepared_bigram[word2look]['prev_words'].add(prev_w.lower())

            if(prev_p != None):
                self.prepared_bigram[word2look]['prev_pos'].add(prev_p.lower())
        else:
            self.prepared_bigram[word2look] = {
                'pos': {word_pos.lower()},
                'frequency': 1,
                'prev_words': set() if prev_w == None else {prev_w.lower()},
                'prev_pos': set() if prev_p == None else {prev_p.lower()}
            }

    def store_prepared_data(self):
        result = False
        set_dbs = set()

        if(len(self.prepared_dic) > 0):
            if(self.redis_handler.mset(self.prepared_dic)):
                result = True
                set_dbs.add(self.DB_DICTIONARY)
        if(len(self.prepared_bigram) > 0):
            with self.redis_handler.pipeline() as pipe:
                for word, data in self.prepared_bigram.items():
                    try:
                        pipe.set("{}_frequency".format(word), data['frequency'])
                        if(len(data['pos']) > 0):
                            pipe.sadd("{}_pos".format(word), *data['pos'])
                        if(len(data['prev_words']) > 0):
                            pipe.sadd("{}_prev_words".format(word), *data['prev_w
ords'])
                        if(len(data['prev_pos']) > 0):
                            pipe.sadd("{}_prev_pos".format(word), *data['prev_pos
'])
                    except TypeError as e:
                        print(str(e))
                        print("{}: {}".format(word, data))
                        return

                pipe_result = pipe.execute()

            if(not False in pipe_result):
                result = True
                set_dbs.add(self.DB_BIGRAM)
        if(len(self.prepared_lencat) > 0):

            with self.redis_handler.pipeline() as pipe:
                for index, words in self.prepared_lencat.items():
                    pipe.sadd(index, *words)

                pipe_result = pipe.execute()

            if(not False in pipe_result):
                result = True
                set_dbs.add(self.DB_LENCAT)
            else:
                print("{} => {}".format(self.DB_BIGRAM, pipe_result))

        if(result):
            with self.redis_handler.pipeline() as pipe:
                for db in set_dbs:
                    self.redis_handler.set(db, '1')

                pipe.execute()

        self.prepared_dic = dict()
        self.prepared_bigram = dict()
        self.prepared_lencat = dict()

        return result

    def add_single_word2dic(self, main_word, root_word):
        word2store = root_word.lower()
        value = self.get_single_word_from_dic(root_word)
        word_type = self.get_word_case(main_word, prev_case=value)

        with self.redis_handler.pipeline() as pipe:
            pipe.set(word2store, word_type)
            pipe.sadd("{}{}".format(self.lencat_prefix, len(word2store)), word2st
ore)
            if(self.words_really_different(main_word, root_word)):
                pipe.set(main_word.lower(), word_type)
                pipe.sadd("{}{}".format(self.lencat_prefix, len(main_word)), main
_word.lower())

            pipe.execute()

        return word_type
    def get_single_word_from_dic(self, word2look, bigram = False, postfix = None,
 type_set = False):
        word = word2look.lower() if not bigram else "{}{}_{}".format(self.bigram_
prefix, word2look.lower(), postfix)

        word = self.redis_handler.get(word) if not type_set else self.redis_handl
er.smembers(word)

        if(word != None and type(word) is not set):
            return word
        elif(word != None and type(word) is set and len(word) > 0):
            return set([term for term in word if term != None])

        return None

    def add_single_word2bigram(self, word, prev_word):
        word2look = "{}{}".format(self.bigram_prefix, word[0].lower())
        word_pos = word[0] if self.preprocessing.is_customized_word(word[0]) else
 word[1]
        prev_w = None if prev_word[0] == None else prev_word[0]

        if(prev_w == None):
            prev_p = None
        elif(self.preprocessing.is_customized_word(prev_w)):
            prev_p = prev_w
        else:
            prev_p = prev_word[1]

        with self.redis_handler.pipeline() as pipe:
            if(self.get_single_word_from_dic("{}_frequency".format(word2look)) !=
 None):
                pipe.incr("{}_frequency".format(word2look))
            else:
                pipe.set("{}_frequency".format(word2look), 1)

            pipe.sadd("{}_pos".format(word2look), *{word_pos.lower()})

            if(prev_w != None):
                pipe.sadd("{}_prev_words".format(word2look), *{prev_w.lower()})

            if(prev_p != None):
                pipe.sadd("{}_prev_pos".format(word2look), *{prev_p.lower()})

            print(word)
            pipe.execute()

    def get_single_word_from_bigram(self, word):
        frequency = self.get_single_word_from_dic(word, bigram=True, postfix='fre
quency')

        if(frequency != None):
            return {
                'pos': self.get_single_word_from_dic(word, bigram=True, postfix='
pos', type_set=True),
                'frequency': frequency,
                'prev_words': self.get_single_word_from_dic(word, bigram=True, po
stfix='prev_words', type_set=True),
                'prev_pos': self.get_single_word_from_dic(word, bigram=True, post
fix='prev_pos', type_set=True)
            }

        return None

    def get_word_case(self, word, prev_case = None):
        if(prev_case == None):
            return self.CASE_UPPER if word.isupper() else self.CASE_LOWER
        elif(prev_case == self.CASE_BOTH):
            return self.CASE_BOTH
        else:
            current_case = self.get_word_case(word)
            return self.CASE_BOTH if current_case != prev_case else current_case

    def get_words_by_length(self, length_list):
        pipe_result = []

        with self.redis_handler.pipeline() as pipe:
            for index in length_list:
                pipe.smembers("{}{}".format(self.lencat_prefix, index))

            pipe_result = pipe.execute()

        return pipe_result

    def store_common_words(self, words):
        if(not self.database_exists(self.DB_COMMON)):
            common_words = set()

            for word in words:
                for cw in word:
                    break
                common_words.add(cw)

            if(len(common_words) > 0 and self.redis_handler.sadd('common_words',
*common_words)):
                self.redis_handler.set(self.DB_COMMON, '1')
```

7.10 Lookup Module Code

