

## **Chatbot z rozmytymi wyrażeniami regularnymi**

Jakub Wiątek - Mateusz Świstak - Grzegorz Zaprzęta

Grupa laboratoryjna nr 5

### **1. Streszczenie pracy**

Opracowanie chatbota bazującego na rozmytych wyrażeniach regularnych obejmowało kilka kluczowych etapów. Były to między innymi: zaprojektowanie oraz zaimplementowanie bota reagującego w odpowiedni sposób na pytania użytkowników oraz przetestowanie jego działania w odpowiedni i szczegółowy sposób.

### **2. Cel**

Celem projektu było stworzenie chatbota zdolnego do obsługiwanie rozmytych wyrażen regularnych. W porównaniu do standardowych wyrażen regularnych, rozmyte wyrażenia regularne są bardziej elastyczne i pozwalają na uwzględnienie stopnia dopasowania do wzorca. Dobrym przykładem na zobrazowanie sposobu działania rozmytych wyrażen regularnych są np. silniki wyszukiwania zaimplementowane w popularnych przeglądarkach. Jeśli przykładowo wpisujemy w pasek przeglądania słowo "facbok" zamiast "facebook" to przeglądarka domyśli się, że chodziło nam o drugie słowo i zwróci odpowiednie wyniki wyszukiwania.

### **3. Zakres**

Zakres pracy obejmuje implementację chatbota w oparciu o rozmyte wyrażenia regularne, zaprojektowanie i zaimplementowanie interfejsu, za pomocą którego bot będzie zdolny do komunikacji z użytkownikiem oraz zaimplementowanie mechanizmu pozwalającego na dostosowywanie stopnia rozmycia wyrażen regularnych, co pozwoli na lepsze dopasowanie się do różnorodnych wariantów pytań użytkowników.

## 4. Implementacja

Wczytywanie odpowiedzi oraz błędów z pliku JSON:

```
def load_answers(self, filename):
    with open(filename, "r", encoding="utf-8") as file:
        data = file.read()
        json_data = json.loads(data)

        self.answers = json_data["answers"]
        self.errors = json_data["errors"]
```

Szukanie najbardziej pasującej odpowiedzi na podstawie wyzwalaczy:

```
def find_answer(self, text: str) -> Answer | None:
    max_match = 50
    best_answer = None

    for answer in self.answers:
        for trigger in answer["triggers"]:
            if isinstance(trigger, str):
                similarity = fuzz.token_set_ratio(text.lower(), trigger.lower())
            else:
                trigger_text = self.process_variables(
                    text, trigger["text"], trigger.get("variables", [])
                )

                similarity = fuzz.token_sort_ratio(
                    text.lower(), trigger_text.lower(), full_process=True
                )

            required_similarity = (
                trigger.get("similarity", 50) if isinstance(trigger, dict) else 0
            )

            if similarity > max_match and similarity > required_similarity:
                max_match = similarity
                best_answer = answer

    return best_answer
```

Przetwarzanie odpowiedzi:

```
def process_answer(self, question: str, answer: Answer) -> str:
    response: Response = random.choice(answer["responses"])
    text = self.process_variables(
        question, response["text"], response.get("variables", [])
    )

    return text
```

Przetwarzanie zmiennych w tekście:

```
def process_variables(
    self, question: str, text: str, variables: List[Variable]
) -> str:
    for variable in variables:
        text = self.process_variable(question, text, variable)

    return text
```

