ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

# NATURAL LANGUAGE PROCESSING: CHATBOT PER GLI STUDENTI DEL CAMPUS DI CESENA

*Elaborato in*
Programmazione Di Applicazioni Data Intensive

*Relatore*
Prof. Gianluca Moro

*Presentata da*
Andrea Lavista

Seconda Sessione di Laurea
Anno Accademico 2018 – 2019

# PAROLE CHIAVE

Chatbot

Natural Language Processing

Machine Learning

Artificial Neural Networks

Information Retrieval

*A tutti coloro che mi hanno supportato e aiutato*
*nel conseguire questo importante traguardo.*

# Introduzione

La possibilità di sviluppare agenti intelligenti che siano in grado di conversare in linguaggio naturale è oggetto di dibattito sin dalla nascita dell'intelligenza artificiale e, ad oggi, è uno dei temi centrali del natural language processing. Attorno ad esso c'è un forte interesse sia nell'ambito della ricerca accademica sia nell'ambito industriale, con prodotti che sempre di più si stanno imponendo nel mercato, basti pensare agli assistenti vocali presenti negli smartphone, ma non solo.
Ben prima della nascita dell'intelligenza artificiale si è aperto il dibattito su cosa sia l'intelligenza umana e se sia possibile realizzare macchine autonome; se c'è un precursore del dibattito moderno sull'intelligenza artificiale, è sicuramente Cartesio, il cui pensiero comprende anche considerazioni specifiche del linguaggio e del suo rapporto con l'intelligenza umana. Egli, nel suo *Discorso sul metodo* [1] affermò che la ragione e il linguaggio, attraverso cui manifestiamo i nostri pensieri, sono caratteristiche proprie dell'uomo ed esse ci distinguono dagli animali. Infatti, la capacità di esprimersi formando nuove proposizioni in merito a nuovi argomenti, anche astraendo dalla realtà circostante, è una caratteristica unicamente presente nel linguaggio naturale e quindi una capacità unicamente umana.
Ad oggi il progresso nel campo del natural language processing ha portato allo sviluppo di applicazioni in grado di trattare il linguaggio naturale con ottimi risultati, sebbene in precisi e delimitati contesti. Perciò sono ancora molti i progressi da fare in questo campo di ricerca e diversi i problemi ancora aperti in cui si è ancora lontani dal raggiungere soluzioni soddisfacenti.
In questa tesi si affronteranno questi temi, approfondendo lo stato dell'arte nel NLP e del machine learning, l'evoluzione storica e le caratteristiche dei chatbot e infine si presenterà lo sviluppo di un chatbot in un contesto universitario, pensato per il Campus di Cesena dell'Università di Bologna.

# Indice

# Elenco delle figure

# Capitolo 1

# Natural language processing

In questo capitolo introdurremo i concetti base del **natural language processing** (NLP) dando anzitutto una definizione della disciplina e qualche cenno storico, poi approfondendo alcuni aspetti del linguaggio naturale e, infine, lo stato dell'arte. In quest'ultima parte esporrò gli approcci, i task e le più importanti applicazioni, ma anche le più importanti tecniche di rappresentazione del testo e delle parole. Da questo breve capitolo si potrà intuire la poliedricità di questa disciplina, che racchiude in sé molti aspetti del sapere, e il forte interesse da parte del mondo della ricerca in merito, per trovare soluzioni sempre più efficaci a problemi complessi e allo stesso tempo utili.

## 1.1 Che cos'è il NLP?

Il NLP è un campo dell'**intelligenza artificiale** (AI) che si pone come obiettivo il trattamento automatico del **linguaggio naturale**, ovvero l'analisi, la comprensione e l'elaborazione del linguaggio da parte dei calcolatori. Con l'espressione linguaggio naturale si intendono tutte le lingue storico-naturali, ossia le lingue nate spontaneamente lungo il corso della civiltà umana. Come possiamo notare da questa definizione il NLP è un ambito di ricerca interdisciplinare, che spazia tra informatica, intelligenza artificiale e linguistica [5].
Il trattamento automatico del linguaggio ha destato interesse fin dalla nascita dell'informatica e dell'intelligenza artificiale, dato il forte legame tra intelligenza e linguaggio. Non a caso, nel 1950 Alan Turing propose il cosiddetto "**Imitation Game**", definito per testare l'intelligenza di una macchina, che è basato proprio sulla capacità, da parte del calcolatore, di comunicare come un essere umano (figura 1.1).

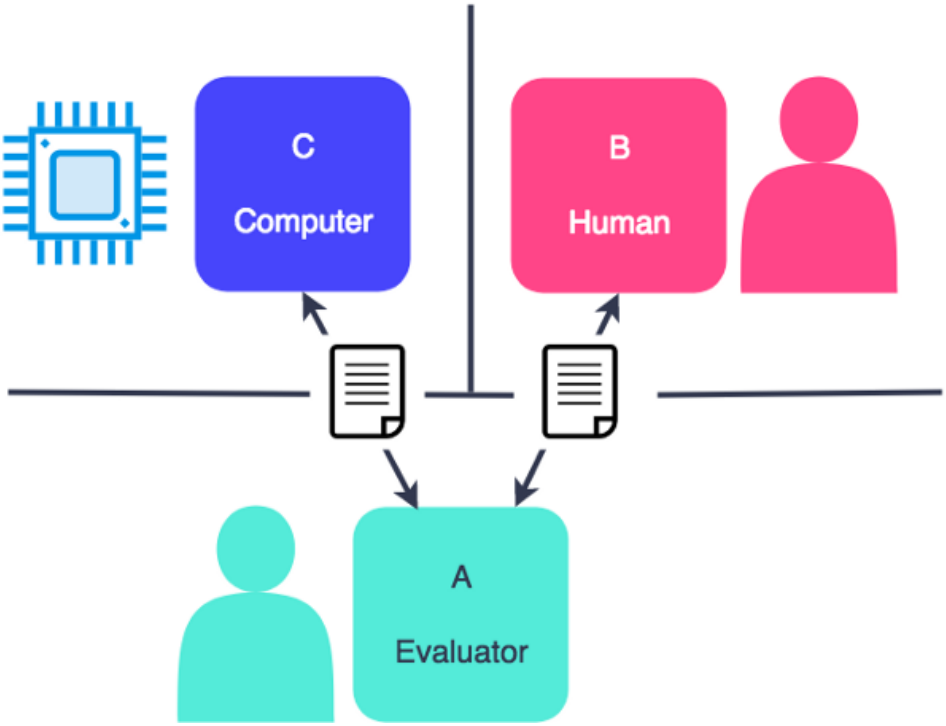


Figura 1.1: A (un essere umano) comunica con B e C senza conoscerli. Se A dopo aver conversato con B e C non riesce a definire chi dei due sia una macchina e chi un uomo, allora la macchina supera il test di Turing [2]

.

La totale comprensione del linguaggio da parte dei calcolatori è un traguardo ancora lontano sebbene negli ultimi anni si stiano facendo enormi passi in avanti, sviluppando sistemi sempre più sorprendenti, capaci di interagire verbalmente, in determinati contesti e modalità. Certamente, il primo passo per una piena comprensione del linguaggio è l'analisi delle regole che determinano le strutture e i processi di formazione delle parole e delle frasi. L'insieme di queste regole si articolano in vari livelli [6]:

- **fonetico**: riguarda i suoni producibili dai parlanti, che sono gli elementi costitutivi di tutti i segni linguistici;
- **morfologico**: lo studio della struttura della parola, in particolare della flessione, della composizione e derivazione delle parole, quindi degli elementi formativi che con la loro composizione danno forma alle parole (morfemi);
- **sintattico**: l'analisi delle strutture delle frasi e degli elementi costitutivi delle stesse (sintagmi), ossia come le parole si combinano tra loro e quali relazioni le legano;

- **semantico**: lo studio del significato delle frasi e delle parole che le compongono;
- **pragmatico**: inerente all'intenzione del parlante e alla funzione comunicativa che un messaggio svolge.

Possiamo constatare che la comprensione di una specifica lingua richiede diverse tipologie di analisi, ognuna delle quali ci dà qualche informazione in più sugli elementi che, a vari livelli, costituiscono i messaggi verbali in una data lingua. Però l'elaborazione del linguaggio naturale non è un compito complicato solo per via delle caratteristiche delle singole lingue, ma dipende da alcune proprietà proprie del linguaggio (inteso come facoltà dell'homo sapiens), e quindi comuni a tutte le lingue. Le principali sono:

- equivocità (o ambiguità): le parole e le frasi possono assumere più significati. Vi sono diversi tipi di ambiguità:
  - lessicale: quando una stessa parola assume più significati (polisemia e omonimia);
  - sintattica: quando l'interpretazione della sintassi non è univoca. Ad esempio nella frase *sono invitate tutte le ragazze e le signore col cappellino* il sintagma *col cappellino* può riferirsi a *le ragazze e le signore* oppure solo a *le signore*;
  - pragmatica: l'interpretazione di un messaggio può dipendere dal contesto linguistico, dalla situazione e dalla condizione del mittente e dei destinatari;
- plurifunzionalità: la possibilità di usare il linguaggio per una lista teoricamente illimitata di funzioni (trasmettere informazioni, esprimere dei pensieri, instaurare rapporti sociali, etc.);
- produttività illimitata: la possibilità di creare messaggi mai prodotti prima e parlare di cose nuove o inesistenti;
- complessità sintattica: i messaggi linguistici presentano strutture molto complesse ed elaborate, di tutt'altro ordine rispetto ai linguaggi formali (tra cui i linguaggi di programmazione).