```
import re
from preprocessing import Preprocessing
from dictionary import Dictionary
from corpus_cleaning import Cleaner
from suggestion import Suggestion

class Lookup:

    CODE_PENDING = 0
    CODE_CORRECT = 1
    CODE_NON_WORD_ERR = 2
    CODE_REAL_WORD_ERR = 3
    CODE_RWE_IGNORED = 4
    CODE_CASE_ERR = 5

    def __init__(self):
        self.preprocessing = Preprocessing()
        self.dictionary = Dictionary()
        self.cleaner = Cleaner()
        self.suggestion = Suggestion()

        self.word_val_in_dic = None
        self.word_val_in_bigram = None
        self.word_status_code = self.CODE_PENDING

        self.textual_status = {
            0: 'In_Progress',
            1: 'Correct Word',
            2: 'Non-Word Error',
            3: 'Real-Word Error',
            4: 'Real-Word Error Ignored',
            5: 'Case Error'
        }
    def load_raw_text(self, file_path = None, passage = None):
        if(file_path == None and passage == None):
            raise Exception('File path or passage should be provided. File path h
as priority!')

        text = self.cleaner.clean_corpus(in_file=file_path, passage=passage)

        if(text):
            text = self.cleaner.bigram_preparation(passage=text)

            return [self.preprocessing.fetch_line_words(line, escape_symbols=Fals
e) for line in text.split("\n")]

        return []

    def validate_word(self, word, prev_word):
        if(self.preprocessing.is_customized_word(word[0])):
            lemma = word[0]
        else:
            lemma = self.preprocessing.fetch_lemmatized_word(word[0], word[1])

        suggestions = dict()

        if(self.word_exists(word[0], lemma)):
            if(self.word_in_real_place(word, prev_word)):
                if(self.word_in_correct_case(word[0])):
                    self.word_status_code = self.CODE_CORRECT
                else:
                    self.word_status_code = self.CODE_CASE_ERR
            elif(self.word_can_be_real(word, prev_word)):
                self.word_status_code = self.CODE_RWE_IGNORED
            else:
                self.word_status_code = self.CODE_REAL_WORD_ERR
        else:
            self.word_status_code = self.CODE_NON_WORD_ERR

        if(self.word_status_code not in [self.CODE_CORRECT, self.CODE_CASE_ERR]):
            suggestions = self.suggestion.get_suggestions(word, prev_word, self.w
ord_status_code)
            if(word[0].lower() in suggestions):
                self.word_status_code = self.CODE_CORRECT

        return {
            'word': word,
            'prev_word': prev_word,
            'status': self.word_status_code,
            'textual_status': self.textual_status[self.word_status_code],
            'lemma': lemma,
            'suggestions': suggestions
        }

    def word_exists(self, main_word, lemmatized_word):
        if(self.preprocessing.is_customized_word(main_word)):
            self.word_val_in_dic = self.dictionary.CASE_BOTH
            return True

        self.word_val_in_dic = self.dictionary.get_single_word_from_dic(lemmatize
d_word)

        if(self.word_val_in_dic == None and self.dictionary.words_really_differen
t(main_word, lemmatized_word)):
            self.word_val_in_dic = self.dictionary.add_single_word2dic(main_word,
 lemmatized_word)

        return False if self.word_val_in_dic == None else True

    def word_in_real_place(self, word, prev_word):
        if(prev_word[0] == None):
            return True
        else:
            self.word_val_in_bigram = self.dictionary.get_single_word_from_bigram
(word[0])

            if(self.word_val_in_bigram != None and
                prev_word[0].lower() in self.word_val_in_bigram['prev_words']):
                return True

        return False

    def word_can_be_real(self, word, prev_word):
        prev_pos = prev_word[0] if self.preprocessing.is_customized_word(prev_wor
d[0]) else prev_word[1]

        if(self.word_val_in_bigram != None and prev_pos != None and
            self.word_val_in_bigram['pos'] != None and
            self.word_val_in_bigram['prev_pos'] != None and
            word[1].lower() in self.word_val_in_bigram['pos'] and
            prev_pos.lower() in self.word_val_in_bigram['prev_pos']):

            return True
        return False

    def word_in_correct_case(self, main_word):
        if(re.match(r"^([A-Z][a-z]+(\-\_)?)+$", main_word)):
            main_word = main_word.lower()

        if(re.match(r"^[A-Z]+(s|es)$", main_word)):
            main_word = re.sub(r"(s|es)$", '', main_word)
            new_val = self.dictionary.get_single_word_from_dic(main_word)
            if(new_val != None):
                self.word_val_in_dic = new_val

        if(len(main_word) > 1 and not main_word[1:].islower() and not main_word[1
:].isupper()):
            return False

        if(self.word_val_in_dic == None):
            return True

        current_case = self.dictionary.get_word_case(main_word)

        if(self.word_val_in_dic == self.dictionary.CASE_BOTH or current_case == s
elf.word_val_in_dic):
            return True

        return False
```

7.11 Suggestion Module Code

```
import re
import operator
from jellyfish import levenshtein_distance
from dictionary import Dictionary

class Suggestion:
    words_repo = dict()
    bigram_repo = dict()

