English Auto-correct in Python

Introduction

Make a python program to automatically detect the spelling mistakes and correct it.

Objective is to implement the auto correct algorithm to find the misspelled word and give the most probable correct words list.

Abstract of Past Similar Approaches

1. <https://github.com/somyajain99/english-autocorrect>

* In this repository it calculates the given word with all the other words vocabulary and calculate the minimum edit distance.
* Print only the words that are smaller than edit distance of one or smaller.

1. <https://github.com/filyp/autocorrect/tree/master>
   * autocorrect/typos.py

constructor generates slices of the word. This class gives possible slices of given word. By insertion, deletion, transpose, replaces,

each method generates possible typographical error.

* + - autocorrect/word\_count.py

def get\_words: this extracts the word based on each language specific regx pattern. It removes capitalized words that appear after punctuation or at the beginning of a line and yields all remaining words that match the language's word regex.

def count\_words: Extracts words from a file based on the given language and encoding. Counts the occurrences of each word. Counts the occurrences of each word. Sorts the words by frequency in descending order.

* + - autocorrect/constants.py

word\_regexes: this dictionary gives regx to extract words from each language from this regx we can get only the words that are matching only to the specific language.

alphabets: this dictionary gives regx to extract each matching letters of any given language.

* + - autocorrect/\_\_init\_\_.py

class ProgressBar: this will show the progress of downloading element

def load\_from\_tar: this will download the dictionary from the internet. This downloads the frequency dictionary from the internet. Downloads the file file\_name="word\_count.json"

class Speller:

self.nlp\_data: this is frequency dictionary in previously generated by text corpus.

def existing: this function returns only the word that are in the frequency dictionary

main logic

* + - 1. The method first checks if the exact word exists by calling self.existing([word]).
      2. If it doesn’t exist, it checks for typo candidates using w.typos().
      3. If still no valid candidates are found, it checks for more complex typo candidates (in the non-fast mode) using w.double\_typos().
      4. If nothing valid is found even after all this, it returns the original word as a fallback.

1. <https://github.com/anassinator/markov-sentence-correction/blob/master/corrector.py>

* Using a bigram model to predict likely word sequences.
* Applying edit distance to measure how close an observed word is to a possible correction.
* Building a trellis graph to track the best word sequences and their probabilities.
* Iteratively increasing error tolerance until it finds the most likely corrected sentence.

It outputs the corrected sentence and additional details like probability and distance if verbose mode is enabled.

1. [https://github.com/ericcornelissen/AutoCorrect- py/blob/master/AutoCorrect/autocorrect.py](https://github.com/ericcornelissen/AutoCorrect- py/blob/master/AutoCorrect/autocorrect.py )

* Dictionary Creation: It stores characters and word relationships (e.g., follows, leads) in a tree.
* Search Algorithms: It has methods to find and suggest corrections for words, including:

Bubble Search: For swapped characters.

Insertion Search: For missing characters.

Replacement Search: For incorrect characters.

Space Search: For splitting words.

* Correction and Learning: It corrects words in text based on stored data and learns new words from input text.
* Feedback: Allows manual feedback to improve suggestions.
* Import/Export: The dictionary can be saved and loaded from a json file for reuse.

Simple Auto Correct

Minimum Edit Distance Algorithm

Given two string s1(target) and s2(source) this algorithm calculates the minimum edit distance to convert source to target

1. Recursion approach

min\_edit(stay,play)

Insert(p)`

Delete(p)

replace(p)

min\_edit(pstay,play)+1

min\_edit(tay,play)+1

min\_edit(ptay,play)+2

Insert(l)`

Delete(s)

replace(s)

min\_edit(plstay,play)+1

min\_edit(ptay,play)+1

min\_edit(pltay,play)+2

…

… … …

1. DP table approach

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | 0 | p | l | a | y |
| 0 | 0 | 1 | 2 | 3 | 4 |
| s | 1 | 2 | 3 | 4 | 5 |
| t | 2 | 3 | 4 | 5 | 6 |
| a | 3 | 4 | 5 | 4 | 4 |
| y | 4 | 5 | 6 | 5 | 4 |

Auto Correct V2

Dataset

First, we have to make frequency dictionary to that we need the English text corpuses.

Used text corpuses:

1. <https://wortschatz.uni-leipzig.de/en/download/English> : (2005-10k)

Clean the data:

1. Data is like this: *12 Search for your new home - if you're for a home in San Diego, CA.*

So, we need to remove the leading number.

* + - re.sub(r'^\d+\s+', '', line): this method replace the leading number of the sentence and any white spaces by “”
    - ^[\W\d]+: Matches one or more non-word characters (\W) or digits (\d) at the beginning of the word.
    - [\W\d]+$: Matches one or more non-word characters or digits at the end of the word.