Università degli studi di Torino

Tecnologie del Linguaggio Naturale

Chatbot "Obi-1"

Autori:
Lorenzo Botto - 882837
Gabriele Naretto - 886277
Nicola Barbaro - 1070668

Anno Accademico: 22/23

Indice

1 Descrizione progetto

Il chatbot "Obi-1" impersonifica il personaggio Obi-Wan Kenobi.
Il sistema è **task-based**: interroga l'utente sulla conoscenza della cultura Jedi per decidere se l'utente può diventare un Padawan.
Il tutto è stato sviluppato in lingua inglese.

La pipeline è composta dai seguenti elementi:

- **Speech Recognition:** l'utente può comunicare verbalmente con il chatbot;
- **Language Understanding:** l'input dell'utente viene elaborato per capire il contenuto delle frasi;
- **Dialog Manager:** gestisce tutto il ciclo di esecuzione della conversazione, tutti i componenti della pipeline e la persistenza dei dati. E' composto da:
  - **Dialog Context Model**
  - **Dialog Control**
- **Response Generation:** in base alla situazione e alla risposta dell'utente, genera la risposta più adatta al contesto;
- **TTS Synthesis:** prende il testo generato dal sistema e lo riproduce vocalmente.

1.1 Simulazione di una esecuzione

La simulazione avviene attraverso questa sequenza di domande e risposte:

1. Viene chiesto all'utente se vuole parlare con il chatbot. In caso di risposta affermativa viene caricato il servizio di Speech Recognition, altrimenti si prosegue per via testuale;
2. Segue una parte di presentazione del chatbot e di richiesta del nome dell'utente;
3. Successivamente, vengono selezionate tre domande, in modo casuale, da una lista di dodici domande a cui deve rispondere l'utente per essere valutato e iniziano i tre cicli di domande e risposte composti da:

- Domanda sulla cultura Jedi
- Risposta dell'utente che può avere quattro risultati:
  - Risposta corretta: il chatbot si congratula con l'utente e passa alla prossima domanda;
  - Risposta incompleta: il chatbot chiede all'utente se può integrare la risposta perchè è incompleta.
    Se l'utente integra completamente la risposta, il chatbot si congratula con l'utente e si passa alla prossima domanda.
    Se l'utente non riesce a completare la risposta, viene ritenuta errata, il chatbot lo fa presente all'utente e prosegue alla prossima domanda.
  - Risposta sbagliata: se ci troviamo alla prima volta in cui l'utente risponde alla domanda, il chatbot dà una seconda possibilità all'utente. Nel caso in cui l'utente rispondesse anche alla seconda opportunità in modo errato, il chatbot ritiene la risposta sbagliata, lo fa presente all'utente e prosegue alla prossima domanda.
  - Risposta fuori contesto: il chatbot possiede delle backup strategy per chiedere all'utente di ristrutturare meglio la risposta.

4. Terminate le domande, il chatbot comunica all'utente se ha passato il test per diventare un Padawan oppure ha fallito e termina la conversazione.

Di seguito viene presentata una esecuzione completa di conversazione:

Do you want to use speech features to comunicate with Obi-1? (if this is the first run, a 500MB model is going to be downloaded.)[y/n]
?-n
Hello, I am Obi-1 and I will question you about Jedi culture.
What is your name?
?- Gabriele
Hello, Gabriele.
We can start the interview with the first question.
What powers the lightsabers?
?- kyber cristal

Bingo! You got it right. Your response is completely accurate.
Where is the headquarters of the Jedi Order located?
?- i don't know
I am sorry, but that is false.
Can you say it again?
?- coruscant?
You are absolutely correct! You have a great understanding of the topic at hand.
What is the most important role a jedi can play?
?- master
You are spot on! Your answer perfectly aligns with the correct solution.
Congratulations, you passed the test with flying colors! You got 3 out of 3 questions correct, making you a Padawan!

---

2 Descrizione componenti

Di seguito verranno descritti in dettaglio tutti i componenti citati nella Sezione 1.