L'equivocità, la plurifunzionalità e la produttività illimitata sono caratteristiche che conferiscono flessibilità e adattabilità, grazie alle quali si possono esprimere contenuti ed esperienze nuove. Queste sono caratteristiche che consentono al linguaggio di evolversi nel tempo, adattandosi ai cambiamenti storici, sociali, tecnologici e contribuendo ad assolvere al meglio la funzione comunicativa. Perciò il linguaggio naturale ha una potenza comunicativa nettamente

superiore ai linguaggi formali (tra cui i linguaggi di programmazione), ma anche una complessità tale da richiedere l'ideazione di soluzioni e approcci specifici, che esamineremo a partire dal prossimo paragrafo.

## 1.2 Rule-based e statistical NLP a confronto

I due approcci impiegati per risolvere i problemi riguardanti il linguaggio sono **rule-based** e **statistical**; alla nascita di questa disciplina si è puntato molto sul primo approccio, anche a causa delle forti influenze delle teorie sintattiche di **Noam Chomsky** nel mondo della linguistica in quel periodo. Egli infatti pose le basi delle grammatiche generative che sollevarono l'opposizione tra **competenze** (conoscenze linguistiche che consentono di produrre messaggi verbali nella propria lingua) ed **esecuzioni** (messaggi verbali realizzati): solo le competenze, innate negli esseri umani, possono fornire una reale consapevolezza e padronanza del linguaggio, il quale perciò non può essere appreso con la sola esperienza. Tuttavia, a partire dagli anni 80, grazie all'aumento della capacità di calcolo dei computer e i passi in avanti raggiunti nella realizzazione degli algoritmi di **machine learning**, sono emerse le maggiori potenzialità degli approcci statistici in presenza di grandi quantità di dati.
I metodi rule-based hanno come obiettivo quello di codificare l'insieme delle competenze linguistiche in programmi comprensibili per il calcolatore. Questo obiettivo è chiaramente molto arduo da raggiugere data l'enorme quantità di regole che regolano il linguaggio e la loro intrinseca complessità; infatti, seppure i parlanti nativi comunichino tra di loro padroneggiando con facilità il linguaggio appreso nei primi anni di vita, inconsapevolmente fanno affidamento su molte regole e relazioni evidenziati dagli studi della linguistica e delle sue branche (morfologia, sintassi, semantica, etc.). Dunque, seguendo questa metodologia, i programmi possono raggiungere un alto livello di complessità e inoltre gli sviluppatori devono progettare algoritmi differenti per ogni lingua oggetto di studio. Ciò nonostante strumenti di questo tipo risultano ancora utili, ottenendo ottimi risultati in alcuni contesti, ad esempio per la tokenizzazione, ovvero la scomposizione delle frasi in parole.
In contrapposizione ai metodi basati su regole abbiamo i metodi statistici, che basano il loro successo sui **corpora linguistici**. Un corpus è una collezione di documenti scelti in base all'obiettivo prefissato (libri, articoli di giornale, chat, lettere, etc.); generalmente i corpora sono annotati, ovvero gli elementi del discorso dei documenti del corpus vengono etichettati con delle informazioni utili. Il potenziale degli approcci statistici può essere sfruttato appieno con grandi quantità di dati: il progressivo incremento delle capacità computazionali dei calcolatori e la disponibilità di grandi quantità di dati (grazie alla diffusione

del web) hanno consacrato i metodi di machine learning e deep learning, indubbiamente capaci di risolvere problemi complessi e ottenere risultati soddisfacenti anche in applicazioni come question answering, summarization e chatbot.
In definitiva, ad oggi, seppur per task particolarmente complessi applicazioni che impiegano deep learning riescono a ottenere risultano notevolmente più soddisfacenti, la scelta tra i due approcci dipende da ciò che si vuole realizzare. È altresì vero che molto spesso può essere utile impiegare soluzioni ibride che sfruttino entrambi gli approcci.

## 1.3 Applicazione pratiche

Le applicazioni che fanno uso di tecniche di NLP sono molteplici, basti solo pensare al web e alle grandi quantità di informazioni memorizzate in forma testuale: categorizzazione di documenti, analisi delle query di ricerca, estrazione di informazioni, ricerca di documenti sono alcuni esempi di compiti necessari per realizzare un motore di ricerca di qualità.
Altri esempi di sistemi fortemente basati sul NLP sono gli assistenti vocali i quali, già presenti nei nostri smartphone e computer, si stanno diffondendo anche nelle nostre case (ad esempio Amazon Alexa di Amazon, capace di interfacciarsi con smart device casalinghi). Applicazioni come queste saranno centrali negli sviluppi futuri della domotica.
Non ultimi i chatbot, capaci di intrattenere conversazioni in linguaggio naturale si stanno mostrando molto utili per servizi al cliente, dalle prenotazioni di servizi, all'assistenza post-vendita.
Di seguito espongo alcuni dei task più importanti utilizzati nel NLP per la comprensione del linguaggio e alcune delle applicazioni più diffuse oggi, le cui soluzioni spesso fanno uso di alcuni dei task che descriverò [7].

### 1.3.1 Task

- **Tokenizzazione**: individua l'insieme dei segni che compongono una frase (le parole, i numeri, la punteggiatura, etc.) che prendono il nome di token. È usata per risolvere altri task ed è generalmente una delle operazioni di preprocessamento del testo. La tokenizzazione viene svolta efficacemente mediante l'uso delle espressioni regolari.

- **POS tagging**: svolge l'analisi grammaticale definendo le categorie lessicale delle parole in una proposizione; ad esempio nella lingua italiana esse sono articolo, nome, aggettivo, verbo, pronome, avverbio, preposizione, congiunzione, interiezione. Il POS tagging svolge una prima forma di disambiguazione, infatti conoscendo la categoria lessicale di una parola è

possibile definire meglio il significato che quella parola assume all'interno di un determinato contesto.

- **Lemmatizzazione**: riduce la forma flessa di una parola al suo lemma. Il lemma è la forma canonica di una parola ed è la voce lessicale presente nel dizionario.

- **Stemming**: riduce la forma flessa di una parola al suo tema, ovvero ne elimina la desinenza. Questa operazione viene impiegata dai motori di ricerca, e più in generale nelle applicazioni di information retrieval, perché aiuta nel calcolo delle similarità tra documenti e query di ricerca. In questo modo si può verificare se una determinata keyword si ripete in un documento molte volte, seppur in altre forme flesse, incrementando la qualità della ricerca.

- **Parsing**: è l'analisi sintattica di una proposizione; ne individua i sintagmi e soprattutto le relazioni fra gli stessi, producendo come risultato un albero sintattico che le descrive. Esso consente un altro livello di disambiguazione e avvicina ad una maggiore comprensione semantica dell'intera frase.

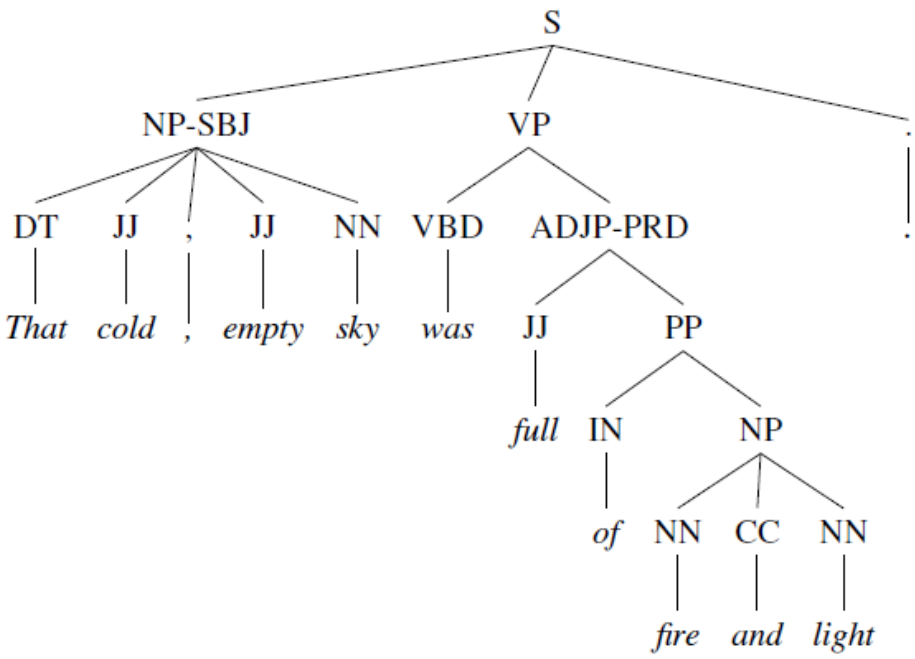


Figura 1.2: L'albero sintattico della frase "That cold, empty sky was full of fire and light", con l'indicazione delle categorie sintattiche delle parole.