    CODE_NON_WORD_ERR = 2
    CODE_REAL_WORD_ERR = 3
    CODE_RWE_IGNORED = 4
    CODE_CASE_ERR = 5
    def __init__(self):
        self.dictionary = Dictionary()
        self.active_word_repo = dict()

    def get_suggestions(self, word, prev_word, error_status, max_output = 5, max_
len_diff = 2):
        start_len = 1 if (len(word[0]) - max_len_diff < 2) else len(word[0]) - ma
x_len_diff
        end_len = len(word[0]) + max_len_diff + 1

        self.prepare_word_repo(start_len, end_len)
        self.select_lowest_distance(word[0].lower(), start_len, end_len)
        self.filter_active_repo(word, prev_word, error_status)
        output = self.select_best_suggestions()
        self.active_word_repo = dict()

        return output

    def prepare_word_repo(self, start_len, end_len):
        need_to_fetch = []

        for i in range(start_len, end_len):
            if(i not in self.words_repo):
                need_to_fetch.append(i)

        if(len(need_to_fetch) > 0):
            new_repo = self.dictionary.get_words_by_length(need_to_fetch)

            for words_set in new_repo:
                if(len(words_set) > 0):
                    for first_elem in words_set:
                        break

                    self.words_repo[len(first_elem)] = words_set

    def select_lowest_distance(self, main_word, start_len, end_len):
        for length in range(start_len, end_len):
            if length in self.words_repo:
                for word in self.words_repo[length]:
                    if(type(word) is str and re.match(r"[a-zA-
Z]", word) and main_word.lower() != word.lower()):
                        distance = levenshtein_distance(main_word, word)
                        if(distance < 3):
                            self.active_word_repo[word] = distance
    def filter_active_repo(self, current_word, prev_word, error_status):
        new_repo = dict()

        if(prev_word[0] == None):
            self.active_word_repo = new_repo
            return

        current_pos = current_word[1].lower()
        prev_pos = prev_word[1].lower()
        prev_word = prev_word[0].lower()

        for word in self.active_word_repo.keys():
            if word not in self.bigram_repo:
                self.bigram_repo[word] = self.dictionary.get_single_word_from_big
ram(word)

            if self.bigram_repo[word] != None and prev_word in self.bigram_repo[w
ord]['prev_words']:
                new_repo[word] = [self.active_word_repo[word], self.bigram_repo[w
ord]['frequency']]
            elif(self.bigram_repo[word] != None and error_status != self.CODE_RWE
_IGNORED and
                current_pos in self.bigram_repo[word]['pos'] and
                prev_pos in self.bigram_repo[word]['prev_pos']):
                new_repo[word] = [self.active_word_repo[word], 0]
            else:
                pass
                #Ignore the word if it doen't exist in previous words and positio
ns

                #new_repo[word] = [self.active_word_repo[word], 0]

        self.active_word_repo = new_repo

    def select_best_suggestions(self):
        try:
            suggestions = sorted(self.active_word_repo.items(), key=lambda kv:(kv
[1][0], 0-int(kv[1][1])))
        except TypeError:
            print('Sort Error: {}'.format(self.active_word_repo))

        result = list()

        for rec in suggestions:
            index = None
            sub_result = dict()
            for elem in rec:
                if(type(elem) is str):
                    sub_result[elem] = -1
                    index = elem
                elif(index != None):
                    sub_result[index] = elem[0]

            result.append(sub_result)

            if(len(result) == 5):
                break

        return result
```

7.12 GUI Processor Code

```
from flask import Flask, render_template, request, url_for, jsonify
import sys
sys.path.insert(0, '../code_repo')
from dictionary import Dictionary
from lookup import Lookup
from preprocessing import Preprocessing
import re
import time

app = Flask(__name__)

@app.route('/')
def homepage():
    return render_template('index.html.j2')

@app.route('/dictionary')
def dictionary():
    my_dic = Dictionary()
    out_words = dict()
    out_words['dictionary'] = []
    out_words['common_words'] = []

    db_words = my_dic.get_words_by_length([i for i in range(1, 101)])
    db_common = my_dic.get_single_word_from_dic('common_words', type_set=True)

    for words_set in db_words:
        for elem in words_set:
            if(re.match(r"[a-zA-Z]", elem)):
                out_words['dictionary'].append({'word': elem})

    for common in db_common:
        out_words['common_words'].append({'word': common})

    return jsonify(out_words)

