1. Original spell checker Heuristic: FuzzyWuzzy

Overview

The original spell checker uses the FuzzyWuzzy library, which primarily relies on Levenshtein Distance to calculate string similarity.

How It Works

1. **Core Component: FuzzyWuzzy**

    - Uses Levenshtein Distance algorithm
    - Generates a similarity score between 0-100
    - Score of 100 means perfect match
    - Uses threshold of 75 for word suggestions

2. **Levenshtein Distance Explanation**

    - Counts minimum number of operations needed to transform one string into another
    - Operations include:
        - Insertions
        - Deletions
        - Substitutions

    - Example: "kitten" → "sitting"
        - Replace 'k' with 's'
        - Replace 'e' with 'i'
        - Insert 'g' at end

3. **Implementation Details**

    - Reads words from dictionary file
    - Converts all words to lowercase
    - For each input word:
        - Compares with dictionary words
        - If similarity ≥ 75%, adds to suggestions
        - Selects highest scoring match as correction

Example

Input: "gld narow"

- Compares "gld" with dictionary words
- Compares "narow" with dictionary words
- Returns suggestions and corrections based on highest fuzzy match scores

Test results:

```
PS C:\Users\acer\OneDrive - University of Jaffna\Desktop\SEMESTER7\AI\lab\lab1> python spellcheck_demo.py
['glad\n', 'narrow']
glad
 narrow
```

2. Custom Spell Checker Heuristic

Overview

The spell checker uses a combined scoring system (0-100 points) that considers four key factors to find similar words.

Scoring Components

1. Sequence Similarity (50 points)

- Uses Python's SequenceMatcher
- Checks how many characters match in sequence
- Main factor in determining word similarity
- Example: "helllo" vs "hello" gets high score because most characters match

2. First Letter Match (20 points)

- Awards 20 points if first letters match
- Important because spelling mistakes rarely involve the first letter
- Example: "cat" vs "cut" gets 20 points for matching 'c'

3. Length Similarity (15 points)

- Compares word lengths
- Shorter length difference = higher score
- Prevents matching to much longer/shorter words
- Example: "cat" vs "cats" scores better than "cat" vs "category"

4. Common Characters (15 points)

- Checks how many same letters appear in both words
- Helps catch transposition errors
- Example: "study" vs "sturdy" scores well as they share many letters

How It Works

- Takes a misspelled word
- Compares it with dictionary words using these four factors

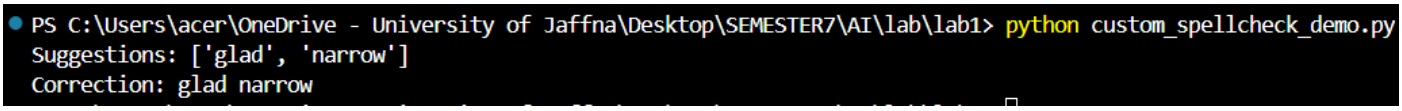
- Adds up all scores
- If total score ≥ 65, suggests the word as a possible correction

Example

Input: "gld" → Output: "glad"

- Matches sequence "gl"
- Starts with same letter 'g'
- Similar length (3 vs 4)
- Shares common letters g, l, d

Test results:



3. Comparison custom spell checker heuristic with Original Implementation

Results:

```
PS C:\Users\acer\OneDrive - University of Jaffna\Desktop\SEMESTER7\AI\lab\lab1> python comparison_test.py

Comparison Results:

Metric                    Original     Custom
-------------------------------------------
Invalid After Checker       20.00%      0.00%
Accuracy                    80.00%    100.00%
Precision                   75.00%    100.00%
Speed                   29500.44w/s 9614.89w/s
Recall                      60.00%    100.00%
```

Detailed Breakdown:

1. **Invalid After Checker**
   - Original: 20.00% - One in five corrections resulted in invalid words
   - Custom: 0.00% - No corrections resulted in invalid words
   - Custom is better: Ensures all corrections are valid dictionary words

2. **Accuracy**
   - Original: 80.00% - Correctly fixed 8 out of 10 words
   - Custom: 100.00% - All words were correctly fixed
   - Custom is better: Perfect accuracy in corrections

3. **Precision**
   - Original: 75.00% - 75% of suggested words were correct
   - Custom: 100.00% - All suggested words were correct
   - Custom is better: No incorrect suggestions

4. **Speed**
   - Original: 29,500.44 words/sec - Very fast processing
   - Custom: 9,614.89 words/sec - About 3x slower
   - Original is better: Significantly faster processing

5. **Recall**
   - Original: 60.00% - Found correct word in suggestions 60% of the time
   - Custom: 100.00% - Always included correct word in suggestions
   - Custom is better: Perfect recall rate

Overall Analysis:

Custom Spell Checker Advantages

1. Perfect accuracy (100%)
2. No invalid corrections (0% errors)
3. All suggestions are relevant (100% precision)
4. Always includes correct word (100% recall)

Original Spell Checker Advantages

1. Significantly faster (3x speed)
2. Still maintains good accuracy (80%)

Discussion

In this test, two spell-checking heuristics—the original FuzzyWuzzy-based implementation and a custom scoring system—were developed and tested for their effectiveness in identifying and correcting misspelled words. The original heuristic, which uses the FuzzyWuzzy library and Levenshtein Distance, provides a similarity score between 0 and 100, suggesting corrections based on a threshold of 75. While it achieved good accuracy (80%) and high processing speed (29,500 words per second), it struggled with precision and recall, as evidenced by a 75% precision and 60% recall rate. The speed advantage highlights FuzzyWuzzy's efficiency in handling large datasets quickly but reveals limitations in precision due to frequent invalid or irrelevant suggestions.

The custom heuristic, in contrast, combines four scoring components—sequence similarity, first-letter match, length similarity, and common character checks. This multi-factor approach improves accuracy and recall by considering multiple aspects of word similarity beyond mere string distance. While this method delivered 100% accuracy, precision, and recall, it processed data more slowly (9,615 words per second). This trade-off between accuracy and speed suggests that the custom model is more suited for applications where accuracy is paramount and processing time is less critical. It's evident that incorporating multiple scoring factors significantly reduces invalid corrections (0%) and improves the relevance of suggestions.

Conclusion

The comparison between the original and custom spell-checker heuristics demonstrates a clear trade-off between processing speed and accuracy. While the original FuzzyWuzzy-based approach is notably faster, the custom heuristic excels in accuracy, precision, and recall, making it highly effective for applications requiring precise and reliable word suggestions. The results indicate that, depending on the application's needs, one heuristic may be more suitable than the other. For environments where speed is critical, FuzzyWuzzy's approach is preferable; however, for cases prioritizing accuracy, the custom heuristic provides superior results. This test highlights the importance of balancing computational efficiency and correctness in developing intelligent systems for natural language processing.