Relazione progetto TLN - Severus-Python

Paolo Bonicco, 833708
Andrea Fancellu, 838776
Fabio Luciani, 863367

7 giugno 2022

Indice

Introduzione

Severus-Python è un sistema di dialogo che simula un'interrogazione ispirata al corso di Pozioni della saga di Harry Potter, tenuto dal professor Severus Piton. Più nello specifico, il sistema interrogherà l'utente sulla composizione di una pozione estratta a sorte da un elenco in memoria, per poi dare una valutazione allo studente. Il software, scritto interamente in Python, compie operazioni di NLP (*Natural Language Processing*), che possiamo far afferire a uno dei due seguenti insiemi:

1. operazioni di NLU (*Natural Language Understanding*), al fine di comprendere il testo fornito dall'utente, estraendo le informazioni necessarie al proseguimento della computazione;

2. operazioni di NLG (*Natural Language Generation*), al fine di realizzare domande e risposte adeguate a seconda del contesto interno.

Il *dialogue system* presentato è *task based*, ossia porta avanti la computazione al fine di realizzare un task interno, l'interrogazione. Questa scelta implementativa permette di ridurre il dominio del dialogo alla sola interrogazione. Inoltre, il sistema è *user-driven*, ossia l'iniziativa viene presa dal software, che pone domande e si aspetta delle risposte.

Implementazione

Frame

La struttura dati utilizzata per rappresentare la memoria del sistema è il *Frame*, e rappresenta lo scheletro del software. Il frame salva tutte le informazioni utili ricevute dall'utente, tra cui possiamo citare:

1. il nome dello studente che si sottopone all'interrogazione;

2. l'insieme degli ingredienti corretti ricevuti in input;

3. il nome della pozione oggetto dell'interrogazione (scelta in maniera randomica tra 3);

4. il numero di tentativi concessi allo studente per dare la risposta corretta alla domanda posta; lo stato d'animo del professore (*mood*), che può assumere i seguenti valori:

   (a) 0, felice;

   (b) 1, neutrale;

   (c) 2, arrabbiato;

   (d) 3, *Potter mode*.

Il *mood* influenza il numero di tentativi concessi all'utente e la valutazione finale, in quanto viene sottratto al numero di tentativi disponibili e al voto generato. Inoltre, al variare di tale parametro, le frasi generate dal sistema cambiano nel tono, passando da un registro più amichevole a uno più aggressivo. Di rilievo è sicuramente la *Potter mode*, modalità che si attiva solo nel caso in cui il nome dell'utente contenga la stringa "Potter". In questo stato le frasi generate mostrano maggiore aggressività del solito, aggiungendo anche alcune citazioni ai film della saga. In caso di attivazione di tale modalità, il docente boccerà lo studente, indipendentemente dal suo rendimento durante l'interrogazione. Esattamente ciò che farebbe Piton con Harry Potter.

NLU

L'implementazione di tecniche di *Natural Language Understanding* ha l'obiettivo di estrarre informazioni utili dalle frasi ricevute in input dallo studente. Di seguito verranno discussi i passaggi necessari al fine di svolgere tale compito.
Il primo passo, una volta ricevuta una stringa in input dall'utente, è quello di parsificare la frase, in modo da individuare, nell'albero a dipendenze, le porzioni di testo che dovrebbero contenere le informazioni richieste. Per fare ciò viene fatto uso del *Dependency Matcher* fornito da *Spacy*, una delle più potenti librerie per svolgere compiti di NLP. Tale risorsa permette di realizzare dei *pattern*, utilizzati per fare "match" con alcune porzioni dell'albero a dipendenze generato a partire da una frase.
Segue un esempio di tale implementazione.
La frase che prendiamo in esame è una possibile risposta fornita da uno studente in fase di interrogazione, "The potion contains Crisopa flies". L'albero a dipendenze è il seguente:

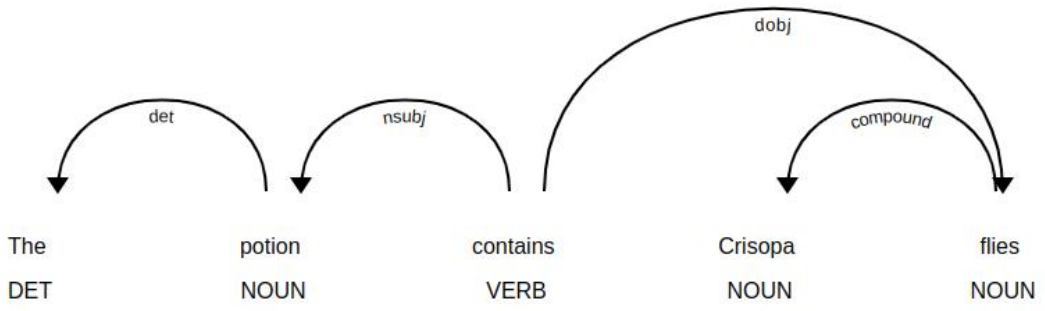

Figura 1. Albero a dipendenze

Il pattern che permette di isolare l'ingrediente, "crisopa flies", è il seguente:

```
pattern_verb_1 = [
    {"RIGHT_ID": "verb",
    "RIGHT_ATTRS": {"LEMMA" : {"IN" : ["contain", "need", "use"]}}
    },
    {
    "LEFT_ID": "verb",
    "REL_OP": ">",
    "RIGHT_ID": "ingredient_1",
    "RIGHT_ATTRS": {"DEP": {"IN" : ["dobj","ccomp"]}}
    },
    {"LEFT_ID": "verb",
    "REL_OP": ">",
    "RIGHT_ID": "ingredient_2",
    "RIGHT_ATTRS": {"DEP": {"IN" : ["dobj","compound", "ccomp"]}}
    }
]
```

Figura 2. Pattern di riconoscimento di una frase

Il pattern opera nel seguente modo. Innanzitutto viene identificato il verbo della frase, l'head, e viene settato un id per tale elemento. Successivamente, riferendosi all'id usato per l'head, viene trovato il suo primo dependent, tale che la relazione di dipendenza tra head e dependent sia nell'insieme scelto ([dobj , ccomp]). Infine si ripete lo stesso procedimento per identificare il secondo dependent, "compund" nel nostro caso. Sono stati realizzati 11 pattern in modo da poter riconoscere una certa varietà di frasi.

NLG

La seconda parte fondamentale di questo progetto è legata alle tecniche di *Natural Language Generation*, tramite le quali possiamo andare a creare delle domande e delle risposte da sottoporre all'utente. La generazione delle frasi avviene tramite la libreria *simpleNLG*. La libreria è scritta in Java, quindi noi abbiamo sfruttato il porting in Python per poterla utilizzare per il nostro progetto.
La libreria permette di creare delle frasi in maniera modulare. Possiamo dividere in tre passaggi fondamentali i processi di *Natural Language Generation*:

- *Text planning*: in questa fase, dopo aver analizzato l'input dell'utente, possiamo decidere che cosa dire. Riguarda esclusivamente il dominio, il linguaggio non gioca nessun ruolo;

- *Sentence planning*: successivamente al text planning, si passa alla fase successiva, il sentence planning, la quale si occupa di decidere quali parole utilizzare e come aggregarle fra di loro. Questo task è influenzato sia dal dominio di cui si sta parlando, sia dal linguaggio;

- *Linguistic realization*: è l'ultimo task da eseguire ed è completamente gestito dalla libreria *simpleNLG*. Questo task riguarda esclusivamente il linguaggio, non è influenzato dal dominio.

A seguire, un esempio della generazione di una domanda.

```
subj = nlg_factory.createNounPhrase("the", f"{pick_random(SIN_POTION)}")
verb = nlg_factory.createVerbPhrase(f"{pick_random(SIN_VERBS)}")
obj = nlg_factory.createNounPhrase(f"{pick_random(SIN_INGREDIENT)}")
p_2 = nlg_factory.createNounPhrase(f"{pick_random(SIN_BESIDES)} {ingredient}")
s_1 = nlg_factory.createClause(subj, verb, obj)
s_1.addPostModifier(p_2)
s_1.setFeature(nlg.Feature.INTERROGATIVE_TYPE, nlg.InterrogativeType.WHAT_OBJECT)
```

Figura 3. Generazione di una frase

Come si può notare dall'immagine, andiamo a costruire le varie parti del discorso della frase che vogliamo creare; in questo caso ogni parola viene scelta randomicamente da varie liste di parole contenenti dei sinonimi, tranne la variabile *ingredient* che tiene traccia dell'ultimo ingrediente corretto risposto dall'utente.
Grazie alle features fornite da *simpleNLG* è necessaria solo una riga di codice per rendere una frase interrogativa.
L'esempio di una possibile frase generata da questo codice è: "What does the potion include besides Bat spleen?"
Infine, per quanto riguarda la valutazione dell'interrogazione, abbiamo deciso di creare delle frasi diverse in base alla votazione presa dallo studente, la quale, come precedentemente spiegato, dipende anche dal *mood* del professore. Come valutazione abbiamo deciso di usare i voti esistenti nella saga di Harry Potter.