@app.route('/spellcheck', methods=['POST'])
def spellcheck():
    handler = Lookup()

    corrections = []
    words_processed = 0
    text = request.form['text']
    start_time = 0
    end_time = 0

    if(type(text) is str and len(text.strip()) > 0):
        start_time = time.time()
        fw = open('log.txt', 'a+', encoding='utf-8')
        fw.write("Proccessing Strated: {}\n".format(start_time))

        lines = handler.load_raw_text(passage=text.strip())
        fw.write("Lines fetched: {}\n".format(lines))

        for line_of_words in lines:
            prev_word = (None, None)
            prev_status = None
            words_processed += len(line_of_words)

            for word in line_of_words:
                word_result = handler.validate_word(word, prev_word)

                fw.write("Validation Result: {}\n".format(word_result))

                if(word_result['status'] != handler.CODE_CORRECT):
                    #If previous word is non-word ignore real-word error
                    if(prev_status == handler.CODE_NON_WORD_ERR and
                        word_result['status'] in [handler.CODE_REAL_WORD_ERR, han
dler.CODE_RWE_IGNORED]):
                        fw.write("Included Status: Ignored (previous non-
word)\n")
                        pass
                    #If previous word has real-
word issue and also the current word has the same issue
                    #ignore it to prevent chain errors
                    elif(prev_status in [handler.CODE_REAL_WORD_ERR, handler.CODE
_RWE_IGNORED] and
                        word_result['status'] in [handler.CODE_REAL_WORD_ERR, han
dler.CODE_RWE_IGNORED]):
                        fw.write("Included Status: Ignored (previous real-
word/rwe)\n")
                        pass
                    else:
                        fw.write("Included Status: Included\n")
                        corrections.append(word_result)

                prev_word = word
                prev_status = word_result['status']


        end_time = time.time()
        fw.write("Proccessing Ended at {} in {} seconds\n".format(end_time, (end_
time - start_time)))
        fw.write("----------------------------------------\n\n")
        fw.close()

    output = {
        'processed_words': words_processed,
        'duration': int(end_time) - int(start_time),
        'corrections': corrections
    }

    return jsonify(output)

@app.route('/add_to_dic', methods=['POST'])
def add_to_dic():
    word = request.form['word']
    word_pos = request.form['word_pos']
    prev_word = request.form['prev_word']
    prev_word_pos = request.form['prev_word_pos']
    next_word = request.form['next_word']
    next_word_pos = request.form['next_word_pos']

    dic = Dictionary()
    pp = Preprocessing()

    root_word = pp.fetch_lemmatized_word(word, word_pos)
    dic.add_single_word2dic(word, root_word)
    dic.add_single_word2bigram((word, word_pos), (prev_word, prev_word_pos))
    dic.add_single_word2bigram((next_word, next_word_pos), (word, word_pos))

    return "success"
```

7.13 HTML Base Template

```
<!doctype html>
<html lang="en">
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">
        {% block meta %}{% endblock %}

        <link href="https://fonts.googleapis.com/css?family=Roboto:400,700&displa
y=swap" rel="stylesheet">
        <link rel="stylesheet" href="{{ url_for('static', filename='css/bootstrap
.min.css') }}">
        <link rel="stylesheet" href="{{ url_for('static', filename='css/fontaweso
me.min.css') }}">
        <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css
') }}">
        {% block style %}{% endblock %}

        <title>SpellCheckDoc {% block title %}{% endblock %}</title>
    </head>
    <body>
        <div class="container-fluid">
            {% block content %}{% endblock %}

            <div class="row status-bar">
                <div class="col-md-6 status-bar-text">Status bar...</div>
                <div class="col-md-3 status-bar-suggestion"></div>
                <div class="col-md-3 status-bar-dictionary"></div>
            </div>

            <footer>
                <a href="https://github.com/hshahsahebi/SpellCheckDoc" target="_b
lank">SpellCheckDoc</a> developed by
                <a href="https://www.linkedin.com/in/hshahsahebi/" target="_blank
">Hossein Shahsahebi</a>
```

```
                for NLP project at Asia Pacific University.
                <i class="fas fa-heart text-danger"></i>
            </footer>
        </div>

        <script type="text/javascript" src="{{ url_for('static', filename='js/jqu
ery.min.js') }}"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/
popper.min.js" integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="an
onymous"></script>
        <script type="text/javascript" src="{{ url_for('static', filename='js/boo
tstrap.min.js') }}"></script>
        <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/lib
s/fuse.js/3.4.5/fuse.min.js"></script>
        <script type="text/javascript" src="{{ url_for('static', filename='js/app
.js') }}"></script>
    </body>
</html>
```

7.14 HTML Index Page

```
{% extends "base.html.j2" %}

{% block content %}
<div class="row">
    <div class="col-xs-12 col-sm-9 col-md-6 pl-0">
        <div class="text-
input" spellcheck=false contenteditable="true" onblur="text_blur">Start typing...
</div>

        <button class="btn btn-success spellcheck-btn"><i class="fas fa-
play"></i></button>
    </div>

    <div class="col-xs-12 col-sm-3 col-md-3 pr-0">
        <div class="suggestions">
            <h1 class="section-header">Suggestions</h1>

            <div class="suggest-list text-center"></div>
        </div>
    </div>

    <div class="col-xs-12 col-sm-12 col-md-3 pr-0 pl-0">
```

```
        <div class="dictionary">
            <h1 class="section-header">Dictionary</h1>

            <div class="dictionary-word-list text-center"></div>

            <div class="dictionary-search-input">
                <input type="text" id="dic_search" placeholder="Start searching t
he dictionary..." />
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

