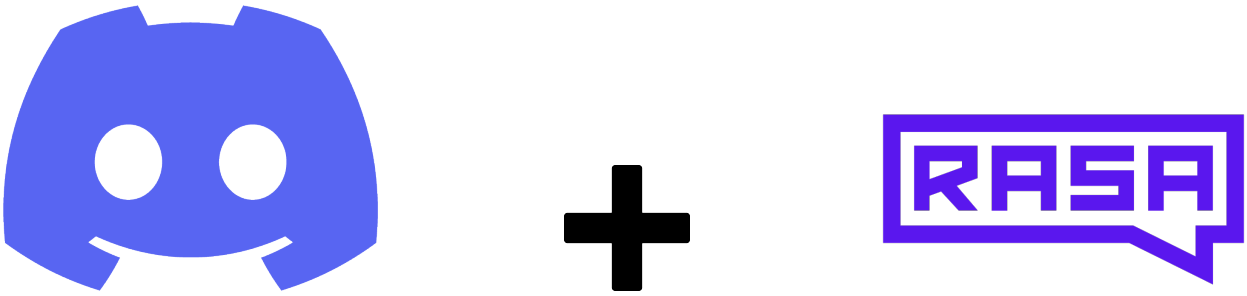
*

Produktywny chatbot Rasa z discordem
Dokumentacja



Wydział Informatyki Politechniki Białostockiej
Politechnika Białostocka
Data:

Spis treści

1 Wprowadzenie i teoria

Rasa to framework, który pozwala na tworzenie chatbotów używających ai. Dzięki niemu można prowadzić konwersację z botem używając różnych API oraz modeli. Aby w pełni zrozumieć czym jest Rasa trzeba wpierw dowiedzieć się czym są modele uczenia maszynowego, CDD oraz NLU.

1.1 Model uczenia maszynowego

Jest to obiekt używający różnych algorytmów do znalezienia wzorców w podanych danych treningowych i na ich podstawie stara się przewidzieć wynik. Model ten może używać różnych algorytmów. Takich jak:

- Regresja Liniowa
- Maszyna Wektorów Nośnych
- Sieci neuronowe

1.2 NLU (Natural Language Understanding)

Jest to część rasy, która odpowiada za analizę poszczególnych wiadomości od użytkownika, czyli klasyfikacja intencji, wyodrębnienie zmiennych z treści wiadomości oraz analiza odpowiedzi. Głównym komponentem są *intencje*, które mogą zawierać *byty*.

1.2.1 Intencje (ang. Intents)

Są to przykładowe wiadomości, jakie użytkownik może wprowadzić. Im mamy ich więcej, tym mniejsza jest szansa, że bot się pomyli. Przykładowa intencja witania się w Rasa wygląda następująco:

Listing 1: Przykładowa intencja

```
- intent: przywitanie
  examples: |
```

```
    - "Cześć!"
    - "Siema"
    - "Hej"
    - "Dzień dobry"
```

1.2.2 Byty (ang. Entities)

Intencje mogą też zawierać byty, czyli informacje zawarte w treści wiadomości. Aby móc je wydobyć potrzebne są specjalne dane treningowe oraz Extractor(ustawiany w konfiguracji chatbota). Takie informacje mogą być przydatne, kiedy chcemy, żeby użytkownik w wiadomości podawał na dane, które możemy później wykorzystać. Przykładem użycia bytów może być firma, która zajmuje się klawiaturami i do tego potrzebuje wiedzieć, jaki model klawiatury jest używany przez użytkownika.

Listing 2: Przykładowa intencja z użyciem bytów

```
- intent:
  examples: |
    - "Zepsułem [superKlawiatura3000]
    - "Nie działa mi [średniaKlawiatura1a00d0]
    - "co mi powiesz o modelu [MFA8W221Ad]?
```

1.3 CDD (Conversation-Driven Development)

CDD jest procesem polegającym na słuchaniu użytkowników i na podstawie zebranych informacji ulepszaniu chatbota. Jest to szczególnie ważny proces, ponieważ nie da się przewidzieć, co powie użytkownik. Użytkownicy używają swoich własnych stwierdzeń oraz wyrażeń, które mogą okazać się niespodziewane. Proces ten nie jest liniowy, ale można go podzielić na pewne kroki:

- Udostępnienie informacji użytkownikom,
- Regularne sprawdzanie oraz ocenianie wyników konwersacji,
- Zapisywanie niespodziewanych wiadomości i dodawanie ich do danych uczących,
- Testowanie asystenta(chatbota),
- Zapisywanie błędów oraz mierzenie wydajności,
- Naprawa niepoprawnych konwersacji.

