Fuzzy String Matching – A Hands-on Guide

BEGINNER LIBRARIES PYTHON PYTHON TEXT

Fuzzy string matching, or fuzzy matching, is a technique used to find strings that partially match a given string rather than requiring an exact match. It is particularly useful when users misspell words or enter partial terms, as seen in search engines. The underlying algorithm measures the similarity between two strings using a distance metric called 'edit distance,' which calculates the minimum changes needed to convert one string into another. Common edit distance types include Levenshtein, Hamming, and Jaro distances. This article explores fuzzy matching in Python, highlighting libraries like FuzzyWuzzy that simplify identifying approximate matches, with practical examples to improve programming efficiency and skills.

This article was published as a part of the Data Science Blogathon

Table of contents

What is Fuzzy String Matching?

Fuzzy string matching is like finding similar things, even if they aren't exactly the same. It's used to find misspelled words, similar names, or even close matches in DNA sequences. It's like saying, "These things are close enough!

Let us illustrate how the Levenshtein distance is calculated.

Example 1:

```
String 1 = 'Put'
```

```
String 2 = ’Pat’
```

Levenshtein distance would be 1 as we can convert string 1 to string 2 by replacing ‘u’ with ‘a’.

Example 2:

```
String 1 = ‘Sun’
```

```
String 2 = ‘Saturn’
```

Levenshtein distance would be 3 as we can convert string 1 to string 2 by 3 insertions – ‘a’, ’t’ and ‘r’.

Fuzzy String Matching in Python:

Comparing Strings in Python

To compare two strings in **python**, we can run the following code:

```
Str1 = "Back" Str2 = "Book" Result = Str1 == Str2 print(Result)
```

The above code will give an output as ‘False’ as the two strings are not the same.

Levenshtein distance in Python

Levenshtein distance in Python using the ‘Levenshtein’ python package.

```
import Levenshtein as lev Str1 = "Back" Str2 = "Book" lev.distance(Str1.lower(),Str2.lower())
```

The above code will give an output of 2 we can convert string 1 to string 2 by 2 replacements.

FuzzyWuzzy in Python

FuzzyWuzzy is a python package that can be used for string matching. We can run the following command to install the package –

```
pip install fuzzywuzzy
```

Just like the Levenshtein package, FuzzyWuzzy has a ratio function that calculates the standard Levenshtein distance similarity ratio between two sequences.

```
from fuzzywuzzy import fuzz Str1 = "Back" Str2 = "Book" Ratio = fuzz.ratio(Str1.lower(),Str2.lower())
print(Ratio)
```

The output of the following code gives 50 as the Levehshtein ratio is calculated by dividing the Levenshtein distance by the maximum of the length of the string1 and string 2.

Let us calculate the ratio for another set of strings.

```
from fuzzywuzzy import fuzz Str1 = "My name is Ali" Str2 = "Ali is my name" Ratio = fuzz.ratio(Str1.lower(),Str2.lower()) print(Ratio)
```

The output of the code gives 50 indicating that even though the words are the same, the order of the words matters while calculating the ratio.

Partial Ratio using FuzzyWuzzy

The partial ratio helps us to perform substring matching. This takes the shortest string and compares it with all the substrings of the same length.

```
Str1 = "My name is Ali" Str2 = "My name is Ali Abdaal" print(fuzz.partial_ratio(Str1.lower(),Str2.lower()))
```

The output of the code gives 100 as partial_ratio() just checks if either string is a substring of the other.

This ratio could be very useful if, for example, we are trying to match a person's name between two datasets. In the first dataset, the string has the person's first and last name, and in the second dataset, the string has the person's first, middle, and last name. The ratio would be 100 because the first string is a substring in the second string.

