Boomer - asystent głosowy

Radosław Żerebiec, Dominik Trzópek

Streszczenie

W ramach projektu skonstruowany został asystent głosowy wykorzystujący używane na zajęciach techniki konstrukcji chatbota (wyrażenia regularne, wektory TF-IDF i wyszukiwanie semantyczne.) Bot pozwala na m.in.:

- podawanie daty i godziny;
- pobieranie informacji o pogodzie w podanym mieście;
- przeszukiwanie plików tekstowych;

Zamiast zalecanych w poleceniu, zostały wykorzystane biblioteki *Vosk* oraz *pyttsx3*.

1 Wstęp

Asystent głosowy to narzędzie, które może wykonywać zadania lub usługi dla użytkownika na podstawie poleceń lub pytań. Użytkownicy mogą zadawać asystentom pytania, sterować urządzeniami automatyki domowej i odtwarzaniem multimediów za pomocą poleceń głosowych oraz zarządzać innymi podstawowymi zadaniami, takimi jak np. poczta e-mail, listy rzeczy do zrobienia i kalendarze za pomocą poleceń głosowych.[4]

Celem projektu było skonstruowanie asystenta głosowego wykorzystującego poznane na wykładach i zajęciach laboratoryjnych techniki konstrukcji chatbota:

- wektory TF-IDF;
- wyszukiwanie semantyczne;
- ekstrakcja informacji przy użyciu wyrażeń regularnych.

Powstały w ramach projektu bot informuje użytkownika o obecnej dacie, godzinie i pogodzie oraz pozwala na przeszukiwanie plików tekstowych wykorzystując wyszukiwanie semantyczne. Asystent wykonuje odpowiednią komendę dopiero po podaniu hasła przez użytkownika określonego w konfiguracji programu. Użytkownik głosowo wprowadza polecenie, które jest przetwarzane przez program z wykorzystaniem wyrażeń regularnych i wektorów TF-IDF.

TF-IDF - (term frequency times inverse document frequency) - odwrotna częstość wystąpienia słowa w dokumentach. Każdy dokument reprezentowany jest przez wektor, składający się z wag słów występujących w tym dokumencie. TF-IDF informuje o częstości wystąpienia terminów uwzględniając jednocześnie odpowiednie wyważenie znaczenia lokalnego danego terminu i jego znaczenia w kontekście pełnej kolekcji dokumentów.[3]

Wartość TF-IDF można wyliczyć jako liczbę wystąpień badanego słowa lub frazy (TF) w dokumencie pomnożoną przez logarytm z liczby dokumentów w korpusie podzielonej przez liczbę dokumentów z korpusu, które zawierają badane słowo kluczowe (IDF), korpus stanowi tu zbiór wszystkich badanych dokumentów.[3][1]

Wektory TF-IDF można wykorzystać w algorytmie LSA w celu wyszukiwania semantycznego, czyli opierającego się na znaczeniu zdania, a nie jego statystyki. LSA to algorytm do analizy macierzy TF-IDF w celu zebrania słów w tematy. LSA optymalizuje również te tematy, aby zachować różnorodność wymiarów tematycznych.[1]

LSA[2] – (latent semantic analysis) adresuje takie problemy jak:

- Polisemia - istnienie słów i zwrotów o więcej niż jednym znaczeniu
- Homonimy - słowa o tej samej pisowni i wymowie, ale o innym znaczeniu
- Zeugma - jednoczesne użycie dwóch znaczeń słowa w tym samym zdaniu
- Homografy - słowa pisane tak samo, ale z inną wymową i znaczeniem
- Homofony - słowa o tej samej wymowie, ale różnej pisowni i znaczeniu (wyzwanie NLP z interfejsami głosowymi)

2 Rozwinięcie

Projekt składa się z kilku głównych komponentów:

Adapterów (nazwa wzorowana na *LogicAdapterach* z biblioteki *ChatterBot*), które odpowiadają za przetwarzanie tekstu. Adaptery sprawdzają czy tekst pasuje do jednej z wyuczonych fraz i zwraca funkcję typu `() -> str` (aby można było dynamicznie tworzyć odpowiedzi tj. obecna godzina, data, pogoda, itd.). W finalnej implementacji projektu zostały wykorzystane:

- `TfidfAdapter`: adapter wykorzystujący wektory TF-IDF;
- `RegexAdapter`: adapter dopasowujący wyrażenia regularne do wprowadzanego tekstu i zwracający na ich podstawie odpowiedzi;
- `AdapterAdapter`: adapter, który pozwala łączyć ze sobą inne adaptery.