2.1 Dialog Manager

Il costrutto di DM[1] è il coordinatore dell'intero processo di conversazione Uomo-Macchina, esso infatti inizializza e gestisce i due moduli principali di convenzione dialogativa del sistema:

1. *DialogContextModel*: è il modulo adibito al controllo e allo storage delle informazioni rilevanti del dialogo. Sarà proprio l'oggetto DCM[2] a contenere un dizionario dove la chiave è la domanda e il valore è la lista di regex associate ad essa. In fase di inizializzazione verranno sottoposte delle query al database di regex per caricarle in memoria; a seguire l'oggetto DCM utilizzerà i propri metodi interni per interrogare le regex in database e trovare eventuali match con le regex associate a quella particolare domanda. Le regex, essendo divise in "positive" e "negative", causeranno l'attivazione di meccanismi distinti in base alla loro attivazione: quelle positive permetteranno all'oggetto DCM di salvare le informazioni che le hanno triggerate nei corretti slot del frame, quelle negative invece faranno fallire il processo di risposta, restituendo un feedback negativo all'oggetto DC[3]. Inoltre, ad inizio conversazione, quando il DC chiederà all'utente di presentarsi, se presente all'interno del suo database, salverà e utilizzerà il nome dell'utente durante la conversazione per aggiungere profondità alla qualità del dialogo.

2. *DialogController*: L'oggetto DC è il coordinatore di tutti i moduli del programma, ed è inoltre l'arbitro del loop principale, dove il meccanismo di risposta alle domande prende vita. Esso decide qual è il prossimo step da fare in base allo stato attuale della conversazione, ergo:

   (a) Inizialmente permetterà al chatbot di presentarsi e chiederà all'utente di fare lo stesso;

[1] Dialog Manager
[2] DialogContextModel
[3] DialogController

(b) Porrà le domande e attenderà risposte: i feedback che può ricevere dal DCM sono disparati (si veda la sezione 2.3). In caso di risposta positiva esso si muoverà al turno successivo, altrimenti darà un'altra chance all'utente di riscattarsi con una risposta corretta in caso di risposta ambigua, o avviserà l'utente di aver intuito come falsa la risposta fornita. In caso di ripetuto fallimento-incapacità di chiarirsi dell'utente, terminerà il turno di questa domanda e contestualmente si passerà alla successiva;

(c) Al termine dell'intervista farà scattare un processo di feedback da parte del sistema dove verrà restituito l'esito dell'interrogazione all'utente, con susseguente chiusura del sistema e, di conseguenza, dell'applicazione.

2.2 Language Understanding

La parte di Language Understanding ha due principali componenti i **Frame** e le **Regex**, che sono due elementi strettamente collegati fra di loro.

Frame: all'inizializzazione del chatbot viene creato un frame per ogni domanda, questi frame hanno lo scopo di mantenere una persistenza relativa alle risposte dell'utente durante tutta l'interrogazione, inoltre alla fine dell'interrogazione vengono contati quanti frame sono stati completati per decidere se promuovere lo studente al rango di Padawan.
I frame sono delle semplici classi python, il costruttore prende in input il dominio del frame, il suo intento e un dizionario contenente tutti gli slot; dopodiché queste tre componenti verranno salvate tutte in un dizionario (come attributo della classe) in modo da essere facili da recuperare e maneggiare. Altri attributi della classe sono: un booleano che indica se il frame è stato completato e un intero che funge da contatore degli slot completati; questi due ci serviranno, ad esempio, per la generazione di risposte.

Regex: ovvero la parte che si occupa effettivamente della comprensione della risposta dell'utente.
La scelta delle regex è derivata dal fatto che andando a confrontare altri metodi di Language Understanding, le regex (nel dominio del progetto) risultavano più efficienti da implementare e modificare. In caso si trovi un imperfezione nella comprensione di una domanda da parte del nostro chat-

bot, è molto facile salire alla radice del problema e risolverlo.
Le regex vengono salvate su un database tramite la libreria *shelve*, cosi da essere disponibili in tutto il codice.
Le regex sono state sviluppate, prima di tutto, selezionando le domande e per ognuna di queste sono state generate una serie di risposte vere o false. Queste risposte ci hanno permesso di individuare una serie di pattern che ci permettono di capire come risponde l'utente. Abbiamo categorizzato le risposte dell'utente in 4 classi:

- *Risposta corretta*: l'utente ha risposto correttamente alla nostra domanda e, in caso di frame semplici (con un solo slot), il chatbot si congratulerà e si proseguirà alla prossima domanda.

