Rozmyte dopasowanie do wzorca

Julia Kamuda

Dominika Hojniak

Kinga Furmanek

1. Abstrakt

Dopasowanie danych pomiędzy rożnymi zbiorami to częsty problem podczas analizy danych, szczególnie gdy nazwy zawierają drobne literówki lub inne mniejsze błędy. W tym raporcie przedstawiono proces i narzędzie użyte do automatyzacji dopasowania danych do wzorca. Użyto przykładu gdy nazwy chorób w dwóch zestawach danych nie są identyczne. Aby rozwiązać problem użyto technik fuzzy string matching z wykorzystaniem biblioteki TheFuzz w Pythonie.

2. Wstęp

2.1 Cel

Celem projektu było przedstawienie skutecznej metody integracji dwóch zestawów danych, zawierających informacje o chorobach, gdzie dane nie mogły zostać połączone z powodu błędów w pisowni nazw. Projekt miał na celu zademonstrowanie wykorzystania metod fuzzy string matching przy użyciu biblioteki TheFuzz w Pythonie jako praktycznego i łatwego w implementacji rozwiązania dla tego typu problemów.

2.2 Zakres

Projekt obejmował:

- zbadanie i wyjaśnienie mechanizmów fuzzy string matching,
- implementację dopasowania danych za pomocą biblioteki **TheFuzz** w Pythonie,
- integrację danych w jedną, spójną strukturę aby udowodnić działanie metod,
- prezentację wyników w formie końcowego zestawu danych.

2.3 Metodyka

Do realizacji projektu wykorzystano:

- **Python**: Język programowania, który posiada prostą składnie i bogaty zbiór bibliotek,
- **Bibliotekę TheFuzz**: biblioteka, wykorzystująca algorytmy porównywania tekstów, takie jak Levenshtein distance,
- **Pandas**: biblioteka służąca do wygodnego przetwarzania danych i analizy danych w tabelach.

3. Teoria Dopasowania Fuzzy

3.1 Czym jest fuzzy string matching?

Fuzzy string matching jest techniką używaną do porównywania tekstów typu string, które są podobne, lecz nie muszą być jednakowe. Jest ona szczególnie przydatna przy danych tekstowych mogących zawierać wszelkie literówki, błędy ortograficzne, skróty czy inną kolejność słów. Przykładowo wpisanie w wyszukiwarkę internetową słowa z błędem nie spowoduje, że nie wyświetlą się żadne wyniki. Z wykorzystaniem tej techniki, jeśli wpisana fraza wykazuje pewne podobieństwo co do docelowo szukanej, to wyniki i tak się wyświetlą. Istnieją różne algorytmy, na który oparte jest fuzzy matching. Są nimi: Odległość Levenshteina, Cosine similarity, Odległość Hamminga, Algorytm n-gram, Algorytm Bitap, Algorytm BK Tree. Stosują one różne podejścia wykorzystywane potem przy znajdywaniu podobieństwa w fuzzy matching. Biblioteka TheFuzz (wcześniej nazywana FuzzyWuzzy), użyta w tym projekcie, jest jedną z najpopularniejszych wykorzystywanych w fuzzy matchingu. Wykorzystuje ona Odległość Levenshteina jako źródło odnajdywania podobieństw wyrazów.

3.2 Algorytm Levenshteina

Odległość Levenshteina, inaczej edycyjna, jest opracowaną przez Władimira Lewensztejna miarą odmienności skończonych ciągów znaków. Służy ona do obliczenia minimalnej liczby edycji jakie trzeba wykonać, by zmienić jeden napis na drugi. Poniżej znajduje się formalna matematyczna definicja:

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & ,gdy\ \min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} \end{cases}$$

, gdzie $1_{(a_i \neq b_j)}$ wynosi 0 gdy a = b.

Przypadki obliczanych minimum oznaczają kolejno usunięcie, wstawienie i zamianę nowego znaku w napisie. Do przeprowadzenia obliczeń wykorzystywana jest macierz, w której każda komórka odpowiada kosztowi przekształcenia prefiksu jednego ciągu w prefiks drugiego ciągu. Kolejność operacji jest następująca: najpierw inicjalizowana jest macierz poprzez ustawienie kosztów przekształcenia pustego ciągu znaków w rosnące fragmenty drugiego ciągu znaków w pierwszym wierszu oraz kosztów przekształcenia pierwszego ciągu znaków w pusty ciąg w pierwszej kolumnie.

