Produktywny chatbot Rasa z discordem
Dokumentacja

Wydział Informatyki Politechniki Białostockiej
Politechnika Białostocka
Prowadzący: Michał Czołombitko
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 informacja zawarta w treści wiadomości. Aby móc je wydobyć potrzebne są specjalne dane treningowe oraz Extractor'a. 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:

1. Udostępnieniu informacji użytkownikom,
2. Regularnym sprawdzaniu oraz ocenianiu wyników konwersacji,
3. Zapisywaniu niespodziewanych wiadomości i dodawaniu ich do danych uczących,
4. Testowaniu asystenta(chatbota),
5. Zapisywaniu błędów oraz mierzeniu wydajności,
6. Naprawie 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 zapisaniei nauczenie go wszystkich możliwych, dlatego będziemy wykorzystywać 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.

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?"
```

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:

1. W pierwszym terminalu należy udostępnić api rasy: `rasa run --enable-api`.
2. W drugim terminalu należy włączyć serwer własnych akcji: `rasa run actions`.
3. 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.

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

3 Projekt produktywny chat

3.1 Wstęp

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.

W pliku domain występują potrzebne odpowiedzi:

- `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_deny` - "Może innym razem."
- `utter_functions` - Wymienienie funkcjonalności.
- `utter_default` - Odpowiedź używana, kiedy bot nie jest wystarczająco pewny swojej odpowiedzi.

3.2 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]]:

        # Ustalenie ścieżki do pliku z listami zadań.
        path = Path("./data/tasks.json")
        # Pobranie nazwy użytkownika z discorda.
        metadata = tracker.latest_message.get("metadata", {})
        user_name = metadata.get("user_name", "nieznany użytkownik")
```

```
# Trzeba to wywalić
if not path.exists():
    dispatcher.utter_message(text="Nie masz jeszcze zadań")

# Otwarcie pliku.
with open("./data/tasks.json", "r") as file:
    data = json.load(file)

#Pobranie zadania podanego przez użytkownika w ostatniej wiadomości.
task = tracker.latest_message['text']

#Sprawdzenie, czy użytkownik został dodany oraz dodanie zadania.
if str(user_name)[1:] not in data:
    data.update({str(user_name)[1:]:{"tasks": [{"task": task}]}})
else:
    data[str(user_name)[1:]]["tasks"].append({"task": task})

with open("./data/tasks.json", "w") as file:
    data = json.dump(data, file)

# Wypisanie odpowiedzi.
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`

3.3 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.

3.4 Informacje o Pokémon'ach

Tutaj działanie jest takie same jak w poprzedniej funkcjonalności, 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`.

3.5 Informacje o Pokémon'ach

Spis rysunków