Przetwarzanie pojedynczej zmiennej:

```
def process_variable(self, question: str, text: str, variable: Variable) -> str:
    context_extractor = variable.get("context_extractor", None)
    method = variable.get("method", None)
    fuzzy_extractor = variable.get("fuzzy_extractor", None)

    value = None

    if context_extractor:
        match = search(context_extractor, question)

        if match:
            value = match.group(1)

        self.ctx[variable["name"]] = f"{value if value else ''}"
    elif method:
        arguments = [self.ctx.get(arg, "") for arg in method["arguments"]]
        value = self.methods[method["name"]](*arguments)
        self.ctx[variable["name"]] = value
    elif fuzzy_extractor:
        [value, real_value] = self.fuzzy_extractor(question, fuzzy_extractor)
        self.ctx[variable["name"]] = value
        self.ctx[variable["name"] + "_real"] = real_value
    else:
        value = self.ctx.get(variable["name"], None)

    return text.replace(f"{{{variable['name']}}}", value if value else "")
```

Rozmyta wnioskowanie wartości z tekstu:

```
def fuzzy_extractor(self, text: str, true_values: List[str]) -> tuple[str, str]:
    words = text.split(" ")

    max_match = 50
    best_match = None
    real_value = None

    for word in words:
        for true_value in true_values:
            match = fuzz.token_sort_ratio(word.lower(), true_value.lower())

            if match > max_match:
                max_match = match
                best_match = true_value
                real_value = word

    return (best_match if best_match else "", real_value if real_value else "")
```

Pętla chatu:

```
def chat(self):
    while True:
        try:
            question = input("You# ")
            answer = self.find_answer(question)

            if answer:
                print(f"{self.name}# ", self.process_answer(question, answer))

                if answer["name"] == "exit":
                    break
            else:
                raise Exception()
        except KeyboardInterrupt:
            print("\nGoodbye!")
            break
        except Exception as e:
            print(random.choice(self.errors))
```

Funkcja uruchamiająca bota:

```
def main():
    bot = Bot("Ravin")
    bot.chat()
```

Funkcja dostarczająca informacje o różnych miastach:

```
def city_info(*args) -> str:
    city_def = {
        "Paris": "Paris, the 'City of Light,' is celebrated for its art, history, and culture.",
        "Rome": "Rome, the 'Eternal City,' offers a journey through history with its ancient ruins and art.",
        "Prague": "Prague is renowned for its bridges, cathedrals, and gold-tipped spires.",
        "Barcelona": "Barcelona is known for its art and architecture, highlighted by Gaudí's Sagrada Família.",
        "London": "London is a blend of ancient history and modernity, home to iconic landmarks like Big Ben.",
        "Amsterdam": "Amsterdam is famous for its artistic heritage, elaborate canal system, and historic architecture.",
        "Berlin": "Berlin, known for its modern architecture, turbulent history, and vibrant cultural scene.",
        "Vienna": "Vienna is celebrated for its classical music heritage, imperial architecture, and coffeehouse culture.",
        "Venice": "Venice is renowned for its beautiful waterways, Gothic and Renaissance architecture.",
        "Dublin": "Dublin, known for its historical landmarks like Dublin Castle and its role in Irish history.",
        "Edinburgh": "Edinburgh is a city of striking contrasts, blending medieval old town with modern New Town.",
        "Athens": "Athens is a historical powerhouse, home to ancient landmarks like the Parthenon.",
        "Budapest": "Budapest is known for its dramatic cityscape marked by the Danube River and thermal baths.",
        "Lisbon": "Lisbon, perched on seven hills, is famous for its pastel-colored buildings and historic neighborhoods."
    }

    city = args[0]
    return city_def.get(city, "I don't know this city")
```

Funkcje matematyczne:

```
def addition(*args) -> str:
    return str(int(args[0]) + int(args[1]))

def subtraction(*args) -> str:
    return str(int(args[0]) - int(args[1]))

def multiplication(*args) -> str:
    return str(int(args[0]) * int(args[1]))

def division(*args) -> str:
    return str(int(args[0]) / int(args[1]))
```

Definicja typów danych:

```
class Method(TypedDict):
    name: str
    arguments: List[str]

class Variable(TypedDict):
    name: str
    context_extractor: str
    method: Method
    fuzzy_extractor: List[str]

class Response(TypedDict):
    text: str
    variables: List[Variable]

class Trigger(Response):
    similarity: int

class Answer(TypedDict):
    name: str

    triggers: List[Trigger | str]
    responses: List[Response]
```

## 5. Język programowania

Jako język programowania wybrany został Python. Jest on doskonałym wyborem z wielu powodów. Posiada on stosunkowo czystą i prostą składnię co znacznie ułatwia implementację różnorodnych algorytmów czy też funkcji. Python posiada także bogaty zasób bibliotek do przetwarzania języka naturalnego.