- **Named entity recognition (NER)**: la classificazione di entità in categorie predefinite come ad esempio nomi di persona, organizzazioni, località oppure identificare espressioni che indicano un orario, una data, una quantità, un valore monetario e molto altro.

- **Word sense disambiguation**: mira a identificare il significato assunto da una parola in un determinato contesto. Questo task ha un forte impatto su applicazioni di information retrieval, machine translation e di dialogo.

### 1.3.2 Applicazioni

- **Sentiment analysis**: consiste nell'estrazione di opinioni dal testo. Sistemi di questo tipo sono molto utilizzati in vari settori, si rivelano utili ad esempio per rilevare la soddisfazione dei clienti in merito ad un prodotto e in generale lo stato d'animo in merito a determinati fenomeni.

- **Question answering**: consiste nel rispondere a domande espresse in linguaggio naturale. È uno dei problemi aperti nel NLP e fra i maggiormente studiati attualmente; si stanno ottenendo risultati promettenti che combinano tecniche di **information retrieval** (per l'individuazione della risposta in uno o più documenti) e di deep learning (per la generazione delle risposte). Una grande difficoltà dei sistemi di question answering (e chatbot) è di saper gestire domande di ogni tipo e su qualsiasi argomento; ad ora infatti, si ottengono risultati efficaci solo in sistemi riguardanti uno specifico dominio. Sistemi di questa tipologia possono rivelarsi molto utili per le aziende che effettuano assistenza per i clienti, garantendo tempi di risposta rapidi e una copertura totale del servizio.

- **Machine translation**: è la traduzione automatica da una lingua a un'altra. Per riuscire in questo obiettivo è d'uopo raggiungere un'ottima comprensione del testo e della lingua, infatti è considerato un problema **AI-completo**. Questa categoria comprende tutti quei problemi talmente complessi, la cui risoluzione è ritenuta possibile solo da intelligenze le cui capacità sono simili a quelle umane.

- **Automatic summarization**: consiste nel riassumere un contenuto testuale, preservandone le informazioni principali e la correttezza sintattica. Anch'esso è uno dei problemi più impegnativi nel NLP e si rivela utile in diversi contesti come ad esempio nei motori di ricerca o nella generazione automatica di contenuti (articoli di giornale, post sui social media, etc.).

## 1.4 Rappresentazione del testo

I modelli di rappresentazioni del testo sono un aspetto dirimente per i sistemi di NLP e hanno un forte impatto nella qualità e nell'efficacia degli stessi. Di modelli ne esistono molteplici, ognuno con vantaggi e svantaggi; certamente di particolare interesse sono i modelli che riescono a rappresentare al meglio la semantica delle parole, che evidenzino le similarità fra vocaboli e che consentano un'adeguata rappresentazione, e quindi buona comprensione, di testi e documenti di grandi dimensioni [8]. Esaminiamo ora i modelli e le tecniche più diffuse per la rappresentazione del testo:

- **Bag of word**: è un modello di rappresentazione del testo nel quale frasi o documenti sono rappresentati con un multiset di parole distinte. In questo modo si pone l'enfasi sulle parole contenute in un testo e sul numero di occorrenze, non tenendo in considerazione l'ordine e la posizione delle parole. Con modelli di questo genere si possono fare analisi che portano ad esempio a conoscere gli argomenti trattati, perciò sono utili in applicazioni di sentiment analysis, spam filtering e, più in generale, ogniqualvolta si svolge una categorizzazione del testo;

- **N-gram**: è una tecnica che consiste nel memorizzare entità linguistiche contigue (fonemi, lettere, sillabe, parole, etc.) come sequenze di dimensione N. È utile per catturare il contesto nel quale delle parole compaiono, evidenziando espressioni composte da più termini o forti correlazioni tra termini distinti. Sono modelli probabilistici in grado di predire il prossimo elemento a partire da una sequenza, si rivelano quindi utili per il controllo ortografico, l'auto completamento e la generazione di testo.

Le caratteristiche dei modelli bag of word e N-gram non li rendono adatti a contesti in cui è necessaria una profonda comprensione della semantica (question answering, summarization, etc.). Per sopperire a queste mancanze sono state sviluppati negli anni algoritmi di **word embedding**, che consentono di realizzare una rappresentazione delle parole coerente con la loro semantica.

### 1.4.1 Word embedding

Per word embedding si intende una tecnica impiegata nel NLP per rappresentare le parole e le frasi come vettori di numeri reali. Ciò viene realizzato

mediante la costruzione di uno spazio vettoriale (anche di centinaia di dimensioni) in cui si deve preservare al meglio la semantica, per cui le posizioni, le vicinanze, e le proporzioni dei vettori nello spazio devono rispecchiare fedelmente le relazioni semantiche fra le parole.
Questa codifica risulta molto utile perché posso essere svolte operazioni matematiche sui vettori, i cui risultati sono coerenti con le relazioni semantiche. Vediamone un esempio (figura 1.3)):

King - Man + Woman = Quenn


Figura 1.3: Il grafico mostra alcuni vettori rappresentanti parole con word embedding, evidenziandone la conservazione delle relazioni semantiche.

Come si può intuire da questo esempio, siamo di fronte ad uno strumento molto potente il quale, trasformando il testo in elementi matematici, facilita molto l'elaborazione del testo e consente di trarre il massimo profitto dagli stessi.
Uno dei modelli più usati che produce word embedding è **word2vec**: consiste in una rete neurale che riceve in ingresso un corpus di dati (anche di modeste dimensioni) e restituisce un insieme di vettori che rappresentano la semantica delle parole contenute nel testo. Un suo punto di forza di è l'essere un modello non supervisionato, il che vuol dire che il testo ricevuto in input non deve essere annotato, facilitando la fase di training.

Dopo aver dato un'introduzione dell'elaborazione del linguaggio naturale, nel

prossimo capitolo daremo una panoramica del machine learning e del deep learning, approfondendo le principali tipologie di reti neurali e quale impatto hanno avuto queste tecniche nell'ambito del NLP.

# Capitolo 2

# Machine learning

Il machine learning è una branca dell'intelligenza artificiale il cui scopo è lo sviluppo di algoritmi volti all'**apprendimento automatico** delle macchine, estraendo conoscenza dai dati, senza la necessità di essere programmate esplicitamente. L'obiettivo di questi sistemi è produrre delle stime o delle ipotesi su un determinato fenomeno dopo averne analizzato i dati, che costituiscono l'**esperienza**. Il machine learning si è rivelato molto utile per problemi molto complessi nei quali è difficile formalizzare una procedura algoritmica classica, ad esempio quando si cerca di far svolgere alle macchine compiti propri delle facoltà umane. Per queste ragioni ha trovato svariati ambiti di applicazione, tra cui l'elaborazione del linguaggio naturale, la visione artificiale e il decision making.
L'elemento imprescindibile per progettare soluzioni di questo tipo sono i dati: generalmente si costituiscono dei set di dati di grandi dimensioni perché maggiore è la quantità di dati fornita, maggiore sarà la precisione del modello costruito su di essi. Successivamente si procede con la fase di addestramento, fase in cui si crea un modello che possa descrivere al meglio il fenomeno. Infine, si effettua generalmente un test per verificare se il modello risultante produce un output corretto anche con input non presenti nel training set (ovvero il set di addestramento).

## 2.1 Tipologie di apprendimento

In base alle caratteristiche del dataset impiegato per addestrare il modello, si possono indicare tre tipologie di apprendimento: supervisionato, non supervisionato e per rinforzo.

### 2.1.1 Apprendimento supervisionato

Nell'apprendimento supervisionato il modello viene addestrato ricevendo **dati annotati**, ovvero un insieme di possibili input, a ognuno dei quali è associato l'output reale. Il modello quindi cerca di approssimare la funzione in grado di generare, per gli input del training set, gli output corretti. Questo tipo di apprendimento è usato anche per il sentiment analysis: ad esempio nel caso volessimo sviluppare un sistema capace di discernere delle recensioni positive da quelle negative, dovremmo procedere con la creazione di un set di dati per l'addestramento con una composizione analoga a quella mostrata nella tabella 2.1.

| **Titolo** | **Testo** | **Sentiment** |
|---|---|---|
| Ottimo | Davvero un bel libro, mi sento di consigliarne a tutti la lettura, soprattutto agli amanti del genere. | *Positiva* |
| Libro molto difficile | Sconclusionato, privo di logica, noioso, pesante. Non lo consiglio minimamente. | *Negativa* |
| Da avere | Ottimo libro da tenere sempre con sé, del resto l'autore si può annoverare tra i migliori scrittori italiani del 900. | *Positiva* |

Tabella 2.1: Un esempio di dati che descrivono delle recensioni di un libro (input) e il sentiment di ognuna di esse (output).