- *Risposta negativa:* le risposte negative ricadono in tutti quei casi in cui l'utente nega la risposta corretta oppure risponde "non lo so" ad una domanda. Andiamo quindi a segnalare che la risposta è sbagliata e diamo un altra possibilità all'utente, se è la prima volta che ha risposto alla domanda, altrimenti si passa a quella successiva.

- *Risposta incompleta:* questa risposta avviene quando l'utente risponde ad una domanda in modo positivo, ma non risponde correttamente a tutti gli slot. In questo caso si continuerà a interrogare, avvertendo l'utente che qualcosa manca.

- *Risposta backup:* questa risposta si attiva quando siamo fuori dal dominio della nostra domanda.
  Esempio: se io chiedessi "quanto fa 2+2?" e l'utente rispondesse "albero", le regex non intercetterebbero nulla e quindi si segnala all'utente che è fuori strada e di provare a cambiare la risposta.

La funzione che controlla le risposte dell'utente è composta da due cicli annidati: nel primo si controlla dalla lista delle regex disponibili e, una volta che è stata trovata la regex che tratta il contesto della domanda, possiamo verificare quali regex combaciano tra quelle negative e positive. Infine si restituisce un booleano per segnalare la classe della risposta.

2.3 Response Generation

Il componente della Response Generation gestisce la generazione della frase da restituire all'utente, in base alla situazione in cui ci troviamo.
La libreria utilizzata è *pySimpleNLG*.

La frase che viene restituita all'utente è scelta in modo casuale da dizionari prefissati di risposte, generati al momento della creazione della classe (e non a ogni richiesta del DM).
Ci sono quattro dizionari, uno per ogni tipologia di risposta (corretta, incorretta, incompleta o incerta), e due dizionari per l'iniziativa (uno per chiedere di riprovare a rispondere e l'altro per chiedere di completare/andare avanti nella risposta). Ogni dizionario contiene cinque diverse risposte come chiavi, e come valori può contenere 0 oppure 1, a seconda se la risposta è già stata utilizzata una volta oppure no. Quando tutte le risposte saranno utilizzate, si resetta il dizionario con tutti i valori a 1. In questo modo si aumenta la casualità.
L'unico dizionario leggermente diverso è quello per le risposte incomplete, in quanto tutte le risposte hanno una parola "sentinel", in modo tale che quando la risposta arriva al DM inserisce al posto della parola "sentinel" ciò che ha capito dall'utente (implementazione dell'active listening).
Esempio generazione della risposta "I did not get that.": si indica che il nome è "I", si indica che il verbo è "get" e si indica che l'oggetto è "that". Inoltre, si definisce che il verbo è al passato ed è negato. Successivamente, *pySimpleNLG* crea la frase che verrà inserita nel dizionario.
Inoltre, avendo le fasi di iniziativa e di risposta separate, vengono create delle frasi ancora più casuali perchè si combinano le scelte casuali di due dizionari separati.
Esempio: se la risposta è stata incompleta, viene selezionata la risposta dal dizionario corrispondente. Poi ci sarà la fase di iniziativa del turno successivo che chiederà di completare la risposta e quest'ultima viene selezionata ancora da un altro dizionario, aumentando la casualità della frase generata.

La richiesta di generazione della frase (ovvero la scelta di una frase, dato che i dizionari sono già generati in precedenza) arriva direttamente dal DM, il quale passa due parametri fondamentali al Response Generator: il turno in cui ci troviamo e la tipologia dell'ultima risposta dell'utente (corretta, sbagliata, incerta o incompleta). La richiesta di generazione può essere un'i-

niziativa (fase iniziale del turno) oppure una risposta (fase finale del turno). Nel caso in cui sia un'**iniziativa** abbiamo diversi casi, in base alla tipologia di turno in cui ci troviamo:

1. Turno di Intro: prima interazione dell'utente con il chatbot, quindi si saluta l'utente e si chiede il suo nome;

2. Turno di Domanda: dipende dall'ultima risposta che ha dato l'utente:

    - Se l'utente ha risposto in modo corretto alla domanda, il chatbot comunica la seconda domanda;
    - Se la risposta è incerta o incorretta, si chiede all'utente di ripetere la risposta;
    - Se la risposta è incompleta, si chiede all'utente se riesce a completare la risposta.

