Fuzzy matching

Dopasowywanie rozmyte

Plan prezentacji

Podstawy porównywania tekstu
Slajd 3 - 5

Dystans Levenshtein’a
Slajd 6 - 10

FuzzyWuzzy
Slajd 11 - 14

Proste dopasowania

```
str1 = "This is a string."
str2 = "This is a string."

if str1 == str2:
    print("strings match")
```

```
strings match
```

rys 1. Porównanie dwóch tekstów, które są w 100% identyczne

Proste dopasowania

```
str1 = "This is a string."
str2 = "this Is a sTriNg."

if str1.lower() == str2.lower():
    print("strings match")
```

```
strings match
```

rys 2. Porównanie dwóch tekstów, które składają się z tych samych liter, ale o różnych wielkościach

Kłopoty z dopasowaniem

W tym przypadku samo *lower* nie pomoże. Różnica jest w znakach, a ich wielkości.

```
str1 = "This is a string."
str2 = "That is a string"

if str1.lower() == str2.lower():
    print("strings match")
else:
    print("Strings don't match")
```

```
Strings don't match
```

rys 3. Przykład tekstów, które ciężko porównać prostymi metodami

Dystans Levenshtein’a

Dystans Levenshtein’a - metryka, która wyznacza minimalną ilość edycji, która jest potrzeba, aby zmienić jeden tekst w drugi. Do edycji zaliczamy: usuwanie znaków, wstawianie znaków i podmiana znaku na inny.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise}, \end{cases}$$

rys 4. Wzór na dystans Levenshtein’a między dwoma tekstami a i b o długości |a| i |b|

Dystans Levenshtein’a - omówienie wzoru

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise}, \end{cases}$$

1. Jeżeli tekst *b* ma długość 0. To potrzeba |*a*| edycji by zamienić tekst *b* na *a*.
2. Jeżeli tekst *a* ma długość 0. To potrzeba |*b*| edycji by zamienić tekst *a* na *b*.
3. Jeżeli pierwszy znak w danym tekście jest ten sam, to wywołujemy funkcję rekurencyjnie na reszcie tekstu *a[1:|a|:1]* i *b[1:|a|:1]* (*funkcja tail wycina pierwszy znak).*
4. W przeciwnym wypadku szukamy takich edycji, które zwrócą najmniejszą ilość edycji.
   a. lev(tail(a), b) - to usunięcie znaku,
   b. lev(a, tail(b)) - to dodanie znaku,
   c. lev(tail(a), tail(b)) - to podmiana znaku

Wartość dopasowania

Wartość dopasowania od 0 do 1 wyznacza się za pomocą podanego poniżej stosunku.

$$\frac{|a|+|b|-dystans\ Levenshteina}{|a|+|b|}$$

rys 5. Wzór na wartość podobieństwa dwóch tekstów

Dystans Levenshtein’a - implementacja

```python
import numpy as np

def levenshtein(s1, s2):
    rows = len(s1)+1
    cols = len(s2)+1
    distance = np.zeros((rows,cols),dtype = int)

    for i in range(1, rows):
        for k in range(1,cols):
            distance[i][0] = i
            distance[0][k] = k

    for i in range(1, rows):
        for j in range(1, cols):
            cost = 1

            if s1[i-1] == s2[j-1]:
                cost = 0

            distance[i][j] = min(distance[i-1][j] + 1,
                                 distance[i][j-1] + 1,
                                 distance[i-1][j-1] + cost)

    Ratio = ((len(s1)+len(s2)) - distance[i][j]) / (len(s1)+len(s2))

    print(Ratio)
    print("The strings are {} edits away".format(distance[i][j]))
    print(distance)
```

