Spell Checker

HW2

Harsha Vardhan, Khurdula.

CS 59000 – 08 Natural Language Processing

Purdue Fort Wayne

Instructor: Jon. Rusert

February 2nd, 2023

Task at Hand

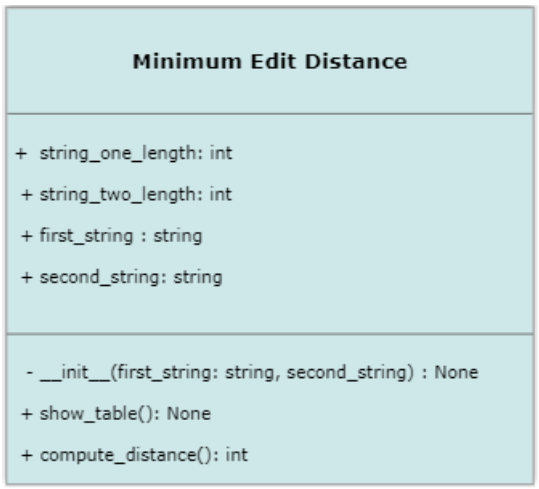
To create a simple spell checker program in Python, which uses the dictionary that was previously created in HW1. This script is expected to run on python shell, with method call being **spell_checker()**. Which takes user input primarily text, and suggests any misspelt words that are not part of vocabulary present in the **dictionary.txt**, and suggests possibly correct strings using **Minimum Edit Distance algorithm**.

Approach:

The initial approach is to write a python script to this problem was to create a console application with *Loose Coupling* of modules namely MinimumEditDistance (class) and SpellChecker (class). Where the MimimumEditDistance class implements the Minimum Edit Distance Algorithm and SpellChecker class just uses this implemented class to seek input from user, find errors in it and make suggestions based on how far the string is from the suggested string (how high the edit distance is from the current string to target string.)

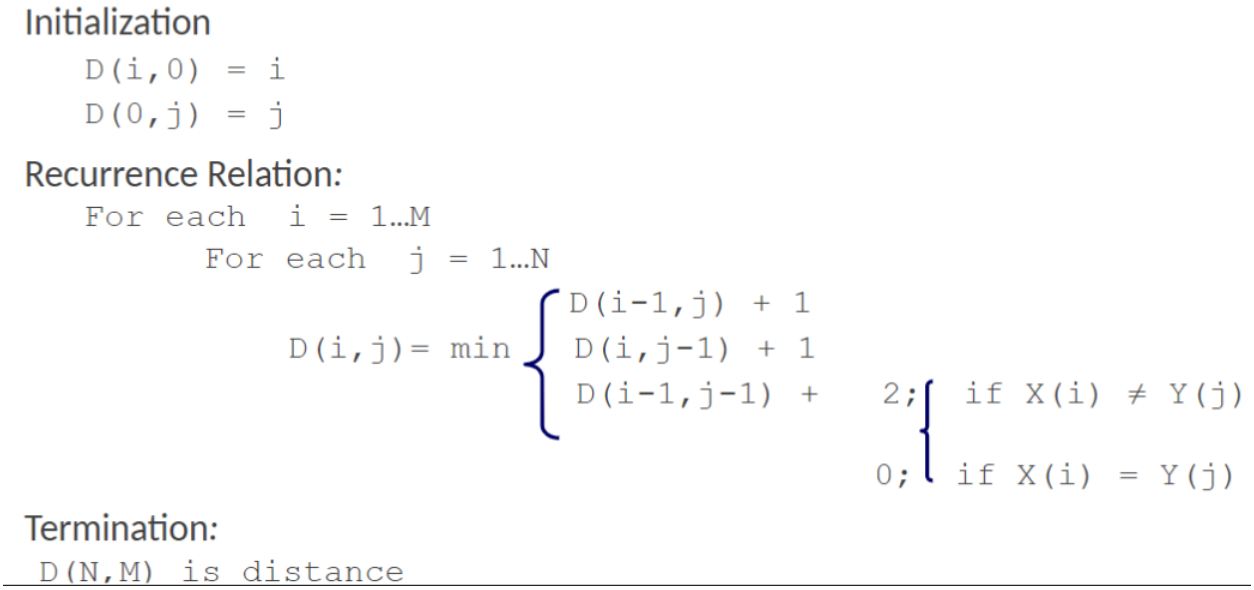
Minimum Edit Distance Algorithm:

This will be our main idea to identify words as possible correct replacements for unknown words, this is implemented by using **MinimumEditDistance** class.



Class Diagram I. MinimumEditDistance()

This class basically implements the algorithm, in a linear fashion with the object creation, all the attributes are initialized by constructor including the two-dimensional array ***self.table*** which acts as an edit distance matrix, followed by the compute_distance() method that fills self.table by choosing the minimum cost at each iteration, and returns the total cost incurred to transform a initial string into target string i.e. **self.table[i][j].** Following is a recurrence relation for this implementation.



