<u>English Auto-correct in Python</u>

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.
2. 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.

3. 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.

4. https://github.com/ericcornelissen/AutoCorrect-py/blob/master/AutoCorrect/autocorrect.py

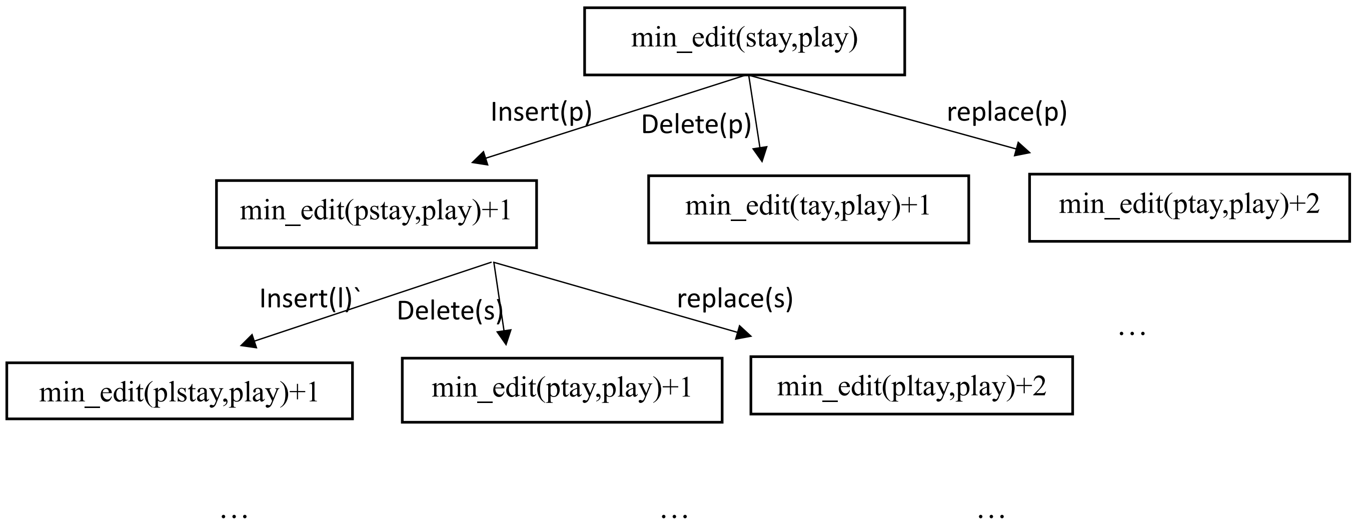
- Dictionary Creation: It stores characters and word relationships (e.g., follows, leads) in a tree.
- Search Algorithms: It has methods to find and suggest corrections for words, including:
  Bubble Search: For swapped characters.
  Insertion Search: For missing characters.
  Replacement Search: For 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



2. 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.

1. Download the Dataset:

- Get the dataset from the Leipzig Corpora Collection, specifically the English version (e.g., 2005-10k). It contains sentences, each prefixed with a number and some additional formatting.

2. Data Cleaning:

- Remove Leading Numbers: Sentences in the dataset have numbers at the start (e.g., "12 Search for your new home..."). We need to remove these numbers, which can be done using regular expressions to strip the number and any following spaces.
- Remove Non-Word Characters: After removing the numbers, clean any unwanted non-word characters (like punctuation marks) or digits that may appear at the start or end of sentences. This ensures we work only with clean, meaningful text.

3. Corpus Preparation:

- Once cleaned, the sentences can be processed further to create a frequency dictionary, where words are counted based on their occurrences in the text. This frequency dictionary will later help in auto-correcting user input by suggesting the most frequent or likely correct word.

3. Generate Possible Edits:

To correct a misspelled word, we generate all possible variations (or "edits") of the word. These edits include:

- Deletions: Removing one letter from the word.
- Transpositions: Swapping adjacent letters.
- Replacements: Changing one letter to another.
- Insertions: Adding a letter at any position.

4. Search for Correct Word in Dictionary:

- Once we generate these possible edits, we search through our frequency dictionary to see if any of these variations exist. The frequency dictionary contains words that are considered correctly spelled and are ranked by how often they appear in the corpus. This helps prioritize likely corrections.

5. Suggest Corrections:

- If one or more possible edits match words in the frequency dictionary, we suggest the most frequent or highest-probability word as the correction. If no direct match is found, we can generate more complex edits (e.g., edits of edits) or suggest no correction.

1. Input Word: User inputs a possibly misspelled word.

2. Generate Edits: Create all variations of the word through deletions, transpositions, replacements, and insertions.

3. Check Dictionary: For each variation, check if it exists in the frequency dictionary.

4. Rank Suggestions: If multiple corrections are found, rank them based on frequency and suggest the most probable one.

5. Output: Display the suggested word to the user as the correction.