R.C. PATEL INSTITUTE OF TECHNOLOGY
An Autonomous Institute

DepartmentofComputerScience&Engineering
(DataScience)

Laboratory Manual

Subject: - ACL (PECS7042T) **Semester: - VII**

Class: - T. Y. B. Tech **Experiment No. : - 8**

Aim: - Implement Spelling Check, Spelling Correction and Auto complete using Language models

Requirement: -Python versions 3.7, 3.8, 3.9, 3.10 or 3.11, jupyter notebook

Theory: -

Build a dictionary: Create a dictionary of valid words that you will use for spell checking and correction. This can be a pre-existing dictionary or a custom one based on your specific requirements.

Spelling check: Given an input word, compare it against the words in your dictionary using an edit distance metric like Levenshtein distance. Compute the edit distance between the input word and each word in the dictionary. If the edit distance is below a certain threshold (e.g., 2), consider the word to be a valid suggestion.

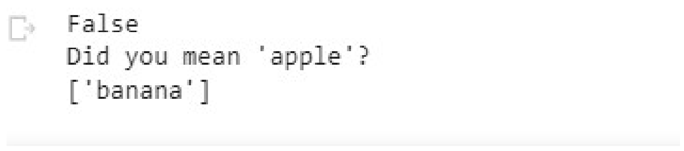
Spelling correction: If the input word is not found in the dictionary or has a high edit distance, you can suggest corrections based on the edit distance metric. Generate a list of candidate words by applying insertion, deletion, substitution, or transposition operations on the input word. Compute the edit distance between each candidate word and the words in the dictionary. Rank the candidates based on their edit distance and suggest the top-ranked corrections.

Auto complete: To implement auto complete, you can use the edit distance matrix to suggest completions for a partially typed word or sentence. Start by finding the closest matching word in the dictionary based on the edit distance with the partial input. Then, suggest words that have the closest edit distance to the matching word as auto complete options.

Post-process and display results: Present the spelling check, correction, or auto complete suggestions to the user in a user-friendly format, such as a dropdown menu or a list of options. Allow the user to select the desired suggestion or continue typing.

```
import numpy as np

# Edit distance function
def edit_distance(word1, word2):
    m, n = len(word1), len(word2)
    dp = np.zeros((m + 1, n + 1))

    for i in range(m + 1):
        dp[i][0] = i

    for j in range(n + 1):
        dp[0][j] = j

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = min(
                    dp[i - 1][j] + 1,  # Deletion
                    dp[i][j - 1] + 1,  # Insertion
                    dp[i - 1][j - 1] + 1  # Substitution
                )

    return dp[m][n]

# Dictionary of valid words
dictionary = ["apple", "banana", "cherry", "dog", "elephant"]

# Spelling check
def spelling_check(word):
    return word in dictionary

# Spelling correction
def spelling_correction(word):
    min_distance = float('inf')
    correction = None

    for dict_word in dictionary:
        distance = edit_distance(word, dict_word)
        if distance < min_distance:
            min_distance = distance
            correction = dict_word

    return correction
```

```
# Autocomplete
def autocomplete(partial_word):
    completions = []

    for dict_word in dictionary:
        if dict_word.startswith(partial_word):
            completions.append(dict_word)

    return completions

# Testing the functions
word = "aple"
print(spelling_check(word))  # False

correction = spelling_correction(word)
if correction is not None:
    print(f"Did you mean '{correction}'?")  # Did you mean 'apple'?

partial_word = "ban"
print(autocomplete(partial_word))  # ['banana']
```



```
False
Did you mean 'apple'?
['banana']
```