

Chatbot z rozmytymi wyrażeniami regularnymi

Julia Kamuda

Dominika Hojniak

Kinga Furmanek

1. Abstrakt

Zrozumienie zapytań użytkowników i dopasowanie ich do precyzyjnych wzorców to jedno z kluczowych wyzwań w tworzeniu aplikacji wykorzystujących przetwarzanie języka naturalnego. W tym raporcie opisano proces budowy chatbota pogodowego, który wykorzystuje techniki fuzzy string matching do rozpoznawania nazw miast, nawet w przypadku literówek lub błędów w zapisie. Aby umożliwić chatbotowi obsługę szerokiej gamy lokalizacji, skorzystano z publicznie dostępnego zbioru danych SimpleMaps, zawierającego szczegółowe informacje o miastach na całym świecie. Aplikacja integruje zewnętrzne API OpenWeatherMap, aby dostarczyć użytkownikom aktualne informacje o pogodzie w czasie rzeczywistym. W projekcie zastosowano biblioteki takie jak TheFuzz i nltk, a interfejs graficzny oparto na bibliotece tkinter. Celem było stworzenie narzędzia, które łączy elastyczność w interpretacji zapytań użytkowników z praktycznym zastosowaniem w obsłudze danych pogodowych.

2. Wstęp

2.1 Cel

Celem projektu było opracowanie aplikacji, która umożliwia użytkownikom uzyskanie informacji o aktualnej pogodzie w wybranych miastach. Kluczowym założeniem było stworzenie systemu zdolnego do rozpoznawania zapytań użytkowników, nawet jeśli zawierają one literówki lub drobne błędy językowe. Ważnym elementem projektu było zaprojektowanie intuicyjnego interfejsu graficznego, który ułatwia korzystanie z aplikacji i zapewnia użytkownikowi pozytywne doświadczenie z chatbotem pogodowym.

2.2 Zakres

Projekt obejmował:

- Wyjaśnienie pojęć związanych z zastosowanymi technologiami w projekcie,
- Przetworzenie danych z SimpleMaps za pomocą biblioteki Pandas w celu utworzenia listy unikalnych miast,
- Implementację algorytmu fuzzy matching do korekcji błędów literowych w zapytaniach użytkownika,
- Wykorzystanie przetwarzania języka naturalnego (NLP) do tokenizacji tekstu i identyfikacji kluczowych informacji,
- Integrację z zewnętrznym API (OpenWeatherMap) w celu pobierania danych pogodowych,
- Stworzenie graficznego interfejsu użytkownika z obsługą zapytań użytkownika i prezentacją wyników.

2.3 Metodyka

Do realizacji projektu wykorzystano:

- **Python:** Język programowania, który posiada prostą składnię i bogaty zbiór bibliotek,
- **Bibliotekę TheFuzz:** biblioteka, wykorzystująca algorytmy porównywania tekstów, takie jak Levenshtein distance,
- **Bibliotekę requests:** do wysyłania zapytań HTTP i odbierania danych z API,

- **Bibliotekę nltk**: do tokenizacji i przetwarzania tekstu,
- **OpenWeatherMap API** do pobierania aktualnych danych pogodowych,
- Zbiór danych **SimpleMaps**: baza danych miast na całym świecie,
- Bibliotekę **Pandas**: do odczytywania i przetwarzania danych z plików CSV,
- **Tkinter** do stworzenia interfejsu graficznego użytkownika (GUI).

3. Teoria

3.1 Chatbot

Chatbot to program komputerowy zaprojektowany do symulowania interakcji z użytkownikiem w sposób zbliżony do ludzkiej rozmowy. Może odpowiadać na pytania, udzielać informacji, a nawet wykonywać zadania na podstawie wprowadzonych zapytań. Chatbot został zaprojektowany w celu udzielania informacji o pogodzie. Jest on wyposażony w mechanizm interpretacji zapytań użytkownika, który uwzględnia potencjalne błędy w pisowni i różnorodność formułowania pytań. Dzięki temu użytkownicy mogą wprowadzać zapytania w naturalny sposób, bez konieczności stosowania sztywno określonych fraz.

