Autocorrect Feature Using NLP in Python

Introduction to Autocorrect

Autocorrect is an essential feature embedded in modern smartphones and text editors, correcting spelling errors as users type.

While it might seem simple, the mechanism behind autocorrect is a sophisticated application of Natural Language Processing (NLP).

In this article, we will delve into how to build a basic autocorrect feature using Python, leveraging concepts from NLP.

We'll use a text corpus from a book as our dataset, but the principles and methods can be applied to any text data.

This is a simplified version compared to the autocorrect systems in smartphones, which use vast amounts of data and complex algorithms.

How Autocorrect Works

At its core, an autocorrect system involves the following steps:

1. Identify Misspelled Words: The first step is to detect if a word is misspelled. This can be done by comparing the word against a known dictionary.

If the word is not found in the dictionary, it is flagged for correction.

2. Generate Candidate Words: Once a word is identified as misspelled, we generate a list of

possible correct words.

This is done using various edit operations (insert, delete, switch, replace) that transform the incorrect word into potential correct words.

3. Filter Candidates: After generating candidate words, we filter them by checking which ones exist in our known dictionary.

This step helps in eliminating nonsensical words.

4. Rank Candidates by Probability: Finally, we rank the remaining candidates based on their likelihood,

which can be determined by the frequency of each word in the corpus.

Python Implementation of Autocorrect

Let's walk through the Python code to implement the autocorrect feature.

We'll start by installing the necessary libraries.

```
pip install textdistance
```

Step 1: Import Required Libraries

```
import re
import pandas as pd
import numpy as np
import textdistance
from collections import Counter
```

Step 2: Load and Preprocess the Text Data

```
# Load the text data
with open('auto.txt', 'r') as f:
    text_data = f.read().lower()

# Extract words using regular expressions
words = re.findall(r'\w+', text_data)

# Create a set of unique words (vocabulary)
vocabulary = set(words)
print(f"Total unique words in the corpus: {len(vocabulary)}")
```

Step 3: Build Word Frequency Distribution

```
# Calculate word frequency using Counter
word_freq = Counter(words)

# Display the most common words
print("Most common words:", word_freq.most_common(10))
```

Step 4: Calculate Word Probabilities

```
# Calculate total number of words
total_words = sum(word_freq.values())
```

```
# Calculate probability of each word
word_probs = {word: freq / total_words for word, freq in word_freq.items()}
```

Step 5: Define the Autocorrect Function

```
def autocorrect(input_word):
    input_word = input_word.lower()

    # Check if the word is already correct
    if input_word in vocabulary:
        return f"'{input_word}' is already correct."

    # Calculate similarity between input word and words in the vocabulary
        similarities = [1 - textdistance.Jaccard(qval=2).distance(word, input_word) for word in word_freq.keys()]

    # Create a DataFrame with words, probabilities, and similarities
    df = pd.DataFrame({'Word': list(word_freq.keys()),
                       'Probability': list(word_probs.values()),
                       'Similarity': similarities})

    # Sort the DataFrame by similarity and probability
    df = df.sort_values(by=['Similarity', 'Probability'], ascending=False)

    # Return the top 5 suggestions
```

```
    return df.head(5)
```

Step 6: Test the Autocorrect Function

```
# Test the autocorrect function
suggestions = autocorrect('neverteless')
print(suggestions)
```

Sample Output:

Word	Probability	Similarity
nevertheless	0.000229	0.75
never	0.000942	0.4
boneless	0.000014	0.416667
elevates	0.000005	0.416667
level	0.000110	0.4

Conclusion:

In this tutorial, we built a simple autocorrect feature using Python and NLP. While this implementation is basic,
it provides a foundation for more advanced autocorrect systems. By using larger corpora, additional NLP techniques,
and machine learning models, you can further enhance the accuracy and efficiency of such a system.

This autocorrect model demonstrates how NLP can be applied to practical tasks, showcasing the importance of understanding and processing language data.