Una volta creato un dataset con un elevato numero di recensioni etichettate, si procede con la fase di training nella quale il modello cercherà di comprendere delle relazioni tra input e output (non note a priori), da cui generare quelle regole utili alla categorizzazione delle recensioni in positive e negative.
L'esempio che abbiamo appena visto è un classico problema di **classificazione**, in cui l'obiettivo è mappare l'input in una delle predeterminate categorie di output. L'altra grande categoria comprende i problemi di **regressione**, nei quali si effettua invece la previsione di un valore numerico (ad esempio prevedere il futuro valore delle azioni di una società quotata in borsa).

### 2.1.2 Apprendimento non supervisionato

L'apprendimento non supervisionato invece prevede che i dati da cui estrarre conoscenza non siano etichettati, con l'implicazione che il sistema deve individuare autonomamente caratteristiche comuni dalle quali effettuare ipotesi o evidenziare similitudini.

L'apprendimento non supervisionato, sebbene più complicato, presenta una grande vantaggio rispetto a quello supervisionato: non necessita della fase di annotazione del dataset, che è una fase molto dispendiosa in termini di tempo. Questo vantaggio assume ancora più rilevanza nell'ambito del NLP, dove sono spesso necessari dataset di cospicue dimensioni e in alcuni casi l'etichettatura può richiedere specifiche competenze linguistiche.

### 2.1.3 Apprendimento per rinforzo

L'apprendimento per rinforzo è una tecnica che consente al modello di apprendere interagendo con l'ambiente nel quale si trova, ricevendo delle ricompense quando svolge un'azione corretta. L'agente, quindi, apprenderà cercando di massimizzare il numero di premi ricevuti. Il vantaggio di questo approccio è che consente all'agente di apprendere costantemente, adattandosi alle mutazioni dell'ambiente circostante (figura 2.1).

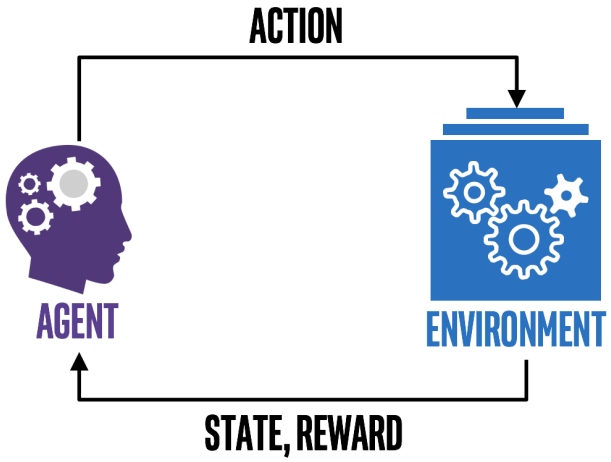


Figura 2.1: Lo schema mostra le interazioni presenti nell'apprendimento per rinforzo.

L'apprendimento per rinforzo è ampiamente utilizzato per la progettazione di **chatbot goal-oriented**, ossia chatbot il cui scopo è risolvere uno specifico problema dell'utente (prenotare una camera d'albergo, comprare il biglietto di un concerto, etc.).

## 2.2 Deep learning

Il deep learning è una branca del machine learning che studia le **reti neurali artificiali**. Questo campo di ricerca sta avendo molto successo nell'ultimo decennio, riuscendo a svolgere alcune delle attività prettamente umane con una confidenza simile e in alcuni casi superiore agli esseri umani. Grazie alle reti neurali si sono ottenuti notevoli miglioramenti, anche nelle applicazioni che fanno uso del linguaggio naturale (e quindi nella realizzazione di chatbot); risultati impensabili sono stati ottenuti altresì con le diagnosi mediche, a riprova della vastità di applicazioni realizzabili con questa tecnologia, e del suo enorme impatto in tutti i rami del sapere.

### 2.2.1 Reti neurali artificiali

Le reti neurali artificiali si ispirano alla struttura delle reti neurali biologiche del cervello umano: esse sono composte da un insieme di unità di calcolo, chiamate neuroni o nodi, connessi tra loro e disposti su più strati (**layer**) [9]. Un esempio di rete neurale è rappresentato nella figura 2.2.

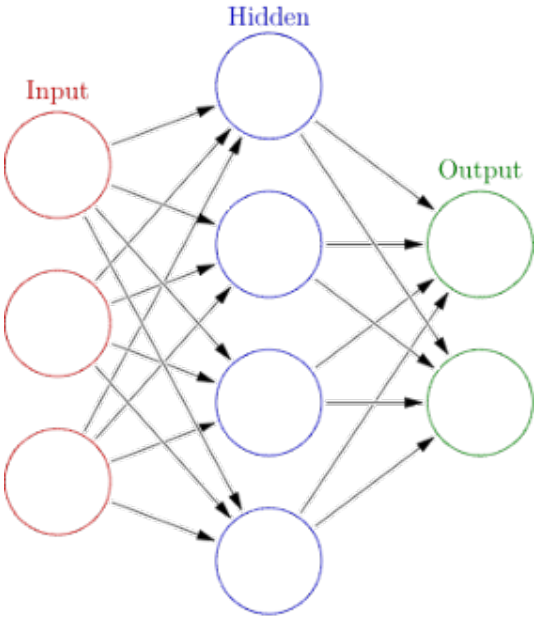


Figura 2.2: Esempio di una rete neurale artificiale [3].

Nell'esempio sopra raffigurato abbiamo tre strati: uno di ingresso (**input layer**) che riceve i dati di input, uno nascosto (**hidden layer**), ma ve ne può

essere anche più di uno, dove avviene effettivamente l'elaborazione dei dati e uno di uscita (**output layer**) che fornisce l'output della rete. I dati quindi percorrono la rete da sinistra verso destra passando attraverso i neuroni, che li elaborano, fino ad ottenere l'output della rete neurale.
L'output finale dipende a sua volta dall'output prodotto dai singoli neuroni presenti nella rete, la cui struttura è rappresentata nella figura 2.3. Ogni neurone riceve degli input, che vengono elaborati attraverso dei pesi, solitamente calcolando una somma pesata; il risultato di questa elaborazione, poi, viene passato ad una funzione di attivazione: è questa a produrre l'output che verrà inoltrato ai neuroni del livello successivo. Di funzioni di attivazione ve ne sono diverse (sigmoide, ReLU, etc.), e la sua scelta dipende dal task che si vuole realizzare.

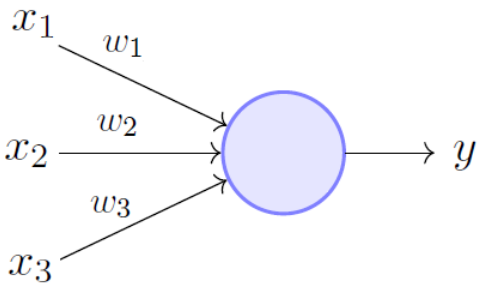


Figura 2.3: Schema di un neurone artificiale. x1, x2, x3 indicano l'input mentre w1, w2, w3 i pesi.

L'output dei neuroni è fortemente legato ai pesi degli input degli stessi e sono questi valori ad essere il fulcro della fase di training. Infatti, l'addestramento di una rete neurale consiste proprio nella definizione dei pesi dei neuroni tali da minimizzare l'errore del task che si vuole implementare. Una tecnica usata per modificare i pesi dei neuroni è la **backpropagation**: basandosi sul calcolo della discesa del gradiente si è in grado di determinare quali modifiche ai pesi riescono a ridurre l'errore, migliorando quindi le prestazioni della rete. Questa procedura viene ripetuta sinché la rete non ha raggiunto l'accuratezza ottimale.

Vi sono diverse tipologie di reti neurali: nell'esempio visto in figura 2.2 ogni neurone è connesso a tutti i neuroni del livello successivo: così facendo l'input di ogni neurone è costituito dall'output dei neuroni del livello precedente. Questo è un esempio di **feedforward neural network**, ovvero reti acicliche in cui i dati si muovono in una sola direzione. Oltre a questa tipologia ne sono sviluppate altre più complesse, di forte interesse per le applicazione di NLP.

### 2.2.2 Recurrent neural network

Le **reti neurali ricorrenti** (RNN) sono una tipologia di rete neurale ampiamente utilizzata nei problemi di NLP. Questa predisposizione è data da una loro caratteristica, assente nelle reti feedforward, ovvero la capacità di conservare in memoria l'elaborazione precedente. Ciò presenta evidenti vantaggi nell'elaborazione del linguaggio dove è necessario tenere in considerazione il contesto nel quale una parola, una frase o un paragrafo si trovano. L'elemento che consente alle reti di avere questa "memoria" sono i cicli (figura 2.4).

