

Autocorrection Tool Using Edit Distance

Report

by

Dhruv Doshi

Roll No: 1814002

Shubham Bhakuni

Roll No: 1814006

Labdhi Jain

Roll No: 1814015

Kunj Gala

Roll No: 1814021



Department of Information Technology
K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch 2021

Table of Contents

1. Introduction	3
2. Applications of similarity measure:.....	4
2.1 Application 1:	4
2.2 Application 2:	4
2.3 Application 3:	5
2.4 Application 4:	6
3. Methodology used:	8
3.1 Levenshtein Distance (or Edit Distance):	8
3.2 Use of Levenshtein Distance in auto correction:	8
3.3 Step by Step Implementation:.....	9
3.4 Workflow diagram:.....	11
4. Implementation:	12
5. Results and discussion:	13
5.1 Step wise illustration of the website:	13
5.2 Testing the application:	14
5.3 Inference:	15
6. References:	15

1. Introduction

We all have experienced that in chat applications or word processing applications like Microsoft Word, Google Docs, etc., we make unintended spelling errors. For example, simple errors such as writing “sory” instead of “sorry”. But nowadays spell check applications are easily available to avoid such errors. They might show a red underline or replace the incorrect word with the correct word.

The question is, how to find the most similar word to the misspelt word. That's when Edit Distance comes into play. Edit distance determines the distance between two words . It gives us a number which represents the degree of similarity between them. The total number of transformations which is also known as edits that are required to convert one word to another is computed by the Edit distance. The distance it returns is inversely proportional to the similarity. It means that the lower the distance, the more similar the words are and vice versa. Continuing with the example discussed above, the algorithm will search for the closest match for the word “sory” in the dictionary and “sorry” would be the suggest fix for the misspelled word.

Dynamic programming approach will be used in this algorithm to calculate the Edit Distance by preparing a distance matrix which will be further explained in detail. Using Python programming language, we will be creating a simple application with autocorrect and autocomplete features. This application will implement the Edit distance to choose the similar words in the dictionary whose degree of similarity is closest to the given word.

2. Applications of similarity measure:

A similarity measure is an essential tool for estimating the degree of similarity between two objects that are being compared. Similarity Measures are very useful and popular in several fields such as machine learning, pattern recognition, image processing and decision making

2.1 Application 1:

Title: A Parallel Approach of Weighted Edit Distance Calculation for Log Parsing

Citation: [Xingyuan Ren](#); [Lin Zhang](#); [Kunpeng Xie](#); [Qiankun Dong](#)

Application: Log parsing

Summary: The Log Key Extraction Algorithm has been used in this paper for Log parsing. The algorithm finds similarity between different log messages and then finds the log message template. The similarity between the log messages is calculated by finding the weighted edit distance between them. It is found that for larger logs, in practical situations a lot of time is consumed to calculate the edit distance on a CPU hence the paper proposes the use of GPU i.e., Graphical Processing Unit. It was finally found out that the GPU reduces the computational time to find the edit distance to a great extent. The formula of the weighted edit distance used in the LKE Algorithm was :

$$\text{WED}(m1, m2) = \sum_{i=1}^{NO} \frac{1}{1 + e^{(x_i - v)}}$$

2.2 Application 2:

Title: An experimental Tagalog Finite State Automata spellchecker with Levenshtein editdistance feature

Citation: [Joseph Marvin R. Imperial](#); [Czeritonnine Gail V. Ya-On](#); [Jennifer C. Ureta](#)

Application: Spelling Checker for the Tagalog language

Summary: In this paper, a 300 words wordlist is used along with 3 inflected words of each

Root	Form 1	Form 2	Form 3
kain	kainan	kainin	kaingin
kagat	kagatin	kagatan	kagatum