3.2 Czym jest fuzzy string matching?

Fuzzy string matching to technika stosowana do porównywania tekstów, które są podobne, lecz nie muszą być identyczne. Jest szczególnie przydatna w przypadku danych tekstowych, które mogą zawierać literówki, błędy ortograficzne, skróty czy zmienioną kolejność słów. Przykładowo, wyszukiwanie w Internecie hasła z błędem ortograficznym nie powoduje, że użytkownik nie otrzyma wyników – mechanizm fuzzy matching umożliwia dopasowanie błędnego hasła do właściwych treści. W projekcie wykorzystano bibliotekę TheFuzz, która opiera się na algorytmie Levenshteina. Algorytm ten oblicza liczbę edycji (np. wstawień, usunięć, zamian znaków), która jest najmniejszą liczbą operacji potrzebnych do przekształcenia jednego tekstu w drugi, co stanowi podstawę dla tej techniki.

3.3 Porównywanie ciągów tekstowych

W projekcie zastosowano metodę `token_set_ratio` z biblioteki TheFuzz, która sprawdza podobieństwo tekstów, ignorując kolejność słów i usuwając wspólne tokeny przed porównaniem. Dzięki temu metoda radzi sobie z zapytaniami zawierającymi te same elementy w różnej kolejności lub dodatkowe słowa. Proces wykorzystuje funkcję `process.extractOne`, która przeszukuje kolekcję dostępnych nazw miast i stosuje metodę `token_set_ratio` do porównania każdego elementu z zapytaniem użytkownika. W ten sposób wybierany jest najlepiej dopasowany element na podstawie współczynnika podobieństwa.

4. Rozwiązanie Problemu

4.1 Dane wejściowe

Aby umożliwić chatbotowi obsługę większej liczby miast, skorzystano z publicznie dostępnego zbioru danych udostępnionego przez **SimpleMaps** pod adresem: <https://simplemaps.com/data/world-cities>. Dane te zawierają szczegółowe informacje o miastach na całym świecie, takie jak nazwa miasta, nazwa miasta w formacie ASCII, współrzędne geograficzne, kraj, populacja oraz liczba ludności w mieście. Dane te zostały przetworzone przy użyciu biblioteki **Pandas**, która umożliwia łatwe odczytywanie, manipulowanie i filtrowanie danych tabelarycznych. Plik został otwarty za pomocą funkcji `pandas.read_csv()`. Wykorzystano kolumnę `city_ascii` w celu utworzenia listy unikalnych nazw

miast, które chatbot może rozpoznawać. Chatbot zyskał możliwość rozpoznawania nazw miast z całego świata, zwiększając funkcjonalność aplikacji.



Rys.1 Mapa miast obsługiwanych przez chatbot

4.2 Implementacja i wyniki

Aplikacja w ramach rozpoczęcia swojego działania uruchamia okno Tkinter, poprzez które jest możliwa komunikacja z chatbotem. Inicjalizowany jest interfejs graficzny GUI, gdzie tworzone jest kolejno okno główne, tytuł, rozmiar, a w dalszej kolejności *Text*, wyświetlający rozmowy oraz kluczowe do działania pole *Entry* przyjmujące wprowadzone przez użytkownika wpisy, które są analizowane od momentu kliknięcia przycisk *Send*, który wywołuje funkcję *get_response()*.