## 6. Użyte biblioteki

Biblioteka użyta do wykonania projektu to TheFuzz. Używa ona tzw. Odległości Levenshteina do obliczania różnic między wyrażeniami. Odległość ta jest minimalną liczbą edycji potrzebnych do zamiany jednego ciągu znaków na drugi. Poniżej kilka przykładów obliczania odległości Levenshteina zaczerpniętych z Wikipedii:

Odległość Levenshteina pomiędzy napisami identycznymi, np.

```
pies
pies
```

jest zerowa – skoro są identyczne, to potrzeba zero działań, by jeden z nich przeprowadzić na drugi.

Odległość Levenshteina pomiędzy napisami:

```
granat
granit
```

wynosi 1, ponieważ do przeprowadzenia pierwszego na drugi wystarcza jedno działanie: zamiana litery **a** na **i**.

## 7. Prezentacja działania chatbota

```
You# hello
Ravin# Hello nice to meet you.
You# who are you
Ravin# I'm Ravin and I'm here to help you.
You# what is 6 plus 3
Ravin# 6 plus 3 is 9
You# tell me about paris
Ravin# Paris, the 'City of Light,' is celebrated for its art
, history, and gastronomy, featuring landmarks like the Eiff
el Tower and Notre-Dame Cathedral. It's also a fashion and c
ulinary capital, offering stylish boutiques and exquisite di
ning.
You# do a flip
I don't understand what you mean. Can you rephrase that?
You# bye
Ravin# See you later
```

## Przykładowa konwersacja z botem

```
You# hi
Ravin# Hey there.
You# hi
Ravin# Hello nice to meet you.
You# hi
Ravin# Hello I'm Ravin and I'm here to help you.
You# hi
Ravin# Hello nice to meet you.
You# hello
Ravin# Hey there.
```

## Randomizacja odpowiedzi bota na to samo pytanie

Silnik dobierania pytań do odpowiedzi bota działa w trybie relacji wielu do wielu. Oznacza to, że na jedno pytanie użytkownik może otrzymać wiele zróżnicowanych odpowiedzi aby zapobiec monotonii, a pytania o podobnej tematyce lecz innej strukturze spotkają się z podobną reakcją bota.

## Zachowanie bota w przypadku podania niepełnego wyrazu

```
You# sth interestng?
Ravin# The Eiffel Tower can be 15 cm taller during the summer due to the expansion of the iron on hot days.
```

```
You# tell me about pari
Ravin# Paris, the 'City of Light,' is celebrated for its art, history, and gastronomy, featuring landmarks like the Eiffel Tower and Notre-Dame Cathedral. It's also a fashion and culinary capital, offering stylish boutiques and exquisite dining.
```

```
You# tell me about brcelone
Ravin# Barcelona is known for its art and architecture, highlighted by Gaudí's Sagrada Família and other modernist landmarks. The lively Las Ramblas street offers street performances and markets.
```

```
You# need some music recomm
Ravin# How about something classic? Try 'Bohemian Rhapsody' by Queen.
```

## 8. Bibliografia

- [1] <https://www.datacamp.com/tutorial/fuzzy-string-python>
- [2] <https://github.com/seatgeek/thefuzz>
- [3] [https://pl.wikipedia.org/wiki/Odleg%C5%82o%C5%9B%C4%87\\_Levenshteina](https://pl.wikipedia.org/wiki/Odleg%C5%82o%C5%9B%C4%87_Levenshteina)