3. Turno di Outro: ultima interazione dell'utente con il chatbot, quindi si comunica all'utente se ha passato o fallito il test.

Nel caso in cui sia una **risposta** abbiamo due casi in base al turno in cui ci troviamo:

1. Turno di Intro: saluto l'utente per nome (se l'ha scritto) e gli dico che sta iniziando l'intervista;

2. Turno di Domanda: si restituisce la risposta all'utente in base all'ultima risposta che ha dato l'utente:

    - Se l'utente ha risposto in modo corretto, il chatbot si congratula (sarà poi compito dell'iniziativa comunicare la prossima domanda);
    - Se la risposta è incerta, si dice all'utente (sarà poi compito dell'iniziativa chiedere di ripetere, in una forma migliore, la risposta);
    - Se la risposta è incompleta, si dice all'utente che manca qualcosa (sarà poi compito dell'iniziativa chiedere di andare avanti);
    - Se la risposta è sbagliata, si dice all'utente che ha risposto in modo sbagliato (sarà poi compito dell'iniziativa chiedere di ripetere la risposta o andare alla prossima domanda).

2.4 Speech Recognition and TTS Synthesis

All'inizio della conversazione con il chatbot, verrà chiesto all'utente se ha interesse nel affrontare le interazioni attraverso comandi e risposte vocali. In caso affermativo, verranno attivati i moduli di speech elaboration. La connessione alla rete è richiesta per entrambi i moduli.

Per questo, la sezione speech è gestita da due principali handlers:

1. *SpeechRecognitionHandler*: verrà dato l'accesso al sistema al driver principale del microfono, e per fase del colloquio dove è previsto l'input dell'utente, il sistema si metterà in ascolto e, al termine dell'input vocale, verrà passato il segnale audio alla libreria *SpeechRecognition*, dove attraverso il modulo *recognizer_google()* verrà interpretato e convertito il segnale in formato testuale, da cui poi seguiranno le analisi della risposta per eventuali rilevazioni di risposte corrette o sbagliate. Nel caso di input mal interpretato, il sistema chiederà, all'utente, di ripetersi, ma nel caso di connessione assente/decaduta il sistema spegnerà il modulo speech e la conversazione continuerà testualmente.

2. *SpeechSynthesizer*: in caso di interesse da parte dell'utente di affrontare il colloquio verbalmente, la libreria *Coqui.TTS* proseguirà ad installare una rete neurale (solo nel caso del primo avvio, poi verrà salvata in cache, ed ha un peso di circa 500MB) per la sintetizzazione vocale del testo. È stato adottato un approccio basato sul Deep Learning su questo task perché le soluzioni di speech preregistrato o di sintesi di linguaggio di markup sono risultate meno coinvolgenti a livello di suono e non permettevano la personalizzazione del timbro di voce. Infatti, il sistema sintetizza, quanto più possibile, la voce del personaggio Obi-Wan Kenobi, rendendo lo scambio quanto più possibile credibile e piacevole. Ogni output del chatbot sarà, a quel punto, sia in streaming audio che, sincronizzato, stampato testualmente su riga di comando, in modo tale da non permettere di perdere informazioni su quanto detto in caso di distrazione o cattiva qualità dell'uscita audio.

3 Valutazione

Trindi Tick List

Domanda 1: Is utterance interpretation sensitive to context? **Yes**
Il sistema interpreta sulla base del contesto ed inoltre non vengono richieste informazioni di un frame/domanda che è già stato concluso.

Domanda 2: Can the system deal with answers to questions that give more information than was requested? **No**
Il sistema non prende in considerazione ciò che non è oggetto della domanda perché, data la necessaria assunzione di mondo chiuso del dominio delle possibili risposte alle domande, non c'è bisogno di investigare tutto ciò che risulta oggettivamente falso, escluse situazioni ambigue o edge cases non previsti dal nostro set di regex.