1.4 Dane konwersacji

Po napisaniu intencji bot musi wiedzieć, co ma odpowiedzieć i do tego będą potrzebne *scenariusze (ang. stories)* oraz *zasady (ang. rules)*. Użytkownik może obrać różne ścieżki, a niemożliwym jest zapisanie i nauczenie go wszystkich możliwości, dlatego będziemy wykorzystywać proces CDD.

1.4.1 Scenariusze

Scenariusze to rozmowy, w których definiuje się, co bot ma odpowiadać na zadane mu pytania. Składają się one z intencji, bytów oraz akcji. Akcja jest odpowiedzią na daną intencję, może być ona prostym zdaniem, albo można zdefiniować skrypt w języku Python. W historiach możemy definiować różne ścieżki, dzięki czemu są elastyczne. Przykładem może być zwykła rozmowa.

Listing 3: Przykładowa historia

```
- story: zwykła rozmowa
  steps:
```

```
  - intent: przywitanie
  - action: utter_zapytanie_co_u_ciebie
  - intent: złe_samopoczucie
  - action: utter_pocieszenie
```

Listing 4: Jak zdefiniowane są odpowiedzi

```
utter_zapytanie_co_u_ciebie:
- text: "Cześć, co u ciebie?"
```

1.4.2 Zasady

Zasady, to część to rozmowy, która nigdy nie może zmienić toru. W odróżnieniu od scenariuszy, w zasadach nie ma różnych ścieżek. Dana zasada jest definitywna i jeśli bot odbierze jakąś intencję, to zasada zmusi bota, żeby zawsze na nią odpowiadał w ten sam sposób. Zasad nie można pisać jak scenariuszy, ponieważ można dodać tylko jedną interakcję użytkownika.

Listing 5: Przykładowa zasada

```
- rule: Zawsze odpowiadaj na przywitanie
  - intent: przywitanie
  - action: utter_zapytanie_co_u_ciebie
```

Listing 6: Jak zdefiniowane są odpowiedzi

```
utter_zapytanie_co_u_ciebie:
- text: "Cześć, co u ciebie?"
```

1.5 Rasa

Teraz, kiedy znane są potrzebne pojęcia, to można odpowiedzieć dokładniej, czym jest Rasa? Jest to framework, który używa wielu modeli uczenia maszynowego oraz technik NLP (Natural Language Processing). Modele są z góry skonfigurowane, ale jest możliwość dostosowania konfiguracji do własnych potrzeb. Rasa pozwala na definiowanie intencji (ang. intents), które reprezentują zamiary użytkowników wyrażane w formie wypowiedzi. Intencje są kluczowe dla zrozumienia, czego użytkownik oczekuje od systemu. Dodatkowo, framework wspiera pracę z bytami (ang. entities), które są istotnymi informacjami wyekstrahowanymi z wypowiedzi użytkownika, takimi jak nazwy, daty, liczby itp. Rasa działa zgodnie z podejściem Conversation-Driven Development (CDD), które promuje iteracyjne rozwijanie systemów konwersacyjnych na podstawie rzeczywistych interakcji z użytkownikami. Dzięki temu, scenariusze (ang. stories) mogą być tworzone i modyfikowane w celu lepszego odwzorowania rzeczywistych dialogów. Framework umożliwia także tworzenie zaawansowanych scenariuszy dialogowych, które definiują sekwencje interakcji między użytkownikiem a botem. Te scenariusze pomagają w zarządzaniu przepływem konwersacji i zapewniają, że bot reaguje w odpowiedni sposób na różne sytuacje. Rasa wspiera również zasady (ang. rules), które pozwalają na definiowanie specyficznych reakcji bota w określonych sytuacjach. Zasady mogą być używane do obsługi prostych przypadków użycia lub do implementacji logiki, która nie wymaga zaawansowanego uczenia maszynowego. Podsumowując, Rasa to potężne narzędzie dla tworzenia inteligentnych asystentów konwersacyjnych, które łączy zaawansowane techniki NLP, elastyczność w konfiguracji modeli oraz podejście CDD do ciągłego ulepszania systemu na podstawie rzeczywistych interakcji użytkowników.