rys 6. Implementacja algorytmu dystansu Levenshtein’a

```python
str1 = "This is a string."
str2 = "That is a string."

levenshtein(str1, str2)
```

```
Ratio: 0.9411764705882353
The strings are 2 edits away
[[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17]
 [ 1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
 [ 2  1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
 [ 3  2  1  1  2  3  3  4  5  6  7  8  9 10 11 12 13 14]
 [ 4  3  2  2  2  3  4  3  4  5  6  7  8  9 10 11 12 13]
 [ 5  4  3  3  3  2  3  4  3  4  5  6  7  8  9 10 11 12]
 [ 6  5  4  4  4  3  2  3  4  4  5  6  7  8  8  9 10 11]
 [ 7  6  5  5  5  4  3  2  3  4  5  5  6  7  8  9 10 11]
 [ 8  7  6  6  6  5  4  3  2  3  4  5  6  7  8  9 10 11]
 [ 9  8  7  6  7  6  5  4  3  2  3  4  5  6  7  8  9 10]
 [10  9  8  7  7  7  6  5  4  3  2  3  4  5  6  7  8  9]
 [11 10  9  8  8  8  7  6  5  4  3  2  3  4  5  6  7  8]
 [12 11 10  9  8  9  8  7  6  5  4  3  2  3  4  5  6  7]
 [13 12 11 10  9  9  9  8  7  6  5  4  3  2  3  4  5  6]
 [14 13 12 11 10 10  9  9  8  7  6  5  4  3  2  3  4  5]
 [15 14 13 12 11 11 10 10  9  8  7  6  5  4  3  2  3  4]
 [16 15 14 13 12 12 11 11 10  9  8  7  6  5  4  3  2  3]
 [17 16 15 14 13 13 12 12 11 10  9  8  7  6  5  4  3  2]]
```

rys 7. Przykładowe uruchomienie

Preprocessing

Preprocessing polega na przetworzeniu tekstów w taki sposób, aby nie zmienić ich znaczenia, ale aby uprościć je jak najbardziej.

- Zamiana małych znaków na duże,
- Usunięcie znaków interpunkcyjnych,
- Powtarzające się znaki białe

FuzzyWuzzy

FuzzyWuzzy - biblioteka zawierająca funkcje ułatwiające dopasowywanie rozmyte. Zawiera funkcję obliczającą dystans Levenshtein'a i wiele innych funkcji m.in

Ratio - dystans Levenshtein'a. Przy czym koszt dla podmiany liter równy jest 2.

```
In [5]:
1 from fuzzywuzzy import fuzz
2
3 str1 = "This is a string."
4 str2 = "That is a string."
5
6 fuzz.ratio(str1, str2) / 100

Out[5]: 0.88
```

rys 8. Przykładowe funkcji ratio z pakietu fuzzywuzzy

FuzzyWuzzy

Partial_ratio - obliczanie dystansu na mniejszych fragmentach tekstu. Wybierany jest krótszy tekst *a*, a dłuższy rozbijany jest na krótsze teksty, z którymi jest porównywany tekst *a*. Wartość zwracana to wartość podobieństwa z najlepszego dopasowania.

```
1  from fuzzywuzzy import fuzz
2
3  str1 = "Boston dynamic builds robots"
4  str2 = "roboty"
5
6  fuzz.partial_ratio(str1, str2) / 100
```

```
0.83
```

rys 9. Przykładowe funkcji partial_ratio z pakietu fuzzywuzzy

FuzzyWuzzy

Token_sort_ratio - tonizuje tekst na słowa, sortuje słowa tekstu alfabetycznie i łączy je razem. Przydatne gdy teksty mają podobną długość, ale słowa są zamienione.

```
1  from fuzzywuzzy import fuzz
2
3  str1 = "Nasa landed a man on the moon"
4  str2 = "A man landed on the moon due to Nasa"
5
6  fuzz.token_sort_ratio(str1, str2) / 100
```

```
0.89
```

rys 10. Przykładowe funkcji token_sort_ratio z pakietu fuzzywuzzy

FuzzyWuzzy

Token_set_ratio - działa podobnie do funkcji poprzedniej, ale wycina powtarzające się słowa co pozwala porównywać teksty o różnej długości.

```
1  from fuzzywuzzy import fuzz
2
3  str1 = "Todays match is the first match between Lakser and Galaxis"
4  str2 = "Lakser v Galaxis"
5
6  fuzz.token_set_ratio(str1, str2) / 100
```

```
0.93
```

rys 10. Przykładowe funkcji token_set_ratio z pakietu fuzzywuzzy

Źródła

https://pypi.org/project/fuzzywuzzy - biblioteka fuzzy wuzzy
https://en.wikipedia.org/wiki/Levenshtein_distance - dystans Levenshtein'a
https://www.datacamp.com/tutorial/fuzzy-string-python - dopasowywanie rozmyte

Dziękujemy

Jakub Usyk
Antoni Zub
Amadeusz Waląg