Funkcja ta rozpoczyna przetwarzanie w chatbocie, który uprzednio jest konfigurowany poprzez jego inicjalizację w konstruktorze klasy **Chatbot**. Tworzone są w nim 2 obiekty: *weather_api* z kluczem API do OpenWeather oraz przekazany obiekt posiadający dostęp do bazy wszystkich miast, a także *intent_recognizer* zajmujący się rozpoznawaniem intencji użytkownika. Tak przygotowany obiekt klasy Chatbot otrzymując odpowiedź od użytkownika wywołuje metodę w klasie **IntentRecognizer**, w celu sklasyfikowania intencji wpis, które mogą być następujące. np. *greeting*, *farewell*, *weather*, *help*, *how_are_you*, które symbolizują kolejno intencje użytkownika jako powitanie, pożegnanie, sprawdzenie prognozy pogody, prośbę o pomoc, grzecznościowe zapytanie o stan rozmówcy. Klasa IntentRecognizer posiada zdefiniowany słownik, który zawiera wszystkie możliwe intencje użytkownika w ramach chatbota oraz frazy przy nich występujące, co przedstawiono na listingu 1.

```
self.intents = {
    "greeting": ["hello", "hi", "hey", "good morning", "good evening"],
    "farewell": ["bye", "goodbye", "see you", "later", "exit"],
    "how_are_you": ["how are you", "how are you doing"],
    "weather": ["weather in", "what's the weather", "tell me the weather", "tell me what's the weather",
                "weather"],
    "help": ["help", "what can you do"]
}
```

Listing 1. Słownik z intencjami użytkownika

Frazy w tym słowniku są w dalszym kroku wykorzystywane do dopasowania najlepszej intencji. Umożliwia to biblioteka *theFuzz*, w której korzystamy z modułu *process*, który zawiera w sobie metodę *extractOne* pozwalającą na znalezienie najlepszego dopasowania wejścia z danymi użytkownika z frazami znajdującymi się w słowniku. Jeśli najlepsze dopasowanie

spośród występujących ma wynik > 70 , to jest ono uznawane za intencję użytkownika. Poznanie intencji użytkownika w dalszym kroku pozwala na zwracanie przez chatbot odpowiedniej odpowiedzi, które w większości przypadków zwracają losową odpowiedź spośród tych, które posiada w swojej bazie wiedzy. Poniższy listing 2 przedstawia przykładowe 2 decyzje, w zależności od rozpoznanej intencji w ramach wyrażeń rozmytych. Pierwsza z nich dotyczy zwykłego zapytania i nie posiada ono większej logiki. Druga z nich jednak jest kluczowa dla działania tego chatbota, gdyż stanowi ona decyzję o sprawdzeniu prognozy pogody, która w dalszej kolejności uruchamia mechanizm rozpoznania

```
if intent == "how_are_you":
    return random.choice(
        ["I'm just a bot, but I'm doing great!", "I'm here to help you. How can I assist you today?"])
if intent == "weather":
    city_candidate = self.extract_city_name(user_input)
    return self.weather_api.get_weather(city_candidate)
```

Listing 2. Dopasowywanie intencji w celu wygenerowania odpowiedzi przez chatbot

W przypadku gdy intencją jest pogoda, następuje proces tokenizacji przy pomocy metody `nltk.word_tokenize`, a następnie usuwane są tzw. *stop words*, w celu wyodrębnienia tylko nazwy miasta. Pozostałe tokeny są następnie łączone we frazy, dzięki czemu możliwe jest znalezienie miast zawierających więcej niż jedno słowo (np. New York). Gdy mamy już kandydata na miasto, zastosowana zostaje metoda `process.extractOne` z biblioteki *thefuzz*, która jako licznik podobieństwa używa *token_set_ratio*. Jeśli wynik podobieństwa przekracza 70%, zwracane jest najlepsze dopasowanie a w przeciwnym razie *None*. Listing 3 przedstawia metodę z dopasowaniem.

```
def correct_city(self, user_input): 1 usage (1 dynamic)
    """
    Dopasowuje nazwę miasta do listy miast w bazie danych.
    """
    if not self.cities:
        return None
    print(f"User input for city matching: {user_input}")
    match = process.extractOne(user_input, self.cities, scorer=fuzz.token_set_ratio)
    if match:
        print(f"Best match: {match[0]} with score {match[1]}")
    if match and match[1] > 70:
        return match[0]
    return None
```