2 Jak używać?

Pierwszym krokiem jest wytrenowanie bota za pomocą komendy: `rasa train` Kolejnym krokiem jest ustawienie bota na discordzie, do niego trzeba stworzyć token bota discordowego w serwisie Discord Developer

Console. Następnie trzeba użyć komendy: `echo "DISCORD_TOKEN=..." > discord_bot/.env` co pozwoli na dodanie zarządzanie botem z discorda. Następnie trzeba włączyć rasę oraz połączyć ją z discordem:

- W pierwszym terminalu należy udostępnić api rasy: `rasa run --enable-api`.
- W drugim terminalu należy włączyć serwer własnych akcji: `rasa run actions`.
- W trzecim terminalu trzeba włączyć bota na discordzie za pomocą komendy: `python discord_bot/bot.py`.

Alternatywnie można używać bota nie w discordzie, tylko w terminalu. Przy wybraniu tej opcji nie da się używać funkcjonalności Pomodoro.

- Najpierw należy włączyć serwer własnych akcji w osobnym terminalu: `rasa run actions`.
- Potem, żeby aktywować bota w terminalu, trzeba użyć komendy: `rasa shell`.

3 Projekt produktywny chat

3.1 Wstęp

W pliku domain występują podstawowe akcje typu response:

- `utter_greet` - "Cześć, co chcesz dzisiaj robić?"
- `utter_goodbye` - "Mam nadzieję, że Ci pomogłem."
- `utter_thank` - "Nie ma problemu, możesz zawsze na mnie liczyć"
- `utter_get_task` - "Jakie zadanie chciałbyś dodać?"
- `utter_get_task_to_remove` - "Jakie zadanie chciałbyś usunąć?"
- `utter_suggest_pomodoro` - "Metoda Pomodoro polega na 25-minutowych blokach pracy z krótkimi przerwami. Chcesz ją wypróbować?"
- `utter_pomodoro_ready` - "Czy jesteś na to gotowy? Jeśli tak, to odpalę pomodoro, ale nie martw się, przypomnę ci o rozpoczęciu kolejnego cyklu lub przerwy."
- `utter_pomodoro_start` - "INITIATE_POMODORO"
- `utter_deny` - "Może innym razem"
- `utter_laptop_greet` - "Witaj! Czy mogę Ci pomóc z wyborem laptopa?"
- `utter_ask_gaming` - "Czy laptop ma być używany do gier?"
- `utter_ask_portable` - "Czy ważna jest dla Ciebie lekkość i przenośność?"
- `utter_ask_budget}` - "Czy masz ograniczony budżet?"
- `utter_recommend_gaming_laptop` - "Polecam laptop do gier. Są one potężniejsze, ale też droższe i cięższe."
- `utter_recommend_portable_laptop` - "Polecam ultrabooka. Są lekkie i przenośne, idealne do pracy w ruchu."
- `utter_recommend_budget_laptop` - "Polecam laptop budżetowy. Są przystępne cenowo, ale mogą mieć ograniczoną moc."
- `utter_recommend_general_laptop` - "Wydaje się, że nie masz szczególnych wymagań. Zalecam uniwersalny laptop."

- `utter_default` - "Nie rozumiem, czy mógłbyś to powtórzyć?"
- `utter_functions` - Potrafie bardzo wiele rzeczy, ale możesz się skupić na:
  - Zarządzanie zadaniami (dodawanie, usuwanie i oczywiście pokazanie)
  - Mogę cię zmotywować
  - Możemy też popracować z pomocą techniki pomodoro
  - Chętnie ci pomogę w kupnie laptopa
  - (szept) mogę też ci coś powiedzieć o pokemonach

Każda intencja ma co najmniej 40 przykładów. Podstawowe intencje:

- `greet` - Przywitanie.
- `thank`- Podziękowanie.
- `inform` - Prośba o podanie funkcjonalności.
- `goodbye` - Pożegnanie.
- `confirm` - Potwierdzenie.
- `deny` - Zaprzeczenie.

W pliku sotries można znaleźć podstawowe historie:

- `story_functions` - Podanie listy funkcjonalności.
- `st_greet` - Przywitanie się.
- `st_goodbye` - Pożegnanie się.
- `thank` - Odpowiedź na podziękowanie.