7.15 JavaScript/jQuery Codes

```
var fuse;
var BreakException = {};
var corrections = {};
var ignored = [];
var error_codes = {
    'non_word': 2,
    'real_word': 3,
    'rw_ignored': 4,
    'case': 5
}

$(document).ready(function () {
    calculate_size();
    load_dictionary();
    $(window).resize(function() {
        calculate_size();
    });

    $('[data-toggle="tooltip"]').tooltip({placement: 'left'});
});

$(document).on('click', '.text-input', text_focus);
$(document).on('blur', '.text-input', text_blur);
$(document).on('input', '#dic_search', update_dictionary_list);
$(document).on('click', '.spellcheck-btn', spellcheck);

$('.text-input').on('input', function () {
    var text = this.textContent,
        count = text.trim().replace(/\s+/g, ' ').split(' ').length;

    if(text.trim().length > 0){
        $('.status-bar-text').text('Word Count: ' + count);
    }
    else{
        $('.status-bar-text').text('Status bar...');
    }
});

$(document).on('click', '.suggest, .highlight', function(){
    var index = $(this).data('index');

    $('.replacements').remove();
    $('highlight.active, suggest.active').removeClass('active');

    if($('.suggest.active').length > 0){
        $('.suggest.active').removeClass('active');
    }
    if($('.highlight.active').length > 0){
        $('.highlight.active').removeClass('active');
    }

    $('[data-index="' + index + '"]').addClass('active');
    show_suggestion_dropdown(index);
    if(document.querySelector('.text-input [data-index="' + index + '"]')){
        document.querySelector('.text-input [data-
index="' + index + '"]').scrollIntoView({
            behavior: "smooth",
            block: "center"
        });
    }
});

document.querySelector('.text-
input').addEventListener('keydown', function(event) {
    $('.replacements').remove();
    $('highlight.active, suggest.active').removeClass('active');

    if (event.which == 8) {
        s = window.getSelection();
        r = s.getRangeAt(0)
        el = r.startContainer.parentElement
        if (el.classList.contains('highlight')) {
            if (r.startOffset == r.endOffset && r.endOffset == el.textContent.len
gth) {
                event.preventDefault();
                el.remove();
                return;
            }
        }
    }
});

$(document).on('click', '.replace-instance', function(){
    var index = $(this).parents('.highlight').data('index');
    var word = $(this).children('.replace-word').text();

    var pattern = '<span\\sdata\-index\\=\\"' + index + '.+?<\\/span>';
    var re = new RegExp(pattern, "gi");
    text = $('.text-input').html();

    if(findings = text.match(re)){
        findings.forEach(instance => {
            text = text.replace(instance, word);
        });

        $('[data-index="' + index + '"]').remove();
        $('.text-input').html(text);
    }
});

$(document).on('click', '.add-to-dictionary', function(){
    var index = $(this).parents('.suggest.active').data('index');

    if(typeof corrections[index] != "undefined"){
        $.ajax({
            url: '/add_to_dic',
            type: 'POST',
            data: {
                word: corrections[index].word,
                word_pos: corrections[index].pos,
                prev_word: corrections[index].prev_word,
                prev_word_pos: corrections[index].prev_pos,
                next_word: corrections[index].next_word,
                next_word_pos: corrections[index].next_pos,
            },
            beforeSend: function(){
                $('.status-bar-
suggestion').html('Adding <b>' + corrections[index].word + '</b> to dictionary');
            },
            error: function(){
                $('.status-bar-suggestion').html('Failed to add the word!');
            },
            success: function(resp){
                if(resp == 'success'){
                    var pattern = '<span\\sdata\-
index\\=\\"' + index + '.+?<\\/span>';
                    var re = new RegExp(pattern, "gi");
                    text = $('.text-input').html();
                    if(findings = text.match(re)){
                        findings.forEach(instance => {
                            text = text.replace(instance, corrections[index].word
);

                        });

                        $('[data-index="' + index + '"]').remove();
                        $('.text-input').html(text);
                        $('.status-bar-
suggestion').html('<b>' + corrections[index].word + '</b> added successfully.');
                    }
                }
                else{
                    $('.status-bar-suggestion').html('Failed to add the word!');
                }
            }
        });
    }
    else{
        $('.status-bar-suggestion').html('Failed to add the word!');
    }
});

$(document).on('click', '.ignore-suggestion', function(){
    var index = $(this).parents('.suggest.active').data('index');

    if(typeof corrections[index] != "undefined"){
        var pattern = '<span\\sdata\-index\\=\\"' + index + '.+?<\\/span>';
        var re = new RegExp(pattern, "gi");
        text = $('.text-input').html();
        if(findings = text.match(re)){
            findings.forEach(instance => {
                text = text.replace(instance, corrections[index].word);
            });

            $('[data-index="' + index + '"]').remove();
            $('.text-input').html(text);
            ignored.push(index);
        }
    }
    else{
        $('.status-bar-suggestion').html('Failed!');
    }
});

function text_focus(){
    var default_text = 'start typing...';
    var actual_text = $('.text-input').text().trim();

    if(default_text == actual_text.toLowerCase()){
        $('.text-input').text('');
        $('.text-input').css('font-size', '20px');
        $('.text-input').css('color', '#444');
    }
}

function text_blur(){
    var actual_text = $('.text-input').text().trim();