Domanda 3: Can the system deal with answers to questions that give different information than was actually requested? **Yes**
Il sistema è in grado di comprendere quando l'utente fornisce informazioni diverse e le reputa come un errore.

Domanda 4: Can the system deal with answers to questions that give less information than was actually requested? **Yes**
Il sistema comprende che l'utente ha fornito parte della risposta completa e chiede all'utente se riesce a completare la risposta.

Domanda 5: Can the system deal with ambiguous designators? **Yes**
Il sistema è in grado di comprendere alcuni designatori ambigui e alcuni errori di battitura possibili da parte dell'utente.

Domanda 6: Can the system deal with negatively specified information? **Yes**
Il sistema è in grado di comprendere quando l'utente sta negando delle informazioni corrette. In questo caso chiede all'utente di esprimersi meglio, se ha ancora la seconda possibilità, altrimenti si passa alla seconda domanda.

Domanda 7: Can the system deal with no answer to a question at all? **Yes**

Il sistema vede una non risposta come una risposta sbagliata, perché è come se l'utente non la sapesse. Se l'utente stava rispondendo per la prima volta alla domanda, il sistema chiede di riprovare, altrimenti il sistema passa alla domanda successiva.

Domanda 8: Can the system deal with noisy input? **Yes**
Il sistema è in grado di riconoscere l'input rumoroso non riconoscendo risposte alla domanda e considerandola come una risposta sbagliata.

Domanda 9: Can the system deal with 'help' sub-dialogues initiated by the user? **No**
Il sistema è progettato in modo tale che sia prevista un'iniziativa da parte del chatbot stesso, e se l'utente chiedesse un aiuto sarebbe come applicare un'inversione di iniziativa.

Domanda 10: Can the system deal with 'non-help' sub-dialogues initiated by the user? **No**
Vedere risposta **Domanda 9**.

Domanda 11: Does the system only ask appropriate follow-up questions? **Yes**
Per follow-up sono previste domande di chiarimento per risposte ambigue o che permettono all'utente di avere una seconda chance nel caso di risposta sbagliata o risposta incompleta. Una volta consumato il secondo tentativo, il sistema prosegue con l'interrogazione senza prendere in considerazione i fallimenti e successi passati.
Al termine del processo, però, la bocciatura/promozione è diretta conseguenza dell'andamento dell'intervista.

Domanda 12: Can the system deal with inconsistent information? **Yes**
Le informazioni inconsistenti vengono ritenute delle risposte errate oppure fuori contesto.

Domanda 13: Can the system deal with belief revision? **Yes**
Il sistema è in grado di riconoscere delle revisioni di credenza quando la risposta diventa da affermativa a negativa e viceversa.
Esempio: poniamo che l'utente debba dire "A, B e C". Al primo tentativo risponde "A e B" e al secondo tentativo risponde "¬A, B e C".

Domanda 14: Can the system deal with barge-in input? **No**
Il sistema non è in grado di riconoscere input interrotti da parte dell'utente.

Domanda 15: Is it possible to get a system tutorial concerning the kind of information the system can provide and what the constraints on input are? **No**
Il sistema non prevede la possibilità di richiedere e di mostrare quali informazioni può dare e ricevere.

Domanda 16: Does the system check its understanding of the user's utterances? **No**
Il sistema è in grado di riconoscere solo pattern discorsivi che combaciano con le potenziali risposte alle domande poste.

4 Librerie utilizzate

1. *pySimpleNLG*, Version 0.2.0 - `https://github.com/bjascob/pySimpleNLG`
2. *SpeechRecognition*, Version 3.10.0 - `https://github.com/Uberi/speech_recognition`
3. *TTS*, Version $\sim=$ 0.13.0 - `https://github.com/coqui-ai/TTS`
4. *pydub*, Version $\sim=$ 0.25.1 - `https://github.com/jiaaro/pydub`
5. *PyAudio*, Version 0.2.13 - `https://people.csail.mit.edu/hubert/pyaudio/`

5 Come eseguire

Sul Git del nostro progetto, si può trovare il Readme con la spiegazione in dettaglio per eseguire il chatbot sul proprio PC: `https://github.com/nicobrb/obi-1`