3.2 Konfiguracja

Konfiguracja jest ustawiona po SpacyNLP, ponieważ ten projekt obsługuje odpowiedzi, które mogą być z różnych dziedzin, a Spacy jest dużo lepiej dostosowane do takiego działania niż podstawowa konfiguracja. Do Spacy użyto modelu języka polskiego `pl_core_news_sm`. Dodatkowo w konfiguracji zmieniono zasady. Podstawową zmianą było dodanie wskaźnika pewności siebie, który jest ustawony na 60%. Jeśli bot nie jest pewny swojego wyboru, to wyśle odpowiedź, że nie rozumie wiadomości. Dodatkowo zwiększono liczbę epok w obu UnexpecTEDIntentPolicy(tutaj też zmieniono rozmiar wsadów) i TEDPolicy.

3.3 Zarządzanie zadaniami

Chatbot może dodawać, usuwać oraz pokazywać zadania. Każdy użytkownik na kanale discorda ma oddzielną listę, te listy oraz ich właściciele są zapisane w pliku tasks.json. Chatbot czyta wiadomości wysłane z discorda i dzięki tej komunikacji może odczytać informację o nadawcy wiadomości. To pozwala na przypisanie oddzielnej listy zadań dla każdego użytkownika. Są tutaj używane trzy akcje niestandardowe:

- `action_add_task` - Akcja odczytuje listę zadań, a jeśli użytkownik jeszcze nic nie dodał, to tworzy nowego użytkownika. Następnie dodaje nowe zadanie na koniec listy.
- `action_remove_task` - Akcja odczytuje listę zadań, a jeśli użytkownik jeszcze nic nie dodał, wyświetla odpowiedni komunikat. W przeciwnym przypadku przeszukuje listę, a następnie usuwa zadanie z listy.
- `action_list_tasks` - Akcja odczytuje listę zadań, a jeśli użytkownik jeszcze nic nie dodał, to wyświetla odpowiedni komunikat. W przeciwnym przypadku iteruje przez listę i wypisudodanie zadania.je kolejno zadania.

Przykładowa akcja:

Listing 7: Akcja dodawania

```
class ActionAddTask(Action):
    def name(self) -> Text:
        return "action_add_task"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        # Przygotowanie danych do pliku.
        path = Path("./data/tasks.json")
        metadata = tracker.latest_message.get("metadata", {})
        user_name = metadata.get("user_name", "nieznany użytkownik")

        # Sprawdzenie, czy plik istnieje.
        if not path.exists():
            with open(path, "w") as file:
                json.dump({}, file)

        # Otwarcie pliku.
        with open(path, "r") as file:
            try:
                data = json.load(file)
            except json.JSONDecodeError:
                data = {}

        # Pobranie ostatnio wysłanej wiadomości przez użytkowika.
        task = tracker.latest_message['text']

        # Dodanie zadania do tablicy z zadaniami.
        if user_name not in data:
            data[user_name] = {"tasks": [{"task": task}]}
        else:
            data[user_name]["tasks"].append({"task": task})

        # Zaktualizowanie pliku nową listą.
        with open(path, "w") as file:
            json.dump(data, file, ensure_ascii=False, indent=4)

        # Powiadomienie użytkownika o dodanym zadaniu.
        dispatcher.utter_message(text=f"Dodano zadanie: {task}")

        return []
```

Ta funkcjonalność wykorzystuje intencje:

- `add_task` - Prośba o dodanie zadania.
- `delete_task` - Prośba o usunięcie zadania.
- `give_task` - Przykładowe zadania, jakie może podać użytkownik. Jest tam 210 przykładów poczynając od prostych takich jak: "Kup mleko", a kończąc na bardziej abstrakcyjnych: "Wysłuchaj ciszy".
- `list_tasks` - Prośba o wypisanie zadań.

Dodano tutaj trzy proste scenariusze, które połączono z zasadami zabezpieczającymi, żeby po prośbie o usunięcie/dodanie zadania, zawsze było można je podać. Scenariusze:

- `Adding a task`
- `Remove a task`
- `Listing tasks`

Zasady:

- `task_remove`
- `task_add`

3.4 Cytaty motywacyjne

Kolejną funkcjonalnością chatu jest wysyłanie motywujących cytatów, kiedy użytkownik np. czuje się przytłoczony lub traci wenę do pracy. Za działanie odpowiada jedna niestandardowa akcja(`action_get_quote`). Wysyła ona zapytanie do zewnętrznego api i w przypadku powodzenia wypisuje cytat wraz z jego autorem, natomiast w przeciwnym przypaku wypisuje odpowiednie powiadomienie. W tej funkcjonalności występuje tylko jedna intencja z prośbą o zmotywowanie(`offer_motivation`) oraz jeden scenariusz(Motivational quote generator) łączący tą akcję z niestandardową akcją.

3.5 Informacje o Pokémon'ach

W tej funkcjonalności działanie jest takie same jak w poprzedniej, z tą zmianą, że trzeba pobrać więcej informacji o Pokémon'ie: nazwa, wzrost, waga, typ. Występuje tutaj jedna, ale bardzo obszerna intencja `get_pokemon` zawierająca 150 przykładów, duża ich ilość jest spowodowana ogromną ilością istniejących Pokémon'ów(w dniu pisania tej dokumentacji liczba ta wynosi 1025). Scenariusz to `Pokemon`. Do zbierania informacji, jakiego pokemona chce poznać użytkownik, wykorzystano byt `pokemon`, który jest zapisywany w slocie `pokemon`.

3.6 Doradca w zakupie laptopa

W tej funkcjonalności chatbot zostaje doradcą zakupowym, który zadaje kilka pytań, na które można odpowiedzieć twierdząco lub przecząco. Na podstawie odpowiedzi rekomenduje odpowiedni rodzaj laptopa. Konwersację zaczyna prośba użytkownika o rekomendację laptopa (`laptop_greet`) Jest to rozmowa, która pokazuje jak rasa pozwala na wybranie różnych możliwych ścieżek wyboru w konwersacji. Wykorzystano tutaj kilka scenariuszy, co tworzy kilka różnych ścieżek:

- `Laptop Recommendation` - Podstawowa ścieżka, w której użytkownik od razu zgadza się na wybór laptopa gamingowego i nie ma sensu dalej prowadzić konwersacji.
- `Laptop Recommendation - Portable` - Ścieżka, w której użytkownik wybiera laptop przenośny i lekki.
- `Laptop Recommendation - Budget` – Ścieżka, w której użytkownik wybiera laptop budżetowy.
- `Laptop Recommendation - General` – Ścieżka, w której użytkownik nie jest pewnien, więc jest mu zarekomendowany laptop ogólny.

Dodatkowo potrzebne są odpowiednie zasady, które ograniczają możliwości wyboru użytkownika, co zwiększa pewność siebie chatbota. Jest w sumie siedem takich zasad nazwanych `laptop r1-7`.

3.7 Pomodoro

Ponieważ tematem projektu jest produktywny bot, wymaga on produktywnego systemu pracy. Jednym z najpopularniejszych i najbardziej efektywnych sytemów jest właśnie technika Pomodoro. Polega ona na cyklach pracy lub uczenia, gdzie praca trwa 25 minut, a przerwa 5 minut. Aby chat mierzył czas na żywo, trzeba było użyć Discorda w połączeniu z Rasą. Do napisania timer'a jest używana funkcja `start_pomodoro_timer(channel)`. Natomiast połączenie między Rasą a Discordem polega na wysłaniu wiadomości włączającej pomodor od chatbota, a w skrypcie Discorda ta wiadomość jest ignorowana, przez co użytkownik jej nie widzi i następnie jest włączana funkcja obsługująca pomodoro. Pomodoro jest inicjowane serią pytań, najpierw jest intencja, gdzie użytkownik pyta się czym jest Pomodoro (`pomodoro`). Następnie chatbot pyta się użytkownika, czy chcę wypróbować pomodoro, a potem pyta się czy jest gotowy i tutaj są trzy różne scenariusze:

- `Pomodoro timer confirm` - Użytkownik chce wypróbować pomodoro.
- `Pomodoro timer deny` - Użytkownik nie chce wypróbowywać tej techniki.
- `Pomodoro timer confirm not ready` - Użytkownik nie jest gotowy.

Dodatkowo są tutaj dwie zasady upewniające się, że pomodoro zawsze się zacznie:

- `pomodoro r1`
- `pomodoro r2`

Spis rysunków