    if(actual_text.length == 0){
        $('.text-input').text('Start typing...');
        $('.text-input').css('font-size', '24px');
        $('.text-input').css('color', '#999');
    }
}

function calculate_size(){
    var footer_size = $('footer').innerHeight();
    var status_size = $('.status-bar').innerHeight();
    var window_size = window.innerHeight;

    boxes_size = window_size - footer_size - status_size - 5;

    $('.text-input, .suggestions, .dictionary').css('height', boxes_size + 'px')
    $('.text-input, .suggestions, .dictionary').css('overflow-y', 'auto')
}

function load_dictionary(){
    $.ajax({
        url: '/dictionary',
        dataType: 'JSON',
        beforeSend: function(){
            $('.dictionary-word-list').html('<div class="mt-
2 loading"><i class="fas fa-spinner fa-spin"></i> Words are loading...</div>');
            $('.status-bar-dictionary').text('Loading...');
        },
        error: function(){
            $('.dictionary-word-list').text('');
            var reload = '<a href="javascript:;" onclick="load_dictionary()"><i c
lass="fas fa-redo-alt"></i> Reload</a>';
            $('.status-bar-dictionary').html('Failed to load! ' + reload);
        },
        success: function(result){
            $('.status-bar-dictionary').html('<i class="fas fa-check text-
success"></i> ' + result.dictionary.length + ' Words loaded!');
            fuse = new Fuse(result.dictionary, {
                shouldSort: true,
                threshold: 0.15,
                location: 0,
                distance: 5,
                keys: ["word"]
            });

            update_dictionary_list(result.common_words);
        }
    });
}

function update_dictionary_list(word){
    if(Array.isArray(word)){
        words = word;
    }
    else if(typeof word != "string"){
        words = fuse.search($('#dic_search').val());
    }
    else{
        words = fuse.search(word);
    }

    counter = 1;
    $('.dictionary-word-list').html('');

    try{
        words.forEach(elem => {
            $('.dictionary-word-list').append('<div>' + elem.word + '</div>');
            if(counter++ == 11) throw BreakException;
        });
    } catch(e){
        if (e !== BreakException) throw e;
    }
}

function spellcheck(){
    text = $('.text-input').html();
    var tmp = document.createElement("DIV");
    tmp.innerHTML = text.replace(/<br>/gi, " ");
    text = tmp.textContent || tmp.innerText || "";
    $('.replacements').remove();

    if(text.trim().length == 0){
        $('.suggest-list').text('');
        $('.status-bar-suggestion').text('There is nothing to check!');
    }
    else{
        $.ajax({
            url: '/spellcheck',
            type: 'POST',
            dataType: 'JSON',
            data: {text: text},
            beforeSend: function(){
                $('.suggest-list').html('<div class="mt-
2 loading"><i class="fas fa-spinner fa-spin"></i> Checking the text...</div>');
                $('.status-bar-suggestion').html('<i class="fas fa-
search"></i> Processing the text...');
                $('.process-duration').remove();
                $('.processed-words').remove();
            },
            error: function(){
                $('.suggest-list').text('');
                var retry = '<a href="javascript:;" onclick="spellcheck()"><i cla
ss="fas fa-redo-alt"></i> Retry</a>';
                $('.status-bar-suggestion').html('Failed to check! ' + retry);
            },
            success: function(resp){
                $('.suggest-list').text('');

                $('.status-bar-text').append('<span class="processed-
words"> (processed: ' + resp.processed_words + ')</span>');
```

```
                $('.status-bar-text').append('<span class="process-
duration float-right"><i class=""></i> ~' + resp.duration + ' seconds</span>');

                prepare_suggestions(resp.corrections);
                show_suggestions();
            }
        })
    }
}

function prepare_suggestions(records){
    corrections = {};
    all_suggestions = 0;
    ignored_numbers = 0;

    records.forEach(function(record, rec_index){
        word = record.word[0];
        prev_word = record.prev_word[0];
        end_sent_err = false;
        start_sent_err = false;

        if(word.toLowerCase() == '|endsent|'){
            word = record.prev_word[0];
            end_sent_err = true;
        }
        if(prev_word.toLowerCase() == '|startsent|'){
            start_sent_err = true;
        }

        pattern_word = word.replace(/[\+\*\{\}\[\]]/gi, '');
        pattern_prev_word = typeof prev_word == 'string' ? prev_word.replace(/[\+
\*\{\}\[\]]/gi, '') : null;
        stopwords_pattern = '[\\s\\,\\.\\?\\!]+';
        highlight_pattern = '(<\/span>)?';

        if(pattern_prev_word && !end_sent_err && !start_sent_err){
            pattern = pattern_prev_word + highlight_pattern + stopwords_pattern +
 pattern_word;
        }
        else if(end_sent_err && start_sent_err){
            pattern = highlight_pattern + '(' + stopwords_pattern + '|^)' + patte
rn_word + '(' + stopwords_pattern + '|$)';
        }
        else if(end_sent_err){
            pattern = pattern_word + '(' + stopwords_pattern + '|$)';
    }
    else if(start_sent_err){
      pattern = '(' + stopwords_pattern + '|^)' + pattern_word;
    }

    active_error = null;
    message = null;
    title = null;

    switch(record.status){
      case error_codes['non_word']:
        active_error = 'non_word';
        title = '<b>' + word + '</b> - check the spelling';
        message = 'The word <b>' + word + '</b> is not in our dictionary.
 ' +
          'If the spelling is correct, you can add it to the dictionary
.';
        break;

      case error_codes['real_word']:
        active_error = 'real_word';
        title = '<b>' + word + '</b> - check the position';
        if(start_sent_err){