The input words are first converted in a regular expression then into an epsilon NFA(Non finite automata) and finally through a DFA(definite finite automata) to check if the word is recognized by the Tagalog language. After this spelling checking is performed by edit distance and depending upon the edit distance spellings suggestions are given. If the edit distance is larger then even correct spellings could be provided with an incorrect suggestion. The output provided in the paper for an edit distance of 2, is of the form:

```
Enter string: Pwede bang abotan mo ako ng tubig?

-----
SPELLING SUGGESTIONS:
-----

Pwede bang abotan mo ako ng tubig?
bang = ['baba', 'bali', 'baba']
abotan = ['abutin', 'abutan', 'kabitan']
```

```
Enter string: Pwede bang abotan mo ako ng tubig?

-----
SPELLING SUGGESTIONS:
-----

Pwede bang abotan mo ako ng tubig?
bang = ['baba', 'bali', 'baba']
abotan = ['abutin', 'abutan', 'kabitan']
```

2.3 Application 3:

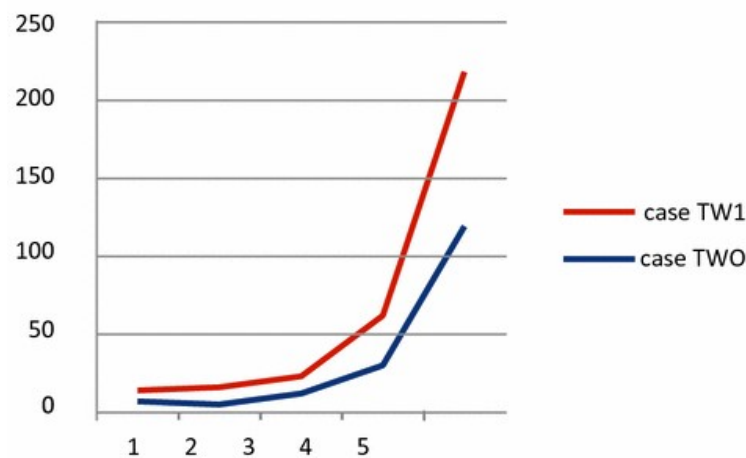
Title: Enhancing Levenshtein's Edit Distance Algorithm for Evaluating Document Similarity

Citation: Shama Rani; Jaiteg Singh

Application: Evaluating document similarity and checking plagiarism

Summary: In this paper Levenshtein's Edit Distance is used to compare a document by ignoring images with search engine results. The values of document 1 and 2 are stored in a separate matrix and each value is matched. If the values match then the cost is 0 otherwise the cost is 1. The paper emphasizes on a new algorithm of edit distance which removes stop words before comparing documents and it is discovered that each document has 20 to 25% stop words and

the efficiency is also improved after removing them. Below is the graph of the time taken to calculate edit distance before and after the removal of stop words.



2.4 Application 4:

Title: Implementing The Levenshtein Distance for Word Autocompletion and Autocorrection


Citation: Ahmed Fawzy Gad

Application: Word Auto Correction

Summary : The word auto correction feature is used by various search engines and chatting applications.



In this article, the author uses a dictionary with 1000 basic words. when user enters a word the dictionary is scanned. Each word scanned from the dictionary is compared against the user's inputted word and edit distance is calculated for each pair. Finally the words in the dictionary which have the least edit distance with the searched word, is returned as a result to the user. An example is provided in the article as follows:



A screenshot of a text input field. The word 'page' is entered in the field. Below the input field, two suggestions are listed: 'paper' and 'age'. The suggestions are displayed in a light blue box.

Here the searched word was page and the system returned 2 suggestions, age and paper, that could help in auto correction or autocompletion. The two important functions that were used for the implementation was the levenshteinDistanceDP() to calculate the distance between the prefixes and the calcDictDistance() function which returns the closest matched word from the dictionary.

3. Methodology used:

3.1 Levenshtein Distance (or Edit Distance):

Levenshtein Distance is the number of edits which are used to make one word same as another. Levenshtein distance is the measure of how alike 2 words are. In the demonstration we have used dynamic programming for levenshtein distance.

