Wk4_SLP1_Advanced text processing with FuzzyWuzzy

March 25, 2021

[ ]:
```
student_name = "Nikki Fitzherbert"
student_id = "13848336"
```

Steps 1 and 2 Download and save the given csv file and install the FuzzyWuzzy library.

Step 3 Import pandas and read in the saved csv file.

[2]:
```
import pandas as pd

df = pd.read_csv('Wk4_SLP1_room_type.csv')
df.head(10)
```

[2]:

	Expedia	Booking.com
0	Deluxe Room, 1 King Bed	Deluxe King Room
1	Standard Room, 1 King Bed, Accessible	Standard King Roll-in Shower Accessible
2	Grand Corner King Room, 1 King Bed	Grand Corner King Room
3	Suite, 1 King Bed (Parlor)	King Parlor Suite
4	High-Floor Premium Room, 1 King Bed	High-Floor Premium King Room
5	Traditional Double Room, 2 Double Beds	Double Room with Two Double Beds
6	Room, 1 King Bed, Accessible	King Room - Disability Access
7	Deluxe Room, 1 King Bed	Deluxe King Room
8	Deluxe Room	Deluxe Room (Non Refundable)
9	Room, 2 Double Beds (19th to 25th Floors)	Two Double Beds - Location Room (19th to 25th …

Steps 4 & 5 Import fuzz from FuzzyWuzzy and use the 'ratio' approach to compare string similarity. The results indicate that this approach is far too sensitive to minor differences in word

order, missing or extra words and other similar issues.

[1]:
```
from fuzzywuzzy import fuzz
```

[7]:
```
string1 = df.Expedia[0]
string2 = df['Booking.com'][0]

score = fuzz.ratio(string1, string2)
print(score)

#-----#
string1 = df.Expedia[5]
string2 = df['Booking.com'][5]

score = fuzz.ratio(string1, string2)
print(score)

#-----#
string1 = df.Expedia[9]
string2 = df['Booking.com'][9]

score = fuzz.ratio(string1, string2)
print(score)
```

```
62
69
74
```

Step 6 Use the 'partial ratio' approach to compare string similarity. It turns out that there is little difference/improvement to the results.

[8]:
```
string1 = df.Expedia[0]
string2 = df['Booking.com'][0]

score = fuzz.partial_ratio(string1, string2)
print(score)

#-----#
string1 = df.Expedia[5]
string2 = df['Booking.com'][5]

score = fuzz.partial_ratio(string1, string2)
print(score)

#-----#
string1 = df.Expedia[9]
string2 = df['Booking.com'][9]
```

```
score = fuzz.partial_ratio(string1, string2)
print(score)
```

```
69
83
63
```

Step 7 The 'token sort ratio' ignores word order when comparing strings.

[9]:
```
string1 = df.Expedia[0]
string2 = df['Booking.com'][0]

score = fuzz.token_sort_ratio(string1, string2)
print(score)

#-----#
string1 = df.Expedia[5]
string2 = df['Booking.com'][5]

score = fuzz.token_sort_ratio(string1, string2)
print(score)

#-----#
string1 = df.Expedia[9]
string2 = df['Booking.com'][9]

score = fuzz.token_sort_ratio(string1, string2)
print(score)
```

```
84
78
83
```

Step 8 And the 'token set ratio' ignores duplicated words. It's similar to the previous approach, but somewhat more flexible. The results indicate that this approach is the most suitable for this dataset.

[10]:
```
string1 = df.Expedia[0]
string2 = df['Booking.com'][0]

score = fuzz.token_set_ratio(string1, string2)
print(score)

#-----#
string1 = df.Expedia[5]
string2 = df['Booking.com'][5]

score = fuzz.token_set_ratio(string1, string2)
```

```
print(score)

#-----#
string1 = df.Expedia[9]
string2 = df['Booking.com'][9]

score = fuzz.token_set_ratio(string1, string2)
print(score)
```

```
100
78
97
```

Step 9 Construct a fucntion using the 'token_set_ratio' approach and apply it to the entire dataset. Over 90% of the pairs exceeded the specified minimum match score of 70.

[12]:
```
def get_ratio(row):
    name = row['Expedia']
    name2 = row['Booking.com']
    return fuzz.token_set_ratio(name, name2)

len(df[df.apply(get_ratio, axis=1) > 70]) / len(df) # set a minimum match score␣
↪of 70.
```

[12]:
```
0.9029126213592233
```