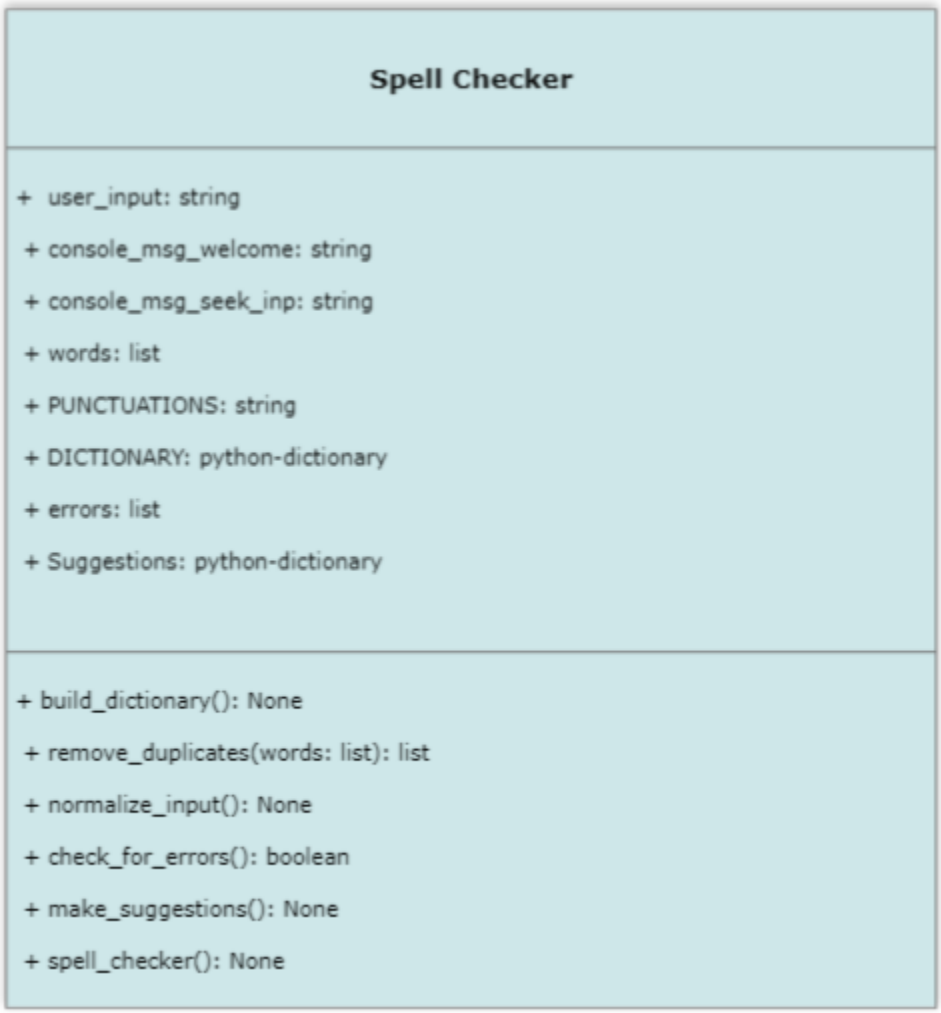
Note: the cost incurred if the two literals are not equal, i.e. first_string(i) != second_string(j) is 2, instead of 1 which is used in certain use cases of this algorithm.

Cost of each operation while filling the distance matrix:

- Insertion: self.table[i-1][j] + 1
- Deletion: self.table[i][j-1] + 1
- Replacement: self.table[i-1][j-1] //if first_string[i] == second_string[j]
  self.table[i-1][j-1] + 2 //otherwise.

And the final cost returned is self.table[i][j].

SpellChecker Module:



Class Diagram II. SpellChecker()

This python module is used to perform three five basic functionalities:

- **Create a dictionary** that stores all available words from dictionary.txt that was previously created from previous hw assignment and **create an instance of MinimumEditDistance class** to use it in further logic. This dictionary is created along with all global attributes initialization by using the ***constructor and build_dictionary() methods***.

- **Seek input from user**, this input can contain irrelevant literals that might cause anomalies while suggesting words, hence we normalize the user input by using

self.normalize_input() method, which lowers the text to have case consistency, gets rid of all punctuations using ***re.sub()*** method, tokenizes user by using ***re.findall()*** method, finally removes any duplicates within this list of tokens (list of words, given as input by user).

- **Check for misspelt words,** the list of words are then iterated over to see if any token does not exist in *self.DICTIONARY*, which is a constant dict, that stores all our vocabulary, any word that does not exist in this data structure is considered to be an error. This is implemented by calling ***self.check_for_errors()*** method. This method returns a Boolean value, which is True in the case that the user has misspelt words, False otherwise.

- **Make Suggestions,** the errors are then captured within a global attribute *self.errors* which are then iterated over to find their correct counter parts, this is done by calling ***self.make_suggestions()*** method that uses the ***MinimumEditDistance.compute_distance()*** method to determine the cost that is incurred to transform the initial string to target string.

- **Determine list of suggested words instead**, the final step is to store all the words, with cost less than or equal to the length of the word. And serve them to the user.

COST <= Length(misspelt_word)

I

Observations:

- A word which is not available in our vocabulary is considered as an error by this program.