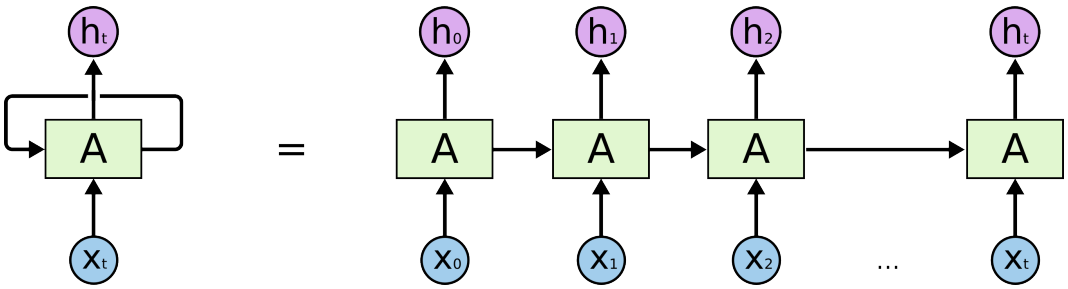


Figura 2.4: Rappresentazione dei cicli in una rete neurale ricorrente (RNN).

L'input elaborato in un neurone in un dato istante di tempo è dato sia dall'input attuale che dal suo output nell'istante di tempo precedente, conservando così traccia di ogni elaborazione in quelle successive. Esempi di applicazioni realizzate con le RNN sono lo speech recognition, ossia il riconoscimento del parlato e la sua conversione in lingua scritta, la predizione di parole e in genere quelle applicazioni che richiedono la generazione di testo.
Le classiche RNN presentano però una problematica: non riescono a gestire al meglio le dipendenze a lungo termine, ad esempio dipendenze tra parole o espressioni distanti tra loro nel testo. A causa di queste limitazioni nella memoria, sono state sviluppate delle RNN che sopperiscono a questa mancanza, denominate **long short term memory** (LSTM) [10].
Le LSTM prevedono al loro interno una memoria che conserva le informazioni più importanti, ritenute necessarie per le successive elaborazioni. Ciò viene realizzato individuando, ad ogni elaborazione, quali sono le informazioni più importanti da inserire in memoria e quali sono invece le informazioni presenti in memoria non più utili.
Vediamo un esempio in cui impiegare le LSTM produce una rete più accurata: ipotizziamo di voler realizzare un modello linguistico che, data una sequenza di parole, predica la prossima parola. Se ad esempio abbiamo la sequenza "Le nuvole sono nel cielo", possiamo notare come la parola *"cielo"* possa essere facilmente predetta conoscendo un numero limitato di parole precedenti. In

altri casi però per fare una previsione accurata potremmo aver bisogno di avere visione di un contesto più ampio, risalendo a molte parole o addirittura a frasi precedenti. Ad esempio, cercando di prevedere l'ultima parola presente nella sequenza "Sono cresciuto in Francia. [...] Io parlo fluentemente il francese", sebbene dalle parole precedenti sia facilmente intuibile che si tratti di una lingua, per predire con certezza la parola *"francese"* è necessario risalire a proposizione precedenti. In questi casi le LSTM offrono prestazioni migliori perché sono in grado di selezionate e conservare in memoria informazioni passate più importanti, anche tra quelle a lungo termine per l'appunto.

## 2.3 Transfer learning

L'addestramento delle reti neurali richiede spesso una grande quantità di dati, il che implica tempi di addestramento piuttosto dilatati. Ciò vale soprattutto nell'ambito del NLP, dove, vista la complessità intrinseca del linguaggio e le sue mille sfaccettature, disporre di un adeguato e rappresentativo dataset è di fondamentale importanza. Ma il tempo di addestramento non è l'unico svantaggio che ne deriva: può capitare di non avere sufficienti dati a disposizione per modellare un dato task, inficiando il risultato finale.
Il **transfer learning** è una tecnica sviluppata di recente che pone rimedio a questi impedimenti e consiste nel creare un modello pre-allenato, che accumuli conoscenza rispetto ad una categoria di problemi. Questo deve essere un modello generale, per garantire la possibilità di essere poi usato per modellare diversi task specifici [11]. Quindi vi è una prima fase (**pre-training**) in cui si genera un modello addestrandolo su grandi quantità di dati e una seconda fase (**fine-tuning**) in cui si trasferisce la conoscenza della rete neurale precedentemente creata in un'altra che soddisferà le specifiche dell'applicazione specifica da realizzare. Grazie al transfer learning si risparmia dunque molto tempo in quanto la fase più dispendiosa, ovvero l'addestramento del modello pre-allenato, è svolta una sola volta ed esso può essere riusato per una molteplicità di task, ottenendo ottimi risultati anche con modeste quantità di dati.

# Capitolo 3

# Chatbot

I chatbot sono software progettati per simulare conversazioni in linguaggio naturale con utenti umani. Nel corso della storia del NLP ne sono stati sviluppati di svariati tipi e applicazioni, dai chatbot pensati solo per intrattenere conversazioni, a quelli utili a richiedere informazioni specifiche, a quelli impiegati in campo medico e molto altro.
Il loro grande successo è senz'altro dovuto alla loro intuitività e semplicità di utilizzo. Il dialogo è la principale attività che svolgiamo per mezzo del linguaggio naturale, sin dai primi anni di età, ed è perciò uno strumento di interfacciamento accessibile ed estremamente ergonomico. Non a caso si è vista una loro ampia diffusione all'interno degli smartphone dove vengono impiegati come interfaccia per altre applicazioni, riducendo l'interazione uomo-macchina prevalentemente ad una sequenza di richieste espresse in forma verbale.

## 3.1 Cenni storici

**Eliza** (1966): è il primo chatbot della storia, creato da Joseph Weizenbaum, è stato progettato per simulare uno psicoterapista rogersiano. Eliza per generare risposte individua le parole chiave del messaggio ricevuto dall'utente con tecniche di pattern matching e formula una domanda pertinente, con lo scopo di far proseguire la conversazione [12].

**Parry** (1972): realizzato dallo psichiatra Kenneth Colby, simula il comportamento di un individuo paranoico e schizofrenico. È più complesso di Eliza, dato che include il modello del proprio stato mentale, che può variare durante la conversazione alterando i propri livelli di paura o rabbia. Parry fu il primo agente di dialogo a passare il test di Turing: si sottoposero delle trascrizioni di interviste a degli psichiatrici, i quali

non riuscirono a distinguere quelle svolte con Parry da quelle con reali individui.

**Jabberwocky** (1981): sviluppato da Rollo Carpenter, questo chatbot si prefigge di effettuare conversazioni in linguaggio naturale in modo divertente e umoristico.

**Dr Sbaitso** (1992): simula uno psicologo, rispetto ai chatbot precedenti riesce a conversare con l'utente tramite la sua voce digitalizzata.

**A.L.I.C.E.** (Artificial Linguistic Internet Computer Entity, 1995): sviluppato da Richard Wallace usando l'AIML (Artificial Intelligence Markup Language), da lui stesso inventato. AIML è un linguaggio di markup che consente di sviluppare agenti di dialogo con metodi euristici di pattern matching; con esso linguaggio è facile sviluppare semplici chatbot relativi a piccoli domini di applicazione.

**Smarterchild** (2001): fu ampiamente diffuso nelle reti di messaggistica istantanea per fornire in tempo reale accesso a informazioni (news, previsione metereologiche, sport, pagine gialle, etc.) e strumenti (calcolatrice, traduttore, etc.).

**Watson** (2006): fu sviluppato da IBM con l'intento di poter competere nello show televisivo "Jeopardy", un quiz televisivo americano. Inizialmente conseguiva scarsi risultati (circa il 15% di risposte corrette) ma nel 2011 riuscì a battere avversari umani. Watson è in grado di rispondere correttamente e in tempi brevi a domande a dominio aperto basandosi su tecniche di machine learning e information retrieval.

**Siri** (2010): è un assistente vocale intelligente di grande successo, che ha dato forte impulso al settore. Attualmente è parte integrante dei sistemi iOS e nasce per supportare gli utenti rispondendo a domande, cercando informazioni in rete e interfacciandosi direttamente gli altri servizi classici degli smartphone (agenda, rubrica, messaggistica, etc.).

**Alexa** (2014): è un'assistente personale intelligente sviluppato da Amazon che è in grado di svolgere compiti impartiti in linguaggio naturale quali riprodurre di musica, memorizzare promemoria, ricercare informazioni e molto altro. È presente in diversi prodotti Amazon, come ad esempio Amazon Echo, progettato per poter essere inserito in un ecosistema più complesso, ad esempio nei sistemi domotici, essendo in grado di gestire e controllare altri dispositivi intelligenti.

**Woebot** (2017): è un agente di dialogo terapeutico che può essere d'aiuto nel trattare determinate condizioni, come depressioni o attacchi d'ansia, mediante l'impiego della terapia cognitivo comportamentale. Sebbene certamente non si possa sostituire ad un terapista umano, può offrire un importante contributo, con terapie personalizzate in base al carattere dell'utente [13].

## 3.2 Tipologie di chatbot

Riguardo all'architettura usata nella progettazione di chatbot, essi si possono suddividere in due macrocategorie: **rule-based** e **corpus-based**, rispecchiando i due principali metodi del NLP, esposti nel primo capitolo (rule-based e satistical). Nei prossimi paragrafi si esamineranno l'approccio orientato alle regole e le architetture dei principali chatbot basati su corpus di dati, ovvero **information retrieval based** e **generativi**.