Levenshtein distance Algorithm uses a 2 dimensional matrix to find the distance between 2 strings. The last value of the matrix holds the value of levenshtein distance between 2 strings.

Levenshtein value tells how much different two strings are. The possible edits allowed in the levenshtein distance are

- Adding a character to either of the strings
- Delete a character from either of the strings
- Replacing a character of string.

The cost of each operation is considered as 1. Though this can be changed accordingly.

Each value of the levenshtein matrix (i,j) can be considered as the minimum edits required to transform 1st String of i character into the second string of the first 'j' characters. An extra row is present in the matrix which represents empty character/string. This ensures the algorithm produces correct results for the edge cases.

3.2 Use of Levenshtein Distance in auto correction:

While typing humans make errors, here we can use the levenshtein distance to predict which word the user intended to type instead of the current written word. The wrong word is used to calculate the average the levenshtein distance with every possible word in the dictionary which can then be sorted in ascending order. For example “disatnce” and “distance” are closely related according to the levenshtein distance, having a value of 2 .

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2	2	1	2	3	4
l	3	3	2	1	2	2
m	4	4	3	2	2	3

3.3 Step by Step Implementation:

1. **levenshteinDistanceDP** accepts 2 string, returns the levenshtein distance between the the 2 strings. Given strings s1 and s2 of length m and n, a 2-D Numpy matrix is made, consisting of m+1 rows and n +1 columns.
 - a. The matrix is initially filled with all zeros.
 - b. Edges are completed. I.e. Cost of converting a empty string to another

```
for t1 in range(len(token1) + 1):
    distances[t1][0] = t1

for t2 in range(len(token2) + 1):
    distances[0][t2] = t2
```

2. Then the Matrix is filled.
 - a. If the ith and jth character are same then there no increase in edit distance
 - b. Else the cost increases by the minimum of either Addition, subtraction or replacement of character.

```
for t1 in range(1, len(token1) + 1):
    for t2 in range(1, len(token2) + 1):
        if (token1[t1-1] == token2[t2-1]):
            distances[t1][t2] = distances[t1 - 1][t2 - 1]
        else:
            a = distances[t1][t2 - 1] #Adding to String 2
            b = distances[t1 - 1][t2] # Removing from String 1
            c = distances[t1 - 1][t2 - 1] # replacing character

            if (a <= b and a <= c):
                distances[t1][t2] = a + 1
            elif (b <= a and b <= c):
                distances[t1][t2] = b + 1
            else:
                distances[t1][t2] = c + 1
```

- c. Return the Levenshtein value which is the last value of the matrix

```
# printDistances(distances, len(token1), len(token2))
return distances[len(token1)][len(token2)]
```

3. **calcDictDistance** reads all the words from the dictionary and finding the closest word.

Here we use a dictionary which contains all the 3000 valid words.

a. Opening Diction and storing all the words in a list

```
def calcDictDistance(word, numWords):
    file = open('3000 words.txt', 'r')
    lines = file.readlines()
    file.close()
    dictWordDist = []
    wordIdx = 0
```