- Since the Dictionary created from the text file TheWaroftheWorlds.txt does not contain all possible correctly spelt words of the English Language, at times even a valid word is considered as an error, this can be enhanced by asking the user if any of the words that they've entered are inaccurately identified as a misspelt word and then store that word to our vocabulary for future.

- The number of suggested words for replacement solely depends on the maximum cost that we want to incur in order to replace that word.

  For ex. The cost incurred to transform the phrase ‘yo’ which is identified as a misspelt word by the program, suggests the following words as correct replacements: ‘you’ with cost 1, ‘youd’ with cost 2.

- The above expression determines the count of total words suggested, and it also inaccurately gives **30+** suggestions for the word, incarnate since there are many words, which can be formed by replacing, adding or deletion certain characters within this string. Ex. The above expression gives an array of following recommendations: ‘invaders’ (cost: 9), ‘invariably’ (cost: 9), ‘invasion’ (cost: 9), ‘invested’ (cost: 9), ‘invited’ (cost: 8), and so on.

- But the argument that I could make is that the algorithm has no metric to know the context of any word, it is just a numerical way to suggest words. Ex. **‘Incarnate’** is a correctly spelt English word, which does not exist in our Dictionary and certain words that are recommended such as ‘iron’ (cost: 7), ‘is’ (cost: 9), ‘its’ (cost: 8), ‘island’ (cost: 9), and so on, have no relevance in real word text correction.

- Idea: One way around this loop hole is to understand the fact that the cost incurred is directly proportional to the percentage of change in the misspelt word. And by tweaking the above expression, we can control how far we want to change our misspelt word. This would drastically reduce the number of words that are suggested to the user.

$$\boldsymbol{COST} \propto \%\, \boldsymbol{OF\ CHANGE\ IN\ MISSPELT\ WORD}$$

II

- **Attempt 1.** Let’s assume that we don’t want to change the misspelt word more than 50%, then the expression would be modified as:

$$COST <= \left(\frac{Length(misspeltWord)}{2}\right)$$

III

Upon using this expression the output array of words are greatly reduced! From 30+ suggestions for the misspelt word '**incarnate**' to just a single suggestion of the phrase '**inanimate**' with cost 4! So does it make our SpellChecker any more accurate? Not really. Certain words, such as 'India' return an empty replacement suggestion. I.e. the SpellChecker identifies this as a misspelt word, but since we are not okay with changing most of the misspelt word, no words are recommended.

- **Attempt 2.** So what's the sweet-spot then? Such that we do not get any empty suggestions for misspelt words and try and reduce the output array of suggestions for each misspelt word? So this time upon trying the following expression:

$$COST <= (Length(MisspeltWord) - 1)$$

IV

This equation is not much better than expression I, although it reduces the output array as suggestions. But still provides more than abundant suggestions. For the word 'incarnate' same problem persists as expression I.

- **Attempt 3.** So let's say we are okay with 75% of change in the misspelt word, this enables us to get suggestions for most misspelt words.

$$COST <= \left(Length(MisspeltWord)/1.5\right)$$

V

This works as the best fit possible, to properly balance the number of suggestions we make and the cost that we incur to reach the target word. But! For random string like, 'Kuchu!' the SpellChecker has the same issue as attempt 1, no words are recommended as replacements. With the results in mind, I decided to add an attribute garbage strings,

which shows that the user has given a string/word as an input that is not misspelt in fact it doesn't exist within the language. With that, our SpellChecker does a better job! Inputs like 'asdfasdfasdfasdf' are clearly randomized and are garbage, these are identified and returned to user, rather than providing an empty array of outputs. E.x. for the phrase 'kuchu! Asdfkasdfasdfk', the SpellChecker identifies both of these as garbage strings and does not try to provide suggestions.

Conclusion:

I would say that the best iteration is clearly ***Attempt 3, with expression V***. it solves the additional issues that are faced by SpellChecker. And enhances the suggestions that it makes. However, this is just a program that measures distances between two strings (how different are they?) and does not add any context, weight or probability to words in order to provide realistic suggestions. I am pretty content with my work on this SpellChecker.

Additional Notes:

- In the case that you encounter a filenotfound error for dictionary.txt, please make sure you have the dictionary.txt file provided on Bright space submission, within the same folder as these two python scripts.
- ! NOTE: all of these scripts must be in same directory to avoid any file_not_found exception.
- Use the following commands to run the script, open a terminal within the directory where all files are present and type:
- 1. "python"
- 2. ">>>from SpellChecker import SpellChecker"
- 3. ">>>SpellChecker().spell_checker()"

References:

Khurdula, H. V. (2023, January 14). *NLP_Dictionary*. Github. https://github.com/Khurdhula-Harshavardhan/NLP_Dictionary.git

Jurafsky, D. (2020, January 8). *NLP: Minimum Edit Distance* [PowerPoint slides]. Stanford University. https://web.stanford.edu/~jurafsky/slp3/slides/2_EditDistance_Jan_08_2020.pdf

Khurdula, H. V. (2023, February 1). *Spell-Checker*. Github. Retrieved February 2, 2023, from https://github.com/Khurdhula-Harshavardhan/Spell-Checker.git