IOProviderów będących abstrakcją wejścia/wyjścia (zarówno konsoli, jak i interfejsu głosowego). Mimo że celem projektu było stworzenie asystenta głosowego, w ramach testowania możliwe jest wprowadzanie tekstu również za pośrednictwem konsoli;

ConfigurationProvidera odpowiadającego za dostarczanie konfiguracji aplikacji (domyślnie JSON za pośrednictwem klasy `JsonConfigurationProvider`).

Przed uruchomieniem bota wymagane jest utworzenie pliku konfiguracyjnego *config.json*. Przykładowy plik konfiguracyjny jest widoczny na listingu 1. Należy w tym miejscu wyjaśnić, za co odpowiadają odpowiednie opcje w konfiguracji:

```
{
    "default_answer": "Sorry, I don't understand",
    "threshold": 0.6,
    "use_wakeup_sentence": true,
    "wakeup_sentence": "okay boomer",
    "use_console_io": false,
    "weather":
    {
        "openweatherapi_key": "<YOUR_OPENWEATHERAPI_KEY>",
        "default_city": "Cracow"
    },
    "search_directory": "./library"
}
```

Listing 1: Przykładowy plik konfiguracyjny

default_answer odpowiedź bota w wypadku, kiedy nie jest w stanie dopasować pytania zadanego przez użytkownika do żadnego adaptera;

threshold próg pewności który musi osiągnąć `TfidfAdapter`, aby uznać zdanie za pasujące do najbliższego z modelu;

use_wakeup_sentence jeżeli `true`, bot nie będzie próbował zrozumieć polecenia od użytkownika, dopóki nie usłyszy zdania zdefiniowanego w opcji `wakeup_sentence`;

wakeup_sentence zdanie aktywujące wczytywanie polecenia przez bota (jeżeli właściwość `use_wakeup_sentence` jest `true`);

use_console_io jeżeli `true`, zamiast interfejsu głosowego zostanie użyty interfejs tekstowy;

weather.openweatherapi_key klucz do OpenWeatherAPI, wymagany do

weather.default_city domyślne miasto używane, jeżeli przy pytaniu o pogodę użytkownik nie podał miasta;

search_directory katalog, z którego pliki będą przeszukiwane podczas wyszukiwania semantycznego.

Zamiast bibliotek *Blather*, *Sphinx* oraz *PyFestival* (podanych w poleceniu) zostały wykorzystane biblioteki *Vosk* (do rozpoznawania głosu) oraz *pyttsx3* (do zamiany tekstu na mowę).

Ważnym i bardzo ułatwiającym pracę klasą jest `AdapterBuilder`. Zawiera metody tj. `with_questions_single_answer` oraz `with_regexes_single_answer`, upraszczające dodawanie akcji, które można wywoływać wieloma poleceniami. Za jej pomocą utworzony jest główny `AdapterAdapter`, wykorzystywany przez asystenta do przetwarzania poleceń.

Wektory TF-IDF oraz analiza semantyczna przeprowadzana jest w identyczny sposób jak na zajęciach laboratoryjnych, więc kod klas `TfidfAdapter` i `SemanticSearch` jest mocno wzorowany na wykładach.

3 Podsumowanie

Pomimo, że nie wykorzystano bibliotek polecanych przez prowadzącego, osiągnięto satysfakcjonujące wyniki. Asystent rozumie polecenia, nawet jeśli nie zostały dokładnie wypowiedziane (np. zdanie "what's the weather like", zdaniem bota pasuje do polecenia "weather"). Biblioteka *Vosk* czasami ma problemy z rozpoznawaniem niektórych poleceń (prawdopodobnie wina domyślnego modelu, który można zmienić), ale można ten problem obejść wymawiając nieco inne zdanie. Biblioteka *pyttsx3* ma problemy z odczytywaniem języka głosów wbudowanych w system Windows 11, więc trzeba było tej problem obejść. Asystent został skonfigurowany w taki sposób, że zadziała na systemie Windows. Natomiast nie ma pewności, że niektóre funkcje (np. zamiana tekstu na mowę) będą działać na systemach tj. Linux.

Bibliografia

[1] Radosław Kycia. *Przetwarzanie Języka Naturalnego - wykłady i laboratoria*. 2022.

[2] Wikipedia. *Latent semantic analysis*. Online; Dostęp 21.12.2022r. URL: `https://en.wikipedia.org/wiki/Latent_semantic_analysis`.

[3] Wikipedia. *TFIDF*. Online; Dostęp 21.12.2022r. URL: `https://pl.wikipedia.org/wiki/TFIDF`.

[4] Wikipedia. *Virtual assistant*. Online; Dostęp 21.12.2022r. URL: `https://en.wikipedia.org/wiki/Virtual_assistant`.