          message = 'The word <b>' + word +
            '</b> does not seem a proper candidate for starting the s
entence.';
        }
        else if(end_sent_err){
          message = 'The word <b>' + word +
            '</b> does not seem a proper candidate to end the sentenc
e.';
        }
        else{
          message = 'The word <b>' + word +
            '</b> seems inappropriate in the correct place.';
        }
        break;

      case error_codes['rw_ignored']:
        active_error = 'rw_ignored';
        title = '<b>' + word + '</b> - double check the position';
        message = 'Although the word <b>' + word +
          '</b> seems in a right position based on its type, you may wa
nt to revise it.';
        break;
```

```
            case error_codes['case']:
                active_error = 'case';
                title = '<b>' + word + '</b> - check the case';
                message = 'The word <b>' + word + '</b> may have a case error. Co
nsider reviewing the word';
        }

        if(active_error != null){
            index = word.toLowerCase() + '_' + record.prev_word[0].toLowerCase();

            if(ignored.indexOf(index) < 0 && (typeof corrections[index] == "undef
ined" ||
                corrections[index].error_code > record.status)){

                next_elem = (typeof records[rec_index + 1] != "undefined") ? reco
rds[rec_index + 1] : null;
                next_word = (next_elem) ? next_elem.word[0] : '|ENDSENT|';
                next_pos = (next_elem) ? next_elem.word[1] : '|ENDSENT|';

                corrections[index] = {
                    'word': word,
                    'prev_word': record.prev_word[0],
                    'next_word': next_word,
                    'pos': record.word[1],
                    'prev_pos': record.prev_word[1],
                    'next_pos': next_pos,
                    'error_code': record.status,
                    'error': active_error,
                    'title': title,
                    'message': message,
                    'pattern': pattern,
                    'is_start': start_sent_err,
                    'is_end': end_sent_err,
                    'suggestions': record.suggestions
                }

                all_suggestions++;
            }
            else if(ignored.indexOf(index) >= 0){
                ignored_numbers++;
            }
        }
    });
    notif = (all_suggestions == 0) ? 'No issues found' :
        ('<span class="badge badge-info suggestions-
count">' + all_suggestions + '</span> suggestion(s)');
    $('.status-bar-suggestion').html('<i class="fas fa-check text-
success"></i> ' + notif);

    if(ignored_numbers > 0){
        $('.status-bar-suggestion').append(' | <span class="badge badge-
warning">' +
            ignored_numbers + '</span> Ignored');
    }
}

function show_suggestions(){
    passage = trim_highlight_tags($('.text-input').html());
    suggest_elements = '';
    highlighted_text = '';
    available_suggestions = 0;

    for(correction in corrections){
        data = corrections[correction];
        add_it = false;

        highlight_element =
            '<span data-
index="' + correction + '" class="highlight error_' + data.error + '">' +
                data.word + '</span>';

        re = new RegExp(data.pattern, "gi");
        findings = passage.match(re);

        if(findings){
            findings.forEach(finding => {
                if(typeof finding == 'string'){
                    new_phrase = finding.replace(data.word, highlight_element);
                    prev_passage = passage;
                    passage = passage.replace(finding, new_phrase);

                    if(passage != prev_passage){
                        add_it = true;
                    }
                }
            });

            if(add_it){
                suggest_elements +=
                    '<div class="suggest error_' + data.error + '" data-
index="' + correction + '">' +
                        '<i class="ignore-suggestion far fa-bell-slash" data-
toggle="tooltip" title="Ignore"></i>' +
                        '<i class="add-to-dictionary fas fa-folder-plus" data-
toggle="tooltip" title="Add to Dictionary"></i>' +
                        '<h3 class="suggest_title">' + data.title + '</h3>' +
                        '<p class="suggest-message">' + data.message + '</p>' +
                    '</div>';
                available_suggestions++;
            }
        }
    }

    $('.suggest-list').html(suggest_elements);
    $('.text-input').html(passage);
    $('.suggestions-count').text(available_suggestions);
}

function trim_highlight_tags(text){
    markups = text.match(/<span.*?class\=.+?highlight.+?<\/span>/gi);

    if(markups){
        markups.forEach(markup => {
            word = markup.match(/\>.+\</gi);
            word = word[0].substring(1, word[0].length - 1);
            text = text.replace(markup, word);
        });
    }

    return text;
}

function show_suggestion_dropdown(index){
    elem = '<div class="replacements" contenteditable="false">';

    if(typeof corrections[index] == "undefined" || !Array.isArray(corrections[ind
ex].suggestions) ||
        corrections[index].suggestions.length == 0){
        elem +=
            '<div class="no-instance">' +
            '<i class="far fa-sad-tear"></i> No suggestion available' +
            '</div>';
    }
    else{
        corrections[index].suggestions.forEach(suggest => {
            for(replace in suggest){
                elem +=
                '<div class="replace-instance">' +
                '<label class="replace-
distance">' + suggest[replace] + '</label>' +
                '<label class="replace-word">' + replace + '</label>' +
                '</div>';
            }
        });
    }

    elem += '</div>';

    $('.highlight[data-index="' + index + '"]').append(elem);
}
```

