Czatbot oparty na rozmytych wyrażeniach regularnych.

M Szewczyk
D Szewczyk

1. **Wstęp**

Cel projektu:

Celem projektu było stworzenie czatbota opartego na rozmytych wyrażeniach regularnych. Czatbot z przygotowanej bazy odpowiedzi wybiera odpowiedzi na pytanie otrzymane od użytkownika.

Technologie wykorzystane w projekcie:

Do przygotowanie czatbota wykorzystano język programowania Python oraz bibliotekę thefuzz.

Python to wysokopoziomowy język programowania ogólnego przeznaczenia, który jest popularny wśród programistów ze względu na swoją prostotę, czytelność kodu oraz bogatą bibliotekę standardową. Jego składnia jest intuicyjna i zapewnia dużą elastyczność w tworzeniu różnorodnych aplikacji, od prostych skryptów po złożone systemy webowe i naukowe.

Thefuzz to biblioteka języka Python służąca do wykonywania operacji związanych z porównywaniem napisów. Biblioteka oferuje różne algorytmy pozwalające na porównywanie napisów pod kątem ich podobieństwa, co może być przydatne w wielu zastosowaniach, np. w wyszukiwaniu tekstu, kategoryzacji lub dopasowywaniu wzorców.

Główne funkcjonalności biblioteki to:

- Obliczanie dystansu Levenshteina, czyli minimalnej liczby edycji potrzebnych do przekształcenia jednego napisu w drugi.

- Porównywanie napisów przy użyciu algorytmu najdłuższego wspólnego podciągu (Longest Common Substring), który mierzy długość najdłuższego wspólnego ciągu znaków występującego w dwóch napisach.

- Porównywanie napisów przy użyciu algorytmu najdłuższego wspólnego podciągu z wagami (Weighted Longest Common Substring), który uwzględnia znaczenie kolejności występowania znaków w napisach.

- Porównywanie napisów przy użyciu algorytmu n-gramów, który dzieli napisy na n-elementowe fragmenty i porównuje je ze sobą.

2. Rozmyte wyrażenia regularne

Rozmyte wyrażenia regularne służą do określania podobieństwa napisów. Wyrażenia te używane są często w przeglądarkach internetowych np. do eliminowania literówek z wypisywanych przez użytkownika zapytań.

Do określania podobieństwa napisów można wykorzystać wiele algorytmów. W przypadku naszego projektu biblioteka thefuzz korzysta z kalkulacji Levenshteina do określania podobieństw tekstu.

Algorytm w celu określenia stopnia podobieństwa pomiędzy dwoma napisami szuka tzw. “edit distance”, jest to metryka która pokazuje ile edycji trzeba wykonać aby zamienić podany napis w ten szukany.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Rys 1. Wzór na odległość Levenshteina

Powyższy wzór służy do wyliczania różnic pomiędzy dwoma sekwencjami słów. Dzięki temu wzorowi możemy określić minimalną liczbę edycji, których potrzebujemy aby zmienić tekst a w tekst b.

Biblioteka thefuzz dostarcza kilku funkcji do porównywania napisów które opisano poniżej.

Simple ratio

Funkcja simple ratio służy do porównywania dwóch napisów i wyliczania “edit distance” pomiędzy nimi. Podobieństwo napisów jest liczbą od 0 do 1 gdzie 1 oznacza, że napisy są identyczne.

Poniżej przedstawiono fragment kodu który prezentuje użycie funkcji ratio.

```python
from thefuzz import fuzz
1 usage
def simple_ratio():
    tekst_1 = "Dawid Kowalski"
    tekst_2 = "Mawid Nowalski"

    print(f"Podobieństwo napisów: {fuzz.ratio(tekst_1, tekst_2)}")



if __name__ == "__main__":
    simple_ratio()
```

```
Podobieństwo napisów: 86

Process finished with exit code 0
```

Partial ratio

Funkcja partial ratio oblicza podobieństwo tekstów na podstawie najkrótszej części tekstu. Funkcja wybiera najkrótszy fragmentu tekstu pierwszego a następnie porównuje go do najkrótszego fragmentu z tekstu do którego wykonujemy porównanie. Funkcja partial ratio bierze pod uwagę również kolejność występowania słów w tekście. Do porównania zostaną wykorzystane pierwsze spotkania najkrótszego słowa z każdego tekstu.

Poniżej przestawiono fragmentu kodu i wyniki ich wykonania które pokazują działanie funkcji partial ratio.

```python
def partial_ratio():
    tekst_1 = "Dawid Kowalski"
    tekst_2 = "Dawid M Kowalski"

    tekst_3 = "Dawid Kowalski"
    tekst_4 = "Dawid Kowalski M"

    print(f"Podobieństwo napisów 1-2: {fuzz.partial_ratio(tekst_1, tekst_2)}")
    print(f"Podobieństwo napisów 3-4: {fuzz.partial_ratio(tekst_3, tekst_4)}")
```

```
Podobieństwo napisów 1-2: 86
Podobieństwo napisów 3-4: 100
```

Kolejność słów w tekstach wpłynęła na wynik podobieństwa tekstów.

Token sort ratio

Funkcja token sort ratio pozwala na ominięcie kolejności słów w porównywanych tekstach. Dzięki tej funkcji możemy porównywać wszystkie słowa występujące w tekście i na tej podstawie określać podobieństwo tekstów.
Poszczególne słowa z tekstów zapisywane są jako tokeny, które następnie porównywane są między sobą.

```python
def token_sort_ratio():
    tekst_1 = "Dawid Kowalski"
    tekst_2 = "Dawid M Kowalski"

    tekst_3 = "Dawid Kowalski"
    tekst_4 = "Dawid Kowalski M"

    print(f"Podobieństwo napisów 1-2: {fuzz.token_sort_ratio(tekst_1, tekst_2)}")
    print(f"Podobieństwo napisów 3-4: {fuzz.token_sort_ratio(tekst_3, tekst_4)}")
```

```
Podobieństwo napisów: 100

Process finished with exit code 0
```

Funkcja pokazała idealne podobieństwo pomimo innej kolejności słów w tekście.