Listing 3. Proces znalezienia najlepszego dopasowania dla nazwy miasta

Wynik jest przekazywany dalej do klasy **WeatherAPI** zawierającą metodę, która odpowiada za pobranie aktualnej pogody przy pomocy OpenWeatherMap API. Początkowo, zbadane zostaje czy zwrócony obiekt nie jest *null*, jeśli jest, zwracana jest wiadomość informująca, że szukane miasto nie zostało odnalezione. Jeśli jednak dopasowanie było trafne, wysyłane zostaje zapytanie do API i jako odpowiedź zwrócony zostaje JSON. Pobieramy z niego informację o temperaturze, opisie pogody oraz mieście i zwracamy sformatowaną odpowiedź. Metoda została pokazana na Listingu 4.

```

params = {"q": city, "appid": self.api_key, "units": "metric"}
response = requests.get(self.base_url, params=params)
data = response.json()

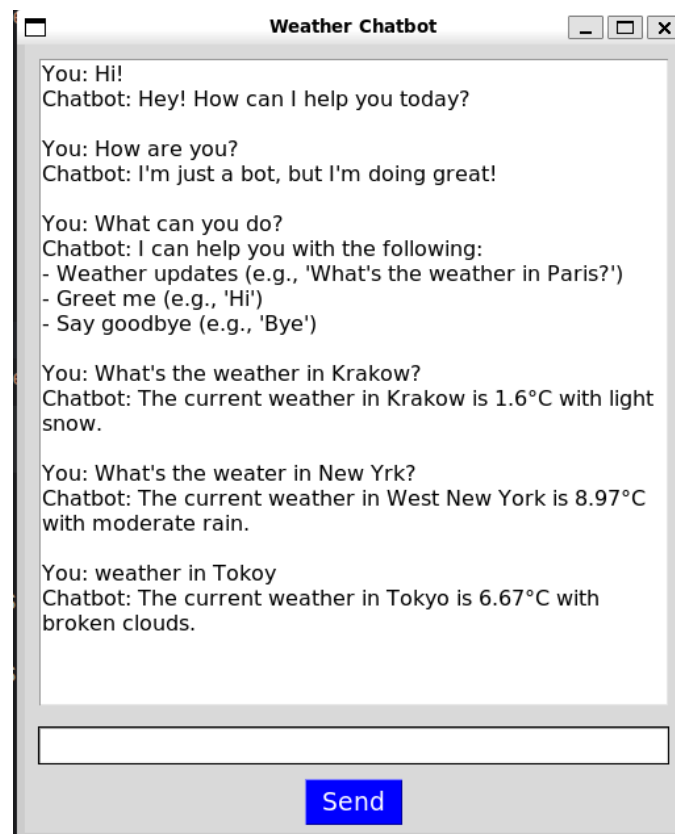
if data["cod"] != 200:
    return f"Error: {data['message']}"

temp = data["main"]["temp"]
weather = data["weather"][0]["description"]
city_name = data["name"]
return f"The current weather in {city_name} is {temp}°C with {weather}."

```

Listing 4. Zapytanie do OpenWeatherMap API i zwrócenie odpowiedzi

Przykład działania Chatbotu:



5. Podsumowanie

Celem projektu było stworzenie prostego chatbota pogodowego z obsługą rozmytych zapytań i interfejsem graficznym. Udało się zrealizować wszystkie główne założenia, a aplikacja działa zgodnie z oczekiwaniami. Chatbot jest nie tylko funkcjonalny, ale także prezentuje przejrzysty i estetyczny interfejs graficzny, który został zaprojektowany z myślą o intuicyjnej obsłudze przez użytkownika. Dzięki swojej prostocie i czytelności, aplikacja pozwala użytkownikowi w łatwy sposób wprowadzać zapytania i otrzymywać odpowiedzi. Aplikacja jest gotowa do dalszego rozwoju.

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/>
- [7] nltk – dokumentacja - <https://www.nltk.org/modules/nltk.html>
- [8] Dokumentacja OpenWeatherMap API: <https://openweathermap.org/api/one-call-api>
- [9] Dokumentacja biblioteki nltk: <https://www.nltk.org/>
- [10] Tkinter - dokumentacja: <https://docs.python.org/3/library/tkinter.html>
- [11] Simplemaps – dokumentacja : <https://simplemaps.com/data/world-cities>