7.16 CSS Style Codes

```
html, body, body *{
    font-family: 'Roboto', sans-serif;
    font-weight: 400;
}

footer{
    background-color: #ece6df;
    color: #969696;
    font-size: 12px;
    position: fixed;
    width: 100%;
    bottom: 0;
    padding: 5px;
    text-align: center;
    margin-left: -15px;
    margin-right: -15px;
}

.status-bar{
    background-color: #ece6df;
    color: #000;
    font-size: 14px;
    border-top: 2px solid #000;
    border-bottom: 2px solid #000;
    padding-top: 5px;
    padding-bottom: 5px;
}

.text-input{
    font-size: 24px;
    color: #999;
    padding: 20px;
    transition: 1s;
    letter-spacing: 0.5px;
    line-height: 30px;
}

.suggestions, .dictionary{
    border-left: 1px solid #999;
}

.section-header{
    font-size: 20px;
    font-weight: bold;
    background-color: #000;
    color: #fff;
    text-align: center;
    margin-bottom: 0;
    padding-top: 15px;
    padding-bottom: 15px;
}

.dictionary-search-input{
    position: absolute;
    bottom: 0;
    border-top: 2px dashed #999;
    width: 99%;
    padding: 5px;
}
.dictionary-search-input #dic_search{
    width: 100%;
    border: 0;
    height: 40px;
}
.dictionary-word-list div{
    padding-top: 10px;
    padding-bottom: 10px;
}
.dictionary-word-list div:nth-child(odd){
    background: #eef;
}
.dictionary-word-list div:nth-child(even){
    background: #d7d7d9;
}

.loading{
    background: #fff !important;
}

.spellcheck-btn{
    position: absolute;
    bottom: 10px;
    right: 10px;
    border-radius: 50%;
    font-size: 40px;
    padding-left: 20px;
}

.suggest-list .suggest{
    text-align: left;
    padding: 10px;
    cursor: pointer;
    transition: .5s;
    position: relative;
}
.suggest-list .suggest:hover{
    text-shadow: 1px 0px 0px #999;
}
.suggest-list .suggest.error_non_word{
    border-bottom: 5px solid #f54545;
    color: #f54545;
}
.suggest-list .suggest.error_real_word{
    border-bottom: 5px solid #f58645;
    color: #f58645;
}
.suggest-list .suggest.error_rw_ignored{
    border-bottom: 5px solid #ecae37;
    color: #ecae37;
}
.suggest-list .suggest.error_case{
    border-bottom: 5px solid #37a8ec;
    color: #37a8ec;
}
```

```css
.suggest-list .suggest.error_non_word.active{
    background: #f54545;
    color: #fff;
}
.suggest-list .suggest.error_real_word.active{
    background: #f58645;
    color: #fff;
}
.suggest-list .suggest.error_rw_ignored.active{
    background: #ecae37;
    color: #fff;
}
.suggest-list .suggest.error_case.active{
    background: #37a8ec;
    color: #fff;
}
.suggest-list .suggest h3{
    font-size: 16px;
    margin-bottom: 0;
}
.suggest-list .suggest h3 b, .suggest-list .suggest p b{
    font-weight: bold;
}
.suggest-list .suggest p{
    font-size: 12px;
    display: none;
    margin-bottom: 0;
    margin-top: 5px;
    padding-top: 5px;
    border-top: 1px dashed #fff;
}
.suggest-list .suggest.active{
    padding-top: 25px;
}
.suggest-list .suggest.active p{
    display: block;
}
.suggest-list .suggest.active .ignore-suggestion{
    display: inline-block;
}
.suggest-list .suggest.error_non_word.active .add-to-dictionary{
    display: inline-block;
}
.suggest-list .suggest .ignore-suggestion{
    display: none;
    position: absolute;
    top: 2px;
    right: 2px;
    font-size: 20px;
}
.suggest-list .suggest .add-to-dictionary{
    display: none;
    position: absolute;
    top: 2px;
    right: 35px;
    font-size: 20px;
}

.text-input .highlight{
    padding: 2px 5px;
    transition: .5s;
    cursor: pointer;
    border-radius: 0;
    position: relative;
    display: inline-block;
}
.text-input .highlight.active{
    border-radius: 12px;
}
.text-input .highlight.error_non_word{
    color: #f54545;
    border-bottom: 3px solid #f54545;
}
.text-input .highlight.error_real_word{
    color: #f58645;
    border-bottom: 3px solid #f58645;
}
.text-input .highlight.error_rw_ignored{
    color: #ecae37;
    border-bottom: 3px solid #ecae37;
}
.text-input .highlight.error_case{
    color: #37a8ec;
    border-bottom: 3px solid #37a8ec;
}
.text-input .highlight.error_non_word.active{
    background: #f54545;
    color: #fff;
}
.text-input .highlight.error_real_word.active{
    background: #f58645;
    color: #fff;
}
.text-input .highlight.error_rw_ignored.active{
    background: #ecae37;
    color: #fff;
}
.text-input .highlight.error_case.active{
    background: #37a8ec;
    color: #fff;
}

.replacements{
    position: absolute;
    top: 40px;
    left: -1px;
    width: auto;
    min-width: 150px;
    background-color: #e7e7e7;
    font-size: 14px;
    box-shadow: 0px 4px 10px #333;
    color:#333;
    line-height: normal;
    z-index: 1;
}
.replacements .no-instance{
    text-align: center;
    font-weight: bold;
}
.replacements .replace-instance{
    border-bottom: 1px solid #333;
    cursor: pointer;
    transition: 0.5s;
}
.replacements .replace-instance .replace-distance{
    display: inline-block;
    width: 20%;
    background: #333;
    color: #fff;
    text-align: center;
    padding: 7px;
    margin: 0;
}
.replacements .replace-instance .replace-word{
    display: inline-block;
```

```css
    width: 80%;
    padding: 7px;
    margin: 0;
    text-align: center;
}
.replacements .replace-instance:hover .replace-word{
    background-color: #28a745;
    color: #fff;
}
.replacements .replace-instance:hover{
    box-shadow: 0 2px 2px #333;
}
```