### 3.2.1 Rule-based chatbot

Usando un approccio basato su regole i chatbot costruiscono le risposte applicando una complessa serie di regole definite in fase di sviluppo. Ciò viene svolto facendo uso di tecniche di **pattern matching** per riconoscere informazioni rilevanti nei messaggi dell'utente, utili alla comprensione degli stessi e quindi alla generazione della risposta. Questa politica comporta che il chatbot è in grado di rispondere adeguatamente nei casi previsti e gestiti a priori, ma non negli altri; perciò maggiore è la complessità delle conversazioni che il chatbot deve riuscire a intrattenere, maggiore è la complessità delle regole che è necessario definire.
Molti dei primi chatbot realizzati furono proprio basati su regole: Eliza, Parry e A.L.I.C.E. sono tra quelli più famosi. Quest'ultimo è basato su **AIML**, un dialetto del linguaggio XML che aiuta nella definizione delle regole e dei pattern di conversazione, e rappresenta una delle migliori tecniche impiegate per questa tipologia di chatbot [14]. Però, a causa della complessità del linguaggio naturale, chatbot di questo tipo si sono dimostrati piuttosto limitanti, tant'è che in seguito ci si è spostati verso altri approcci.

### 3.2.2 IR-based chatbot

I chatbot basati su tecniche di IR fanno uso di corpus di grandi dimensioni nella lingua scelta, spesso contenenti conversazioni (chat di social network, dialoghi di un film, etc.). Per rispondere all'utente essi scelgono all'interno

del corpus la risposta appropriata, mediante tecniche di IR che analizzano la similarità tra la conversazione in corso e quelle immagazzinate. Questo calcolo è spesso implementato con la funzione coseno applicata sulla frequenza delle parole (**tf-idf**) o su word embeddings.
Le politiche impiegate per la scelta della risposta sono molto varie: estrarre la risposta alla frase più simile a quella appena inserita dall'utente oppure direttamente quella più simile ne sono due semplici esempi; inoltre, tenere in considerazione il contesto della conversazione nel quale un messaggio è presente può migliorare la qualità dei risultati. Logicamente, chatbot di questo tipo operano meglio con corpus di grandi dimensioni, potendo avere a disposizione un maggior numero di risposte da usare nella conversazione. Un esempio ne è Cleverbot, un'evoluzione di Jabberwacky, che estende la sua conoscenza memorizzando le conversazioni svolte, apprendendo costantemente.

### 3.2.3 Generative chatbot

I chatbot generativi invece non conversano costruendo frasi basate su modelli predefiniti (rule-based) né ricercate nello storico delle conversazioni già effettuate (information retrieval) bensì generano una risposta da zero, in base al messaggio dell'utente e allo stato della conversazione.
Questi sistemi fanno uso di reti neurali che devono essere allenate per generare risposte grammaticalmente corrette: ciò comporta necessariamente la formazione di dataset di grandi dimensioni o, in alternativa, l'utilizzo del transfer learning di cui si è parlato nel capitolo precedente. Questo è un caso tipico in cui il transfer learning può al tempo stesso facilitare la fase di training (potendo impiegare più piccoli dataset) e migliorare il risultato finale (riuscendo a garantire frasi grammaticalmente corrette grazie al modello pre-allenato).
I chatbot generativi sono tipicamente realizzati con il modello modello **sequence to sequence**, generalmente impiegato nel machine translation, mostrato in figura 3.1. Applicare questo modello allo sviluppo di chatbot vuol dire allenare la rete a tradurre la frase dell'utente nella risposta del chatbot e, sebbene applicazioni di traduzione e di dialogo siano molto differenti, questo paradigma si è diffuso in entrambi i contesti. Il modello Seq2Seq è costituito da due RNN (meglio se LSTM), che fungono una da **encoder** e l'altra da **decoder** [15]. L'encoder elabora l'intero input (nel nostro caso il messaggio dell'utente), parola per parola, e codifica uno stato che ne rappresenta il contesto. Questo stato (corrispondete all'ultimo stato generato dalla rete, ossia relativo all'intera frase) viene poi passato al decoder, che lo utilizza come stato iniziale da cui generare la risposta.

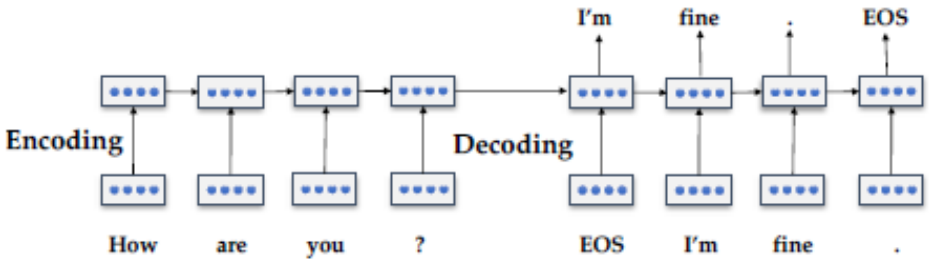


Figura 3.1: Un modello Seq2Seq applicato agli agenti di dialogo.

I chatbot con questa architettura sono molto promettenti e attualmente sono oggetto di studi e ricerche, per via del forte interesse attorno a questa materia; tuttavia sono difficili da addestrare e presentano alcuni difetti: i modelli prodotti si focalizzano nel generare singole risposte, perciò tendono a non comportarsi molto bene in caso di lunghe conversazioni e, in taluni, si ottengono risposte che generiche, che bloccano la conversazione, come ad esempio "Non lo so" o "Sto bene".

## 3.3 Task-oriented chatbot

I chatbot **task-oriented** (o goal-oriented) sono una tipologia degli agenti di dialogo progettati per svolgere attività ben precise e hanno lo scopo di condurre l'utente al raggiungimento di un obbiettivo, come l'ottenimento di informazioni o l'espletamento di un'azione. Fanno parte di questa categoria tutti gli assistenti digitali (tra cui Siri, Google Now, Cortana e Alexa) che consentono di realizzare compiti ben precisi e definiti, come chiedere informazioni (indicazioni stradali, previsioni metereologiche, etc.), far svolgere azioni ad altre applicazioni (chiamate, messaggi, memorizzazione promemoria, etc.) o interagire con altri oggetti interconnessi (smart devices).

### 3.3.1 Architettura

Questi chatbot generalmente sono implementati seguendo la seguente struttura, composta da tre componenti:

- **natural language understanding** (NLU): consiste nella comprensione del messaggio dell'utente e include le attività di **intent classification** e di **slot filling**. Gli intent corrispondo alle intenzioni dell'utente, ovvero alle richieste ricevibili e comprensibili dall'agente di dialogo. Gli slot invece corrispondono alle variabili, di cui è necessario conoscere il valore per completare il task. Ad esempio, a seguito dell'analisi della frase

"Vorrei ordinare una pizza margherita per le 19:30", l'intent ottenuto rappresenterebbe la volontà di ordinare una pizza, mentre gli slot riempiti sarebbero quelli relativi alla tipologia della pizza e l'orario di consegna. Generalmente l'intent classification e lo slot filling sono implementati con delle reti LSTM.

- **dialog management** (DM): si divide in due componenti, **dialog state tracker** e **dialog policy**. Il primo si occupa di mantenere lo stato della conversazione, aggiornandolo ad ogni messaggio dell'utente, comprendendo anche gli slot ottenuti in fase di NLU. Il secondo componente invece sulla base dell'andamento della conversazione sceglie la prossima azione da compiere, cercando di capire quale sia la migliore per il completamento del task nel minor numero di turni.

- **natural language generation** (NLG): è il processo della generazione della risposta, che costruisce una frase in linguaggio naturale contenente le informazioni che si vogliono comunicare all'utente. Spesso questo componente è di tipo rule-based: in questi casi esso dispone di una serie di frasi template, da completare con le informazioni opportune.

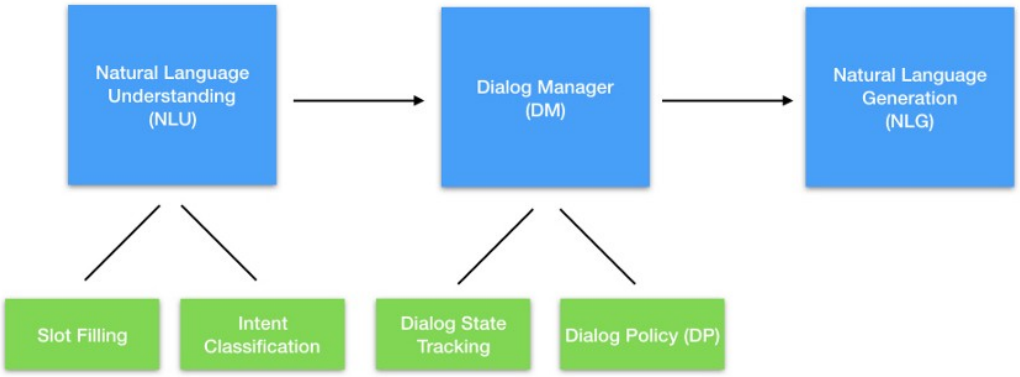


Figura 3.2: Architettura dei chatbot task-oriented [4].