Esecuzioni

A seguire aggiungiamo tre simulazioni dell'esecuzione del nostro Dialogue System.

Esecuzione testuale

Nella prima esecuzione il sistema ha un *mood* felice, quindi il professore concederà più tentativi allo studente e sarà più clemente con le risposte e con la votazione.

In questo caso l'interrogazione è andata bene e lo studente Ronald Weasley ha preso il massimo dei voti.

Nella seconda esecuzione il sistema ha un *mood* arrabbiato, quindi si noteranno delle differenze nel modo di porsi del professore.

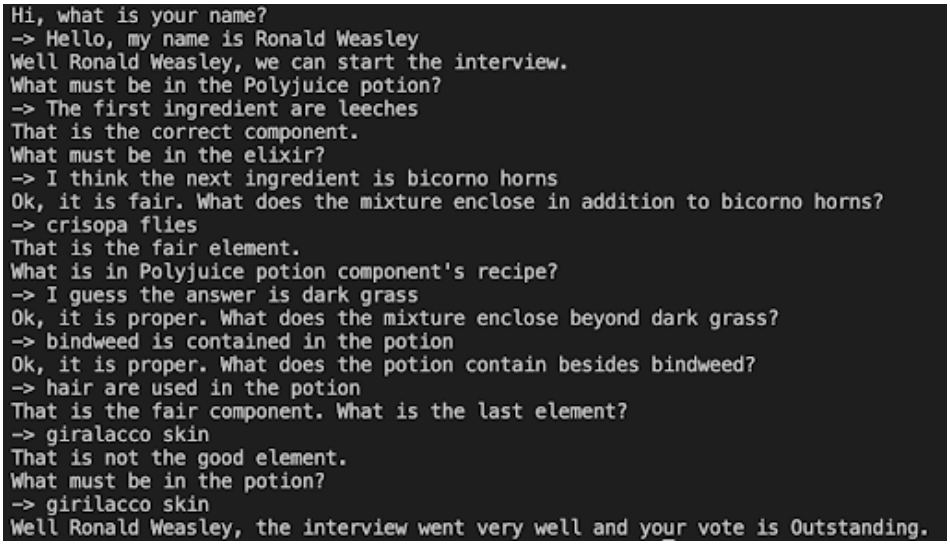

Figura 4. Prima esecuzione

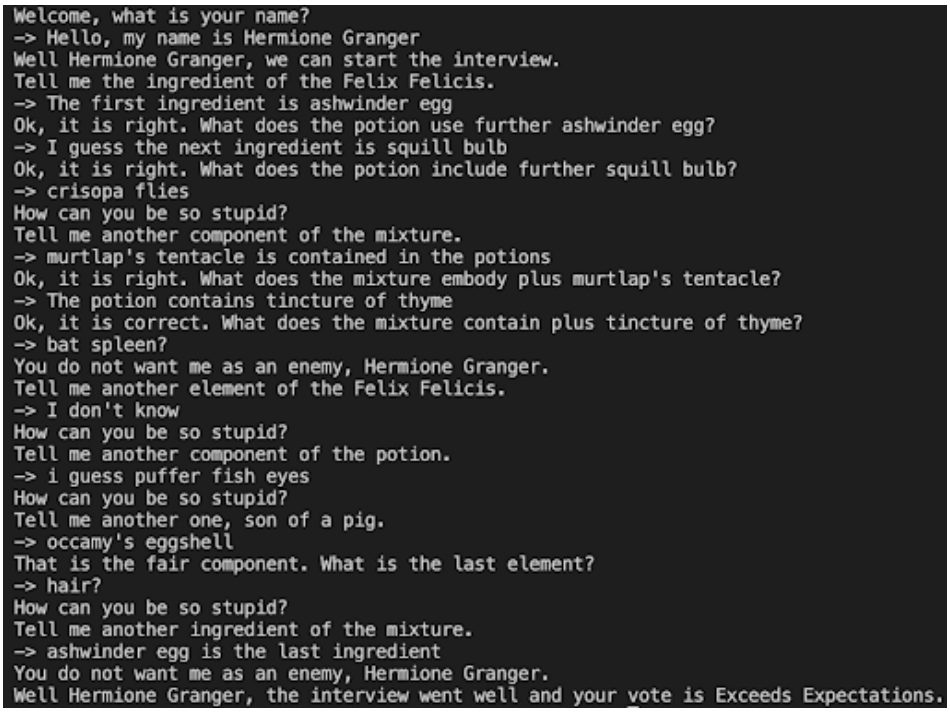

Figura 5. Seconda esecuzione

La terza esecuzione riguarda invece la *potter mode*, infatti alla domanda riguardante il nome, è stato inserito "Harry Potter", scatenando così l'ira funesta del sistema. Possiamo notare come al primo errore commesso, il docente vada su tutte le furie e bocci lo studente.

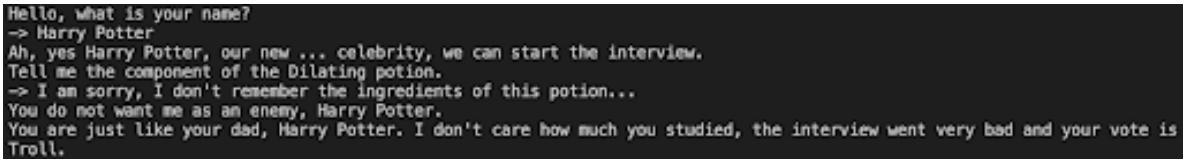

Figura 6. Terza esecuzione

Esecuzione vocale

Come estensione al progetto abbiamo deciso di aggiungere la *speech recognition*, per permettere all'utente di parlare direttamente al sistema, e il *text-to-speech* in modo da permettere al sistema di rispondere in maniera vocale. Commentiamo brevemente l'implementazione di queste due tecnologie:

- speech recognition: abbiamo fatto uso della libreria *speech-recognition* fornita da Google, che mette a disposizione varie funzioni in grado di trascrivere un file audio;
- text-to-speech: abbiamo utilizzato le librerie *gtts* (*Google Text To Speech*) e *playsound*, rispettivamente per produrre ed eseguire un file audio contenente il testo passato in input.

La criticità maggiore a questo livello riguarda proprio la speech recognition, in quanto, a causa del dominio *fantasy* il sistema fatica a riconoscere i nomi degli ingredienti delle pozioni. Per ovviare a questo problema abbiamo deciso, a scopo educativo, di utilizzare al posto delle pozioni di Harry Potter, delle semplici ricette di pasticceria in modo che gli ingredienti siano parole di uso