b. The function then filters the words according to their distance value

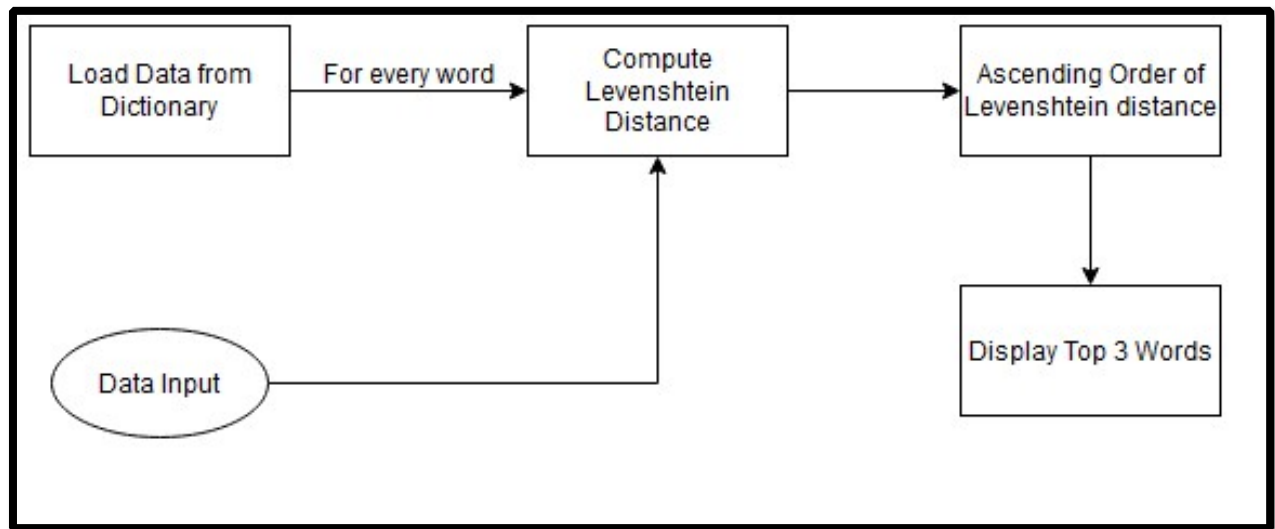
```
for line in lines:
    wordDistance = levenshteinDistanceMatrix(word, line.strip())
    if wordDistance >= 10:
        wordDistance = 9
    dictWordDist.append(str(int(wordDistance)) + "-" + line.strip())
    wordIdx = wordIdx + 1
```

c. The list is then sorted which leaves the word which is the most closely related to the input word at the first index(least levenshtein distance).

d. We return the first 3 words in the sorted list

```
3     closestWords = []
2     wordDetails = []
1     currWordDist = 0
0     dictWordDist.sort()
1     # print(dictWordDist)
2     for i in range(numWords):
3         currWordDist = dictWordDist[i]
4         wordDetails = currWordDist.split("-")
5         closestWords.append(wordDetails[1])
6     return closestWords
7
```

3.4 Workflow diagram:



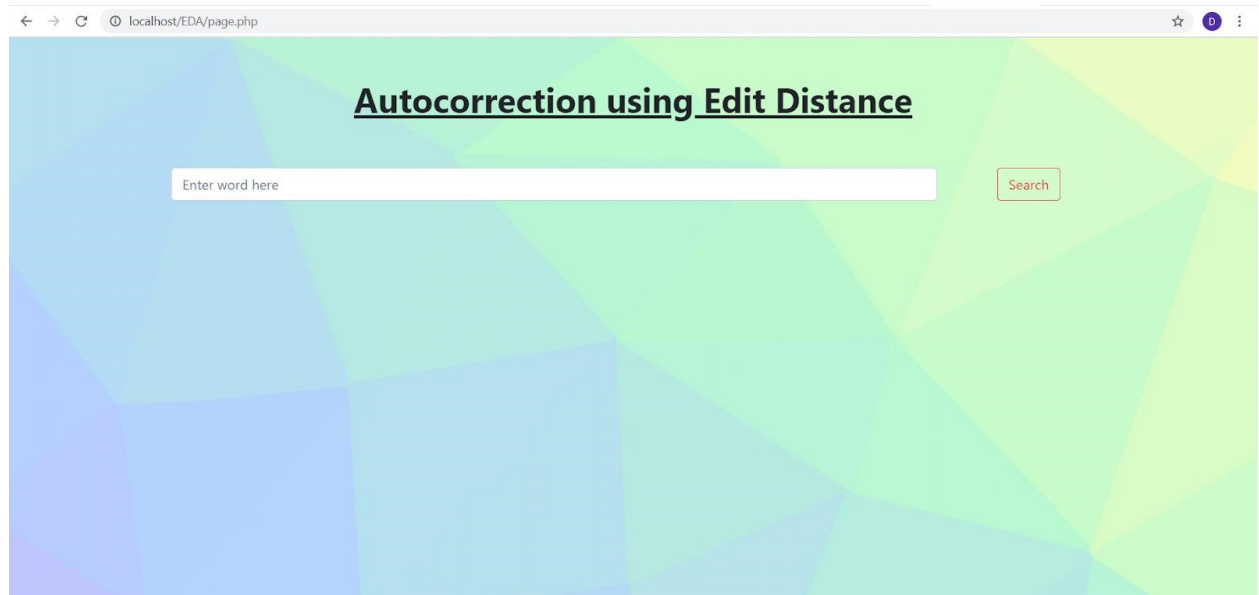
4. Implementation:

Python Programming Language	Python is an interpreted high-level general-purpose programming language. We have used python programming language to develop the program for the Implementation of Word Autocompletion and Autocorrection using Edit Distance. By using this language we have created three user defined functions and then at the end we call one function by passing value as a word.
NumPy Library	NumPy is a library for the Python programming language, adding support for big, multi-dimensional matrices and arrays, in conjunction with an oversized assortment of high-level mathematical functions to work on these arrays. We have used NumPy to create a 2-D matrix. The distances between every prefix of the 2 words that are being compared will be stored in this matrix.
Sys Module	The variables and functions which are utilized to change different sections of the python runtime environment are provided by the sys module of python. We have used the sys module to extract the value of the variable passed by the php page to the python page.
Dictionary	We have downloaded a dictionary consisting of 3000 common words. It is a text file from which is read and each word is extracted. Then using the user defined functions the program returns the best matched words.
HTML	The Hypertext Markup Language (HTML) is the language used for designing the documents that are to be presented in a web browser. We have developed a web page to display the implementation in a better way. Using Html we have created the base of our web page and used the form feature of HTML to create a form that will take the input from the user.
Bootstrap	Bootstrap is an open-source and free CSS framework which is used for developing responsive webpages which can also be accessed on mobile phones. We have used Bootstrap to style and provide the user with the search bar to put the input and with a search button upon which on clicking the user will be able to see the results of the program.
PHP	PHP is a scripting language for server. It is also a strong tool for creating interactive and dynamic Website pages. We have used the htmlspecialchars() function of PHP to convert special characters inputted by the user into HTML entities. Shell_exec function of php is also used to execute python code in php script.

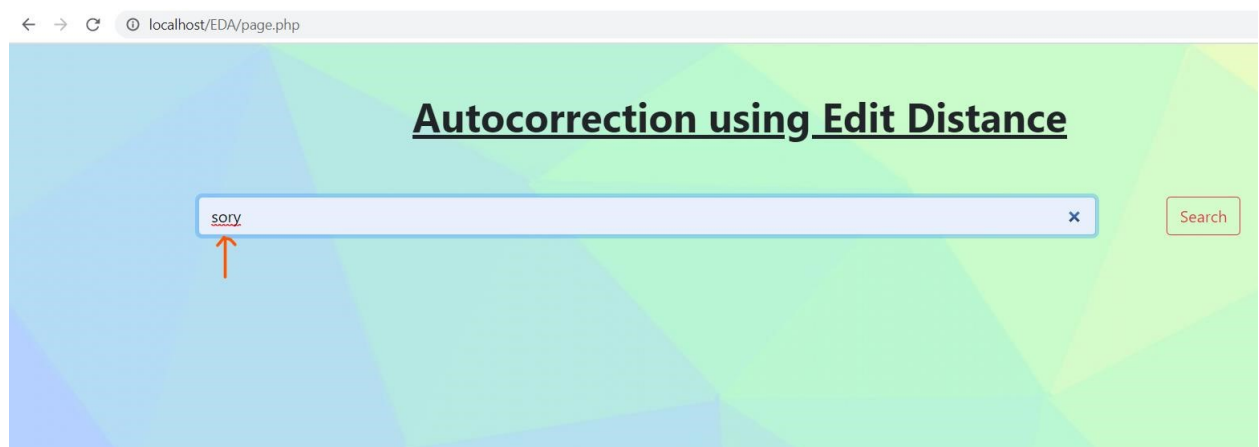
5. Results and discussion:

5.1 Step wise illustration of the website:

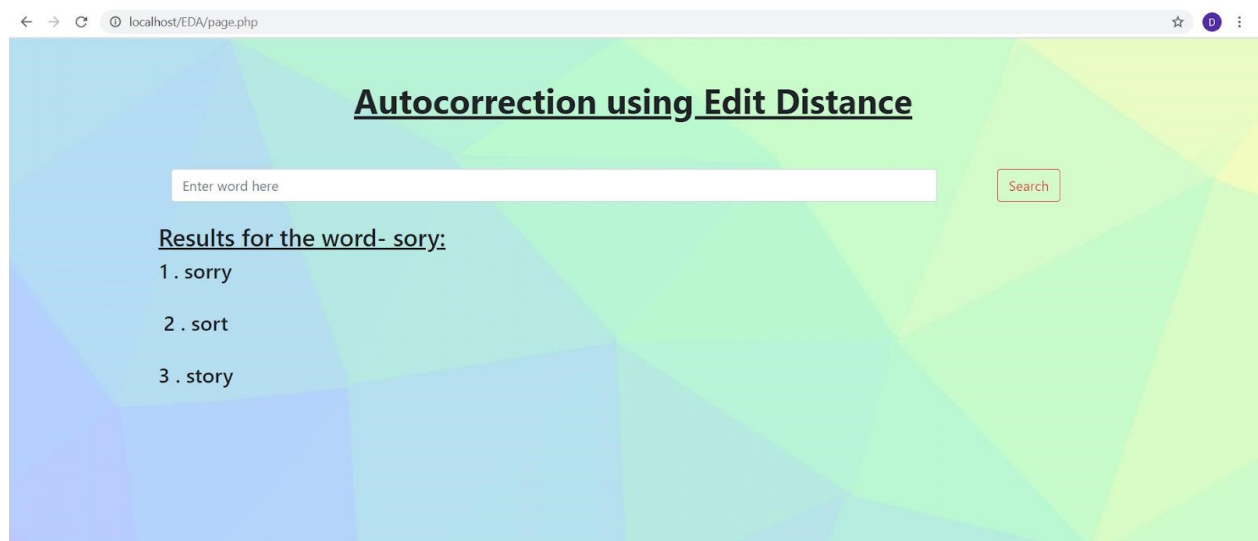
Open the website home page:



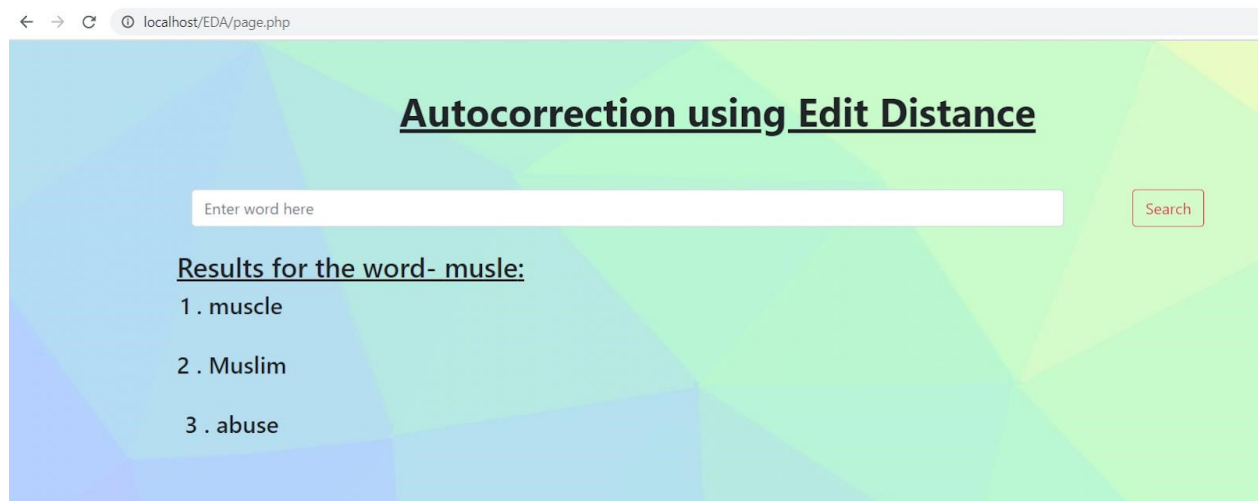
Enter word and then click on search button (sorry):