# Capitolo 4

# Progetto

Nei capitoli precedenti si è analizzato lo stato dell'arte, approfondendo temi del natural language processing, le tecniche di machine learning e deep learning principalmente usate (con particolare attenzione alle loro applicazioni nel NLP) e infine la storia dei chatbot e le architetture principali utilizzate nel loro sviluppo.
In questo capitolo sarà esposto il progetto realizzato, anzitutto descrivendo le caratteristiche e le specifiche dell'applicazione, il framework utilizzato (**Rasa**), l'architettura dell'agente di dialogo e gli aspetti principali concernenti la sua progettazione e il suo sviluppo.

## 4.1 Analisi dei requisiti

Si vuole realizzare un chatbot per gli studenti iscritti al **Campus di Cesena** dell'Università di Bologna, ossia un software capace di fornire, mediante linguaggio naturale, tutte quelle informazioni utili agli studenti iscritti ai corsi di studio afferenti a questo campus. In particolare, si vogliono fornire, su richiesta dell'utente, informazioni in questi ambiti:

- **corsi di studio** del campus;
- **bandi** rivolti agli studenti iscritti (borse di studio, mobilità internazionale, assegni di tutorato, etc.).

In merito ai corsi di studio il chatbot deve poter fornire i link alle risorse web (ogni corso di studio dell'Università di Bologna ha un proprio sito web) tra cui: scheda del corso (homepage del sito), piano di studi, orario delle lezioni, calendario didattico e appelli di esame. Per quanto riguarda la ricerca dei bandi invece, compito del chatbot è inviare i link da cui poter scaricare i bandi più pertinenti con la richiesta dell'utente. Inoltre, possono essere richieste

informazioni sulla segreteria studenti (contatti, ubicazione, etc.).
Il chatbot deve essere in grado di comprendere queste richieste degli utenti in linguaggio naturale e, per agevolare l'utente, deve anche poter gestire forme di interazione base, come salutare e spiegare quali informazioni è in grado di fornire.

## 4.2 Rasa

Rasa, il framework scelto per lo sviluppo del chatbot, è open source ed è principalmente basato su machine learning. Ideato per lo sviluppo di agenti di dialogo testuali e vocali, è uno strumento molto efficace, che offre supporto per la realizzazione di bot in grado di svolgere conversazioni all'interno di un determinato contesto e task-based.

### 4.2.1 Architettura

Per comprendere il funzionamento di Rasa, attraverso tutti i passaggi che vanno dalla comprensione dei messaggi dell'utente alla generazione delle risposte, diamo uno sguardo alla sua architettura, che è ben raffigutata dalla figura 4.1. L'insieme di operazioni svolte può essere descritto dai seguenti passi [16]:

1. il messaggio è inizialmente analizzato dall'interprete che classifica l'intent e estrae le entità corrispondenti;

2. il risultato prodotto dall'interprete viene poi memorizzato nel tracker che è l'oggetto adibito a tenere traccia dello stato della conversazione;

3. il componente policy riceve lo stato del tracker;

4. e sceglie qual è l'azione da intraprendere in base all'ultimo messaggio ricevuto e allo storico precedente;

5. l'azione scelta viene immagazzinata nel tracker;

6. l'utente riceve una risposta generata sulla base dell'azione del passaggio precedente.

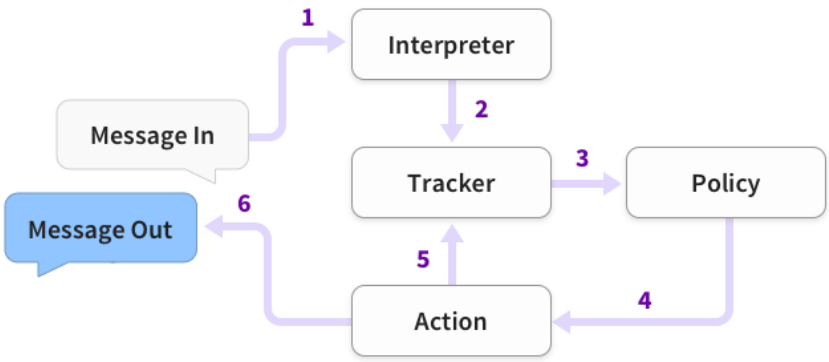


Figura 4.1: Architettura ad alto livello del framework Rasa.

Questa è in astratto la struttura di Rasa; in pratica questo framework è costituito principalmente da due componenti, che possono anche essere utilizzati in modo indipendente, Rasa NLU e Rasa Core.

### 4.2.2 Rasa NLU

**Rasa NLU** si occupa della comprensione dei messaggi dell'utente e dell'estrazione delle informazioni necessarie per poter proseguire la conversazione e far svolgere all'agente la propria funzione.
I suoi compiti principali consistono nella classificazione degli **intent** e nell'estrazione delle **entità** rilevanti all'interno del dominio contenute nel messaggio dell'utente. Per realizzare ciò viene addestrato un modello, per cui è necessario definire preventivamente gli intent e le entità ammissibili, e soprattutto fornire un sufficiente numero di esempi di frasi annotate.
Il testo però, prima di essere elaborato dal modello sopra descritto, viene preprocessato da una serie di componenti, che rendono il testo più facilemente elaborabile. Questi elementi di pre-processing, uniti agli altri (entity extraction, intent classifier ed altri eventuali) compongono l'intera pipeline, dalla quale viene poi generato l'intero modello ai quali verranno fatti analizzare i messaggi dell'utente.

### 4.2.3 Rasa Core

**Rasa Core** si occupa della gestione del dialogo, consentendo di portare avanti la conversazione. Ad esso spetta la scelta delle giuste **azioni** da intraprendere in seguito ai singoli messaggi ricevuti, e la loro esecuzione.

Le politiche con cui vengono scelte le azioni da intraprendere spesso si fondano su modelli probabilistici (ad esempio reti neurali, in particolare LSTM); le azioni invece sono definite in fase di sviluppo e possono basarsi su template, ovvero modelli predefiniti di messaggi, o implementare azioni personalizzate, che possono contenere informazioni ottenute da un database o richieste a API terzi.

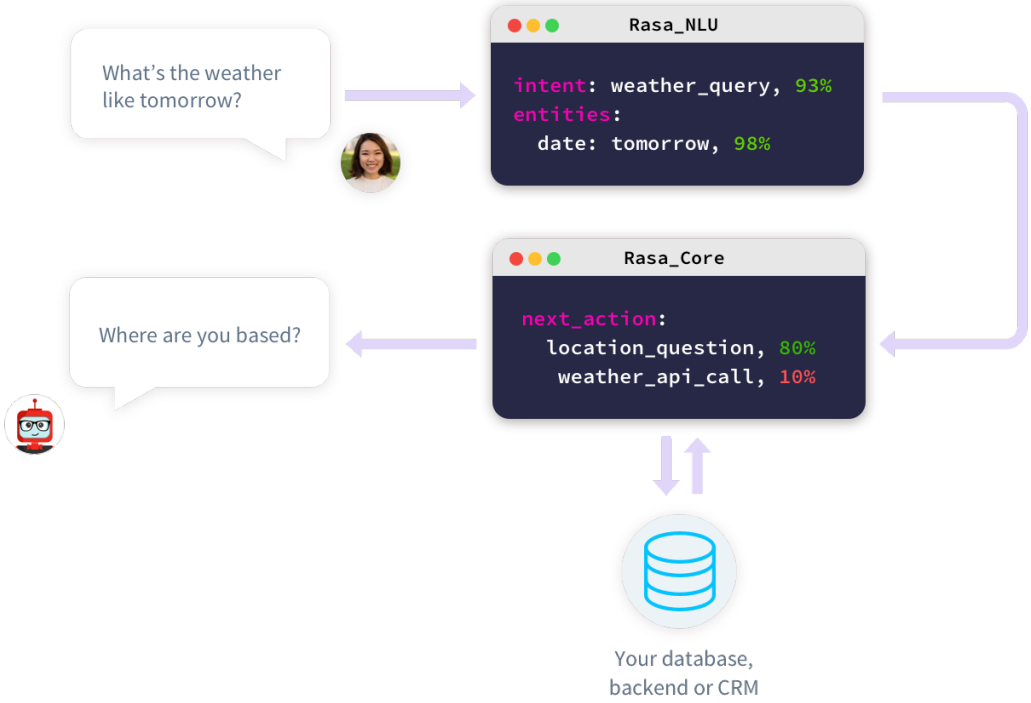


Figura 4.2: Esempio del comportamento di Rasa NLU e Rasa Core.

### 4.2.4 Installazione del framework

Installare Rasa è molto semplice, è sufficiente eseguire il seguente comando da terminale:

```
$ pip install rasa
```

In questo modo si dispone del framework, a cui possono essere aggiunti diversi componenti conformemente alle necessità. Nel caso in questione, è stato necessario installare una libreria per svolgere preprocessing e, in generale, task utili al componente di NLU, ovvero **Spacy** il quale offre supporto anche per la lingua italiana:

```
$ pip install rasa[spacy]
$ python -m spacy download it_core_news_sm
$ python -m spacy link it_core_news_sm it
```

Nello specifico il modulo **it_core_news_sm** è un modello pre-allenato per task in ambito NLP, allenato su un corpus in lingua italiana.