Następnie, wypełniając po kolei macierz, w każdej komórce idąc wiersz po wierszu, kolumna po kolumnie wybierany jest najniższy koszt spośród możliwych operacji edycyjnych, w oparciu o obliczone wcześniej sąsiadujące komórki. Wynik, który stanowi szukaną odległość Levenshteina znajduje się w prawym dolnym rogu macierzy.

Poniżej znajduje się przykładowa transformacja ciągu znaków „psy” na ciąg znaków „koty”.

	""	k	o	t	y
""	0	1	2	3	4
p	1				
s	2				
y	3				

➡

	""	k	o	t	y
""	0	1	2	3	4
p	1	1	2		
s	2				
y	3				

➡

	""	k	o	t	y
""	0	1	2	3	4
p	1	1	2	3	4
s	2	2	2	3	4
y	3	3	3	3	3

Odległość przekształcenia zaczynając od słowa „psy”, a kończąc na słowie „koty” wynosi 3.

3.3 Porównywanie ciągów tekstowych

TheFuzz oferuje kilka metod, dzięki którym z łatwością możemy wyliczyć podobieństwo między ciągami tekstowymi:

- **ratio**: prosty współczynnik prawdopodobieństwa, oblicza odległość edycji na podstawie kolejności znaków w porównywanych słowach,

- **partial_ratio**: współczynnik częściowy, porównuje krótszy ciąg tekstowy z podciągiem dłuższego ciągu, szukając tym samym najlepiej dopasowanego fragmentu,
- **token_sort_ratio**: oblicza podobieństwo ignorując kolejność słów; dzieli słowa na tokeny,
- **token_set_ratio**: podobnie jak poprzednik, ignoruje kolejność słów, lecz usuwa wspólne tokeny przed porównaniem.

Process to narzędzie które porównuje ciąg znaków do kolekcji tekstów i zwraca listę w kolejności od najlepszego dopasowania wraz z oceną podobieństwa. Wartość podobieństwa obliczana jest za pomocą metod z biblioteki ***thefuzz***, a domyślnie wykorzystywana jest metoda **ratio**.

4. Rozwiązanie Problemu

4.1 Dane wejściowe

Zestaw danych obejmuje dwa słowniki (pary klucz-wartość). Pierwszy z nich przechowuje dane o chorobach i objawach:

```
dict_one = {
    "disease": ["Cukrzyca", "Nadciśnienie", "Astma" ,"Choroba Wieńcowa", "Przewlekła Choroba Nerek"],
    "symptoms": ["Częste oddawanie moczu, pragnienie", "Ból głowy, zawroty głowy",
                 "Świszczący oddech, duszność", "Ból w klatce piersiowej, zmęczenie", "Obrzęk, zmęczenie"]
}
```

Drugi zawiera nazwy chorób (z błędami) oraz metody leczenia:

```
dict_two = {
    "disease": ["Ckrzy-ca", "Nadiśniene", "Ath-ma", "Wieńcowa Choroba", "Przeleła Choroba Nerk"],
    "treatment": ["Insulina, zmiana stylu życia", "Leki przeciwnadciśnieniowe", "Inhalatory",
                  "Angioplastyka, leki", "Dializy, leki"]
}
```

4.2 Implementacja i wyniki

Początkowo obliczono podobieństwo ciągów z nazwami chorób, wykorzystując każdą z metod.

```
from thefuzz import fuzz
import pandas as pd

dict_one = {
    "disease": ["Cukrzyca", "Nadciśnienie", "Astma", "Choroba Wieńcowa", "Przewlekła Choroba Nerek"]
}

dict_two = {
    "disease": ["Ckrzy-ca", "Nadiśniene", "Ath-ma", "Wieńcowa Choroba", "Przeleła Choroba Nerk"]
}

results = []
for disease_one, disease_two in zip(dict_one["disease"], dict_two["disease"]):
    results.append({
        "disease_one": disease_one,
        "disease_two": disease_two,
        "ratio": fuzz.ratio(disease_one, disease_two),
        "partial_ratio": fuzz.partial_ratio(disease_one, disease_two),
        "token_sort_ratio": fuzz.token_sort_ratio(disease_one, disease_two),
        "token_set_ratio": fuzz.token_set_ratio(disease_one, disease_two),
    })

df_results = pd.DataFrame(results)

print(df_results)
```

Wyniki:

Nazwa choroby	Nazwa choroby z błędami	Ratio	Partial Ratio	Token Sort Ratio	Token Set Ratio
Cukrzyca	Ckrzy-ca	88	88	62	62
Nadciśnienie	Nadiśniene	91	90	91	91
Astma	Ath-ma	73	60	73	73
Choroba Wieńcowa	Wieńcowa Choroba	50	67	100	100
Przewlekła Choroba Nerek	Przeleła Choroba Nerk	93	90	93	93

Analizując powyższe wyniki, **token_set_ratio** jak i **token_sort_ratio** wypadły najlepiej. Obie metody radzą sobie dobrze w przypadkach, gdzie kolejność słów jest różna, pojawiają się literówki i nadmiarowe ciągi tekstowe. Do dalszej implementacji wybrano **token_set_ratio.**

Ostateczna integracja obydwu zbiorów danych:

```
import pandas as pd
from thefuzz import process, fuzz

dict_one = {
    "disease": ["Cukrzyca", "Nadciśnienie", "Astma" ,"Choroba Wieńcowa", "Przewlekła Choroba Nerek"],
    "symptoms": ["Częste oddawanie moczu, pragnienie", "Ból głowy, zawroty głowy",
                 "Świszczący oddech, duszność", "Ból w klatce piersiowej, zmęczenie", "Obrzęk, zmęczenie"]
}

dict_two = {
    "disease": ["Ckrzy-ca", "Nadiśniene", "Ath-ma", "Wieńcowa Choroba", "Przeleła Choroba Nerk"],
    "treatment": ["Insulina, zmiana stylu życia", "Leki przeciwnadciśnieniowe", "Inhalatory",
                  "Angioplastyka, leki", "Dializy, leki"]
}

existing_data = pd.DataFrame(dict_one)
exported_data = pd.DataFrame(dict_two)

exported_data["disease"] = exported_data["disease"].apply(
  lambda x: process.extractOne(x, existing_data["disease"], scorer=fuzz.token_set_ratio)[0]
)

data = pd.merge(existing_data, exported_data, on="disease", how="left")

pd.set_option('display.max_columns', None)
pd.set_option('display.width', 0)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.colheader_justify', 'center')
print(data.head())
```

Wyniki:

```
           disease                      symptoms                       treatment
0                Cukrzyca  Częste oddawanie moczu, pragnienie  Insulina, zmiana stylu życia
1            Nadciśnienie           Ból głowy, zawroty głowy    Leki przeciwnadciśnieniowe
2                   Astma        Świszczący oddech, duszność                   Inhalatory
3         Choroba Wieńcowa  Ból w klatce piersiowej, zmęczenie          Angioplastyka, leki
4  Przewlekła Choroba Nerek                  Obrzęk, zmęczenie                Dializy, leki
```

Oba zestawy danych zostały połączone poprawnie, tworząc wspólny DataFrame zawierający informacje o chorobie, symptomach oraz leczeniu.

5. Podsumowanie

Celem projektu było skuteczne połączenie dwóch zestawów danych, w tym jeden z błędami w nazwach chorób. Użycie fuzzy string matching pozwoliło na pełne dopasowanie, dzięki czemu uzyskano spójny zestaw danych. Algorytmy fuzzy string matching okazały się niezwykle przydatne w przetwarzaniu danych w którym występują błędy w tekście.

6. Bibliografia

[1] DataCamp - Fuzzy String Matching in Python - https://www.datacamp.com/tutorial/fuzzy-string-python (dostęp 20.11.2023)

[2] Medium - Fuzzy matching with FuzzyWuzzy: A comprehensive guide - https://medium.com/@alphaiterations/fuzzy-matching-with-fuzzywuzzy-a-comprehensive-guide-04873f07de31 (dostęp 24.11.2024)

[3] Wikipedia - Odległość Levenshteina - https://pl.wikipedia.org/wiki/Odleg%C5%82o%C5%9B%C4%87_Levenshteina (dostęp 24.11.2024)

[4] Python – dokumentacja - https://docs.python.org/pl/3/

[5] The Fuzz – dokumentacja - https://pypi.org/project/thefuzz/

[6] Pandas – dokumentacja - https://pandas.pydata.org/docs/