comune. Un'alternativa possibile consiste nell'utilizzare una variante della funzione di speech recognition di Google che prende in input un parametro aggiuntivo sotto forma di lista di stringhe: il testo in atteso in output. Lo svantaggio più grande nell'utilizzo del sistema vocale rispetto a quello testuale è legato all'hardware che viene utilizzato e alla stanza in cui si utilizza il sistema; infatti in una stanza con del rumore di fondo o con un microfono non molto buono si avranno sicuramente più problemi a fare *speech recognition*.

Conclusioni

Lavorare su questo progetto ci ha permesso di esplorare le difficoltà che presenta la costruzione di un sistema di dialogo. Il nostro lavoro è stato sicuramente facilitato dal lavorare su un dominio ristretto, cioè con solo 3 pozioni e legate allo stesso scenario: una interrogazione del corso di pozioni.

Q1: Is utterance interpretation sensitive to context? Non proprio, lavora solo nel contesto dell'interrogazione.

Q2: Can the system deal with answers to questions that give more information than was requested? Si, il sistema è in grado di estrarre l'ingrediente della pozione da una frase che contiene altre parole oltre all'ingrediente.

Q3: Can the system deal with answers to questions that give different information than was actually requested ? Si, il sistema è in grado di distinguere quando una frase contiene un ingrediente corretto e un ingrediente errato.

Q4: Can the system deal with answers to questions that give less information than was actually requested? No, però un possibile sviluppo consiste nel creare un metodo che valuta la similarità della risposta con l'ingrediente detto. Così, in caso che ci fosse un errore grammaticale o una parola mancante dell'ingrediente, il sistema può suggerire la risposta all'utente.

Q7: Can the system deal with no answer to a question at all? Si, la stringa vuota viene trattata come una risposta errata, come in una vera interrogazione.

Q11: Does the system only ask appropriate follow-up questions? Si, il sistema fa delle domande in base alle risposte dell'utente.