## 4.3 Scraping e memorizzazione dei dati

Per lo sviluppo di questo sistema è necessario attingere ad un insieme di informazioni del dominio applicativo disponibili online sui siti web dei corsi di studio e sul sito `https://bandi.unibo.it/` per quanto concerne i bandi di Ateneo. Per estrarre questi dati è stato impiegato il linguaggio di programmazione Python e, col fine di automatizzare al meglio quest fase, si è fatto affidamento su Open Data, Beautiful Soup e Selenium, che saranno approfonditi nei paragrafi successivi.

### 4.3.1 Open Data

**Open Data** (`https://dati.unibo.it`) è il portale online di Unibo che dà libero accesso a dati sui corsi di studio, sugli insegnamenti e su altri aspetti relativi alla didattica e all'Ateneo [17]. Questo servizio, attualmente in versione beta, mette a disposizione un'API consentendo di effettuare anche query complesse sui dati.
Nel progetto questo strumento è stato usato sia per ottenere la lista degli indirizzi dei siti web di tutti i corsi di studio del campus di Cesena, sia per ottenere la lista degli indirizzi alle pagine web degli insegnamenti di ogni corso di studio.

### 4.3.2 Beautiful Soup e Selenium

**Beautiful Soup** è una libreria python per estrarre dati da risorse **HTML** ed XML. Esso fornisce una serie di funzioni che, mediante tecniche di **parsing**, consentono di estrarre contenuti sulla base dei tag e degli attributi. Nella fattispecie, per quanto riguarda le risorse HTML, è possibile ricercare elementi che abbiano determinate caratteristiche (tag, classi, id) nonché navigare nel DOM di documenti HTML [18].
Beautiful Soup è uno strumento molto utile, ma non è stato sufficiente per completare l'attività di scraping. Questo perché i siti dei corsi di studio, così come molte applicazioni web al giorno d'oggi, caricano i propri contenuti in

modo asincrono, per velocizzare il caricamento della pagina. Questo impedisce di ottenere l'intera risorsa HTML con una sola richiesta http sincrona, perciò ad esso è stato abbinato **Selenium**.
Selenium è un framework nato per testare le applicazioni web [19]; il suo componente che è stato impiegato è webdriver, il quale consente di navigare tra le pagine web appoggiandosi su un browser, nel mio caso **Firefox**. Così facendo è stato possibile creare delle sessioni attraverso le quali caricare l'intero contenuto delle pagine web, dopodiché analizzarle con Beautiful Soup per estrarre le informazioni da ricercare.

### 4.3.3 Dati raccolti

I dati raccolti sono stati memorizzati in file in formato **JSON**, ed essi si compongono delle seguenti informazioni:

- 16 corsi di laurea triennali, magistrali e magistrali a ciclo unico: sono tutti i corsi di laurea erogati in lingua italiana presenti nel Campus di Cesena, e per ciascuno di essi si sono memorizzati i link alla homepage, al piano di studi, all'orario delle lezioni, al calendario didattico, agli appelli di esame e la lista degli insegnamenti (nome e link alla sua scheda);
- 38 bandi di Ateneo: sono l'insieme dei bandi reperibili online, nelle categorie di maggior interesse per gli studenti (ovvero quelli in merito alle borse di studio, mobilità internazionale, premi di laurea, assegni di tutorato, etc.); di essi sono stati memorizzati il titolo, il link e il contenuto testuale.

## 4.4 Architettura del chatbot

Dopo aver estratto e catalogato tutte le informazioni necessarie al funzionamento del chatbot, è seguita la fase di progettazione e sviluppo dello stesso. Il chatbot realizzato rientra nella categoria dei **task-oriented chatbot** e si riferisce ad un particolare dominio (quello universitario), perciò il primo passo è stato definire il dominio, mediante gli strumenti messi a disposizione dal framework (ovvero intent, entità e azioni).

### 4.4.1 Intent

Come già spiegato precedentemente, gli intent corrispondono alle intenzioni dell'utente nella comunicazione, ovvero alle tipologie di messaggi o richieste che l'utente può fare. Per questo chatbot sono stati definiti i seguenti intent:

- *greet*: il saluto per iniziare la conversazione
- *ask_bot_task*: per domandare al chatbot cosa è in grado di comunicare, a cosa può rispondere e, più in generale, informazioni sul suo dominio applicativo
- *ask_information*: per richiedere al chatbot informazioni sui corsi di laurea (piano di studi, calendario, etc.)
- *ask_announcement*: per svolgere ricerche sui bandi di Ateneo
- *info_student_office*: per chiedere informazioni riguardanti la segreteria studenti
- *thank*: ringraziamento
- *goodbye*: saluto per terminare la conversazione

### 4.4.2 Slot

Gli slot sono la memoria del chatbot, ovvero le informazioni, che l'utente fornisce durante la conversazione, necessarie per soddisfare le richieste; possono essere di vario tipo (boolean, categorical, float, text, list). In questo progetto gli slot definiti sono necessari per soddisfare le diverse richieste in merito ai corsi di laurea. Essi sono:

- *tipo_laurea*: la tipologia del corso di studi (triennale, magistrale o magistrale a ciclo unico)
- *corso*: il nome del corso di laurea del campus di cui si vuole avere informazioni
- *tipo_info*: il tipo di risorsa a cui si vuole accedere di un determinato corso di laurea (scheda del corso, piano di studi, orario accademico, calendario didattico, appelli di esame)
- *insegnamento*: il nome dell'insegnamento di un determinato corso di laurea

### 4.4.3 Pipeline NLU

La pipeline è un insieme di componenti che processano il testo ricevuto in input per consentire la classificazione degli intent e l'estrazione di eventuali entità. Rasa mette a disposizione già due pipeline preconfezionate, che svolgono

questo compito: *spacy_ sklearn* e *tensorflow_ embeddings*. La scelta tra le due dipende soprattutto dal tipo di dataset che si ha a disposizione: la prima è sicuramente più adatta nel caso non si disponga di un dataset di enormi dimensioni, dato che fa uso di rappresentazioni vettoriali del testo pre-allenate mentre la seconda genera word embedding dal solo dataset fornito per il dominio. Sebbene queste due pipeline siano consigliate e generalmente ben funzionanti per ogni tipo di chatbot, è possibile personalizzare le suddette pipeline, definirne di nuove ed anche integrarle con nuovi componenti creati su misura.
La pipeline impiegata è stata definita inserendo i componenti di spacy_sklearn e un componente personalizzato, posto all'inizio della stessa. Quest'ultimo, chiamato *FetchSimilarEntities*, ha lo scopo di aiutare ad intercettare le entità anche quando sono inserite dall'utente con degli errori di battitura. In caso contrario, il componente addetto all'estrazione delle entità sarebbe in grado di riconoscere i corsi di laurea solo se inseriti correttamente; essendo i nomi dei corsi di laurea spesso molto lunghi si è ritenuto necessario quest'operazione per facilitare la prosecuzione della conversazione. Questo componente aggiuntivo migliora quindi il riconoscimento degli slot, e lo fa basandosi sulla **distanza di edit**.
Gli ultimi componenti della pipeline invece, sono l'estrattore delle entità e il classificatore degli intent che vanno allenati per lo specifico dominio di applicazione.

### 4.4.4 Storie

Le storie sono rappresentazioni di conversazioni tra utente e agente di dialogo e servono per insegnare al modello addetto al dialog management quale sia la migliori azione da intraprendere in una determinata situazione. Le storie rappresentano i messaggi dell'utente come intent (comprensivi di eventuali slot) e le risposte dell'assistente come azioni. Eccone un esempio:

```
##story_example
* greet
  - utter_greet
* ask_announcement
    - action_give_announcements
* thank
    - utter_you_are_welcome
```

In questo caso l'utente inizialmente saluta, ricambiato dal chatbot; successivamente si chiedono informazioni su un bando e infine l'utente chiude la conversazione ringraziando il chatbot.
Per generare storie realistiche è possibile impiegare la modalità di **interac-**

**tive learning** di Rasa, nella quale si svolge una conversazione col chatbot etichettando i messaggi inviati (indicando intent e slot) e scegliendo, dopo ogni messaggio, le azioni che dovrebbero essere intraprese. Con questa funzionalità è possibile esportare le conversazioni svolte, rappresentandole sotto forma di storie.

### 4.4.5 Azioni

Le azioni corrispondono alle tipologie di interazioni che il chatbot ha con l'utente. Dopo aver ricevuto un messaggio viene classificato l'intent e individuati gli slot, dopodiché viene scelta l'azione più adatta da svolgere dal dialog manager. Ci sono vari tipi di azioni:

- **utterance actions**: consistono in risposte predefinite da utilizzare in determinate situazione (ad esempio nel saluto). Possono essere definite tramite template, ovvero tramite frasi che contengono delle variabili che assumono un valore diverso in base al contesto. Di seguito vi sono alcune delle utterance