Token set ratio

Funkcja token set ratio działa podobnie do funkcji token sort ratio z tym wyjątkiem, że nie wszystkie tokeny muszą się powtarzać w obu tekstach w celu osiągnięcia pełnego podobieństwa.

```python
tekst_1 = "Dawid Kowalski"
tekst_2 = "Dawid M Kowalski"

tekst_3 = "Dawid Kowalski"
tekst_4 = "Dawid Kowalski M"

print(f"Podobieństwo napisów 1-2: {fuzz.token_set_ratio(tekst_1, tekst_2)}")
print(f"Podobieństwo napisów 3-4: {fuzz.token_set_ratio(tekst_3, tekst_4)}")
```

```
Podobieństwo napisów 1-2: 100
Podobieństwo napisów 3-4: 100

Process finished with exit code 0
```

Po porównaniu tokenów funkcja token set ratio określiła podobieństwo tekstów na 100, co jest maksymalną wartością.

3. Implementacja chatbota

Chatbot komunikuje się z użytkownikiem za pomocą terminala. Użytkownik może wprowadzać zapytania do czatbota na które czatbot odpowiada na podstawie przygotowanej bazy odpowiedzi. Dzięki zastosowaniu funkcji token set ratio z biblioteki thefuzz pytania nie muszą dokładnie odpowiadać tym zapisanym w bazie odpowiedzi.

```
Witaj!
Wpisz pytanie> Hej
Cześć! Jak mogę Ci pomóc?
Wpisz pytanie> Jak się nazywasz?
Jestem Czatter!
Wpisz pytanie> Podaj mi swój ulubiony kolor?
Mój ulubiony kolor to niebieski.
Wpisz pytanie> Pa
Miło było z Tobą porozmawiać. Do zobaczenia!
```

Komunikacja z czatbotem

```
answers_list = {
    "Cześć": ["Cześć! Jak mogę Ci pomóc?", "Witaj! Co mogę dla Ciebie zrobić?"],
    "Jak się masz?": ["Dobrze, dziękuję! A Ty?", "Czuję się świetnie! A co u Ciebie?"],
    "Co robisz?": ["Odpowiadam na Twoje pytania. W czym mogę Ci pomóc?", "Staram się być pomocny. O co pytasz?"],
    "Do widzenia": ["Miło było z Tobą porozmawiać. Do zobaczenia!", "Dziękuję za rozmowę. Żegnam!"],
    "Jaki jest twój ulubiony kolor?": ["Mój ulubiony kolor to niebieski.", "Nie mam ulubionego koloru."],
    "Gdzie mieszkasz?": ["Mieszkam w czarnej skrzynce zwanej komputerem!"],
    "Jak się nazywasz?": ["Jestem Czatter!"],
}
```

Baza pytań czatbota

W czatbocie została również zaimplementowana logika która losowo wybiera przywitanie użytkownika z przygotowanej bazy przywitań.

```python
greetings = [
    "Witaj!", "Dzień dobry!", "Cześć!", "Witam!", "Hej!", "Siema!", "Halo!"
]
```

```python
def chatbot():
    random_greeting = random.choice(greetings)
    print(random_greeting)
```

W głównej części programu po wyświetleniu przywitania czatbot oczekuje na zadanie pytania. Tekst z zapytania jest przekazywany do funkcji find_token_set_ratio która sprawdza czy znaleziono odpowiedź w bazie. Jeżeli czatbot nie znajdzie odpowiedzi na pytanie to wyświetlamy komunikat o braku odpowiedzi.

```python
def chatbot():
    random_greeting = random.choice(greetings)
    print(random_greeting)

    while True:
        question = input("Wpisz pytanie> ")
        found_match = find_match_token_set_ratio(question)
        if found_match:
            answer = select_answer(found_match)
            print(answer)
        else:
            print("Przepraszam, nie rozumiem. Czy możesz powtórzyć?")

        if question.lower() in ["do widzenia", "exit", "wyjscie", "koniec"]:
            print("Miło było z Tobą porozmawiać. Do zobaczenia!")
            break
```

Główna pętla programu odpowiadająca za komunikację z użytkownikiem.

```python
def find_match_token_set_ratio(text):
    match = None
    best_match = 0

    for answer in answers_list:
        similarity = fuzz.token_set_ratio(text.lower(), answer.lower())
        if similarity > best_match:
            best_match = similarity
            match = answer

    return match
```

Funkcja sprawdzająca czy odpowiedź na pytanie znajduje się w bazie.

Jeżeli odpowiedź na pytanie istnieje w bazie to następnie czatbot szuka najbardziej pasującej odpowiedzi. Najbardziej pasująca odpowiedź jest wyszukiwana za pomocą funkcji token_set_ratio z biblioteki thefuzz.

```python
def select_answer(text):
    best_match = 0
    template_matching = None
    answers = []

    for answer, list_of_answers in answers_list.items():
        similarity = fuzz.token_set_ratio(text.lower(), answer.lower())
        if similarity > best_match:
            best_match = similarity
            template_matching = answer
            answers = list_of_answers

    return process.extractOne(text, answers)[0] if answers else None
```

Wyszukiwanie najbardziej pasującej odpowiedzi.

4. Bibliografia

Fuzzy String Matching in Python Tutorial Link
Python wikipedia Link
Thefuzz github Link