Checkout this article about the machine learning algorithms

Token Sort Ratio using FuzzyWuzzy

In token sort ratio, a method used in fuzzy string matching, the strings are tokenized and pre-processed by converting to lower case and getting rid of punctuation. The strings are then sorted alphabetically and joined together. Post this, the Levenshtein distance similarity ratio is calculated between the strings.

```
Str1 = "My name is Ali" Str2 = "Ali is my name" print(fuzz.token_sort_ratio(Str1,Str2))
```

The output of the code gives 100 as the token sort ratio is found after sorting the strings alphabetically and hence the original order of words doesn't matter.

Token Set Ratio using FuzzyWuzzy

Token set ratio performs a set operation that takes out the common tokens instead of just tokenizing the strings, sorting, and then pasting the tokens back together. Extra or same repeated words do not matter.

```
Str1 = "My name is Ali" Str2 = "Ali is my name name" print(fuzz.token_sort_ratio(Str1,Str2)) print(fuzz.token_set_ratio(Str1,Str2))
```

The output of the token sort ratio comes to be 85 while that of the token set ratio comes to be 100 as the token set ratio doesn't take into account the repeated words.

Let us illustrate another example for the token set ratio for a deeper explanation.

```
Str_A = 'Read the sentence - My name is Ali' Str_B = 'My name is Ali' ratio = fuzz.token_set_ratio(Str_A, Str_B) print(ratio)
```

The output of the above code gives us 100. This is because, under the hood, the token set ratio has a more flexible approach. After it takes out the common strings ('My name is Ali'), it finds out the fuzz ratio for the following pairs and then returns the maximum value amongst the three:

- common string and the common string with the remainder of string one
- common string and the common string with the remainder of string two
- common string with the remainder of one and common string with the remainder of two

Process Module using FuzzyWuzzy

If we have a list of strings and we want to find the closest matching string from the list with a given string, we can leverage the 'process' module.

```
from fuzzywuzzy import process query = 'My name is Ali' choices = ['My name Ali', 'My name is Ali', 'My Ali'] # Get a list of matches ordered by score, default limit to 5 process.extract(query, choices)
```

```
[('My name is Ali', 100), ('My name Ali', 95), ('My Ali', 86)]
```

If we want to extract out the top match, we can run the following code:

```
process.extractOne(query, choices)
```

```
('My name is Ali', 100)
```

Conclusion

Fuzzy string matching is a powerful technique for comparing and identifying similar text, even when there are variations or errors. In Python, libraries like FuzzyWuzzy and Levenshtein provide efficient ways to measure string similarity using methods such as Levenshtein distance, Partial Ratio, Token Sort Ratio, and Token Set Ratio. The Process module in FuzzyWuzzy further enhances matching by ranking multiple strings based on similarity. These techniques are widely used in data cleaning, search engines, and NLP applications. Understanding and applying fuzzy string matching can significantly improve text analysis tasks by handling inconsistencies in real-world data effectively.

Frequently Asked Questions

Q1. What is fuzzy match in regex Python?

A. Fuzzy matching in regex Python is a technique used to match patterns in text data that are similar or partially match the target pattern. Fuzzy matching allows for variations in spelling, punctuation, and spacing in the text data. In Python, fuzzy matching can be achieved by using regular expressions and

string distance functions like Levenshtein distance, Jaro-Winkler distance, or fuzzywuzzy library. The fuzzywuzzy library provides a set of functions for fuzzy string matching and can be used to find the best match among a set of possible matches.

Q2. What is fuzzy matching vs stemming?

A. Fuzzy matching and stemming are both techniques used in natural language processing, but they serve different purposes. Fuzzy matching allows for variations in spelling, punctuation, and spacing in the text data, while stemming is used to reduce words to their root or base form. Fuzzy matching is useful for matching similar or partially matching patterns, while stemming is useful for grouping words with the same root or meaning.

Article Url - https://www.analyticsvidhya.com/blog/2021/07/fuzzy-string-matching-a-hands-on-guide/

Nibedita Dutta

Nibedita completed her master's in Chemical Engineering from IIT Kharagpur in 2014 and is currently working as a Senior Data Scientist. In her current capacity, she works on building intelligent ML-based solutions to improve business processes.