Autocorrection results are shown below:



Another example (entering musle):



5.2 Testing the application:

Sr. No.	Word entered	Dictionary word	Distance	Rank expected*	Rank observed*	Pass/Fail
1.	sory	sorry	1	1	1	Pass
		sort	1	2	2	Pass
		story	1	3	3	Pass
2.	musle	muscle	1	1	1	Pass
		muslim	2	2	2	Pass
		abuse	3	3	3	Pass

5.3 Inference:

From the above screenshots we can infer that autocorrection using Edit Distance works as intended. It shows the closest three words that match to the word entered by the user. In this given illustration the closest words to the entered word “sory” are shown on the website, which are sorry, sort and story. In an application which uses autocorrection only the first word (closest match) will be shown (or replaced with).

6. References:

- [1] Xingyuan Ren; Lin Zhang; Kunpeng Xie; Qiankun Dong, “A Parallel Approach of Weighted Edit Distance Calculation for Log Parsing” , IEEE International Conference on Computer and Communication Engineering Technology (CCET), 2019.
- [2] Joseph Marvin R. Imperial; Czeritonnice Gail V. Ya-On; Jennifer C. Ureta, “An experimental Tagalog Finite State Automata spellchecker with Levenshtein edit-distance feature” , International Conference on Asian Language Processing (IALP), 2019.
- [3] Shama RaniJaiteg Singh, “Enhancing Levenshtein’s Edit Distance Algorithm for Evaluating Document Similarity” , International Conference on Computing, Analytics and Networks, 2018.
- [4] <https://blog.paperspace.com/measuring-text-similarity-using-levenshtein-distance/>
- [5] <https://blog.paperspace.com/implementing-levenshtein-distance-word-autocompleteautocorrect/>
- [6] [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
- [7] <https://www.geeksforgeeks.org/python-sys-module/>
- [8] <https://www.javatpoint.com/php-string-htmlespecialchars-function>
- [9] <https://www.php.net/manual/en/function.shell-exec.php>

Originality report

COURSE NAME

Plag check

STUDENT NAME

KUNJ GALA

FILE NAME

EDA_IA2

REPORT CREATED

Apr 30, 2021

Summary

Flagged passages	1	0.7%
Cited/quoted passages	0	0%

Web matches

semanticscholar.org	1	0.7%
---------------------	---	------

1 passage

Student passage FLAGGED

Title: An experimental Tagalog Finite State Automata spellchecker with Levenshtein editdistance feature

Top web match

An experimental Tagalog Finite State Automata spellchecker with Levenshtein edit-distance feature.

@article{Imperial2019AnET, title={An experimental Tagalog Finite State Automata spellchecker with...

An experimental Tagalog Finite State Automata spellchecker with

... <https://www.semanticscholar.org/paper/An-experimental-Tagalog-Finite-State-Automata-with-Imperial-Ya-On/7a42f6e7081773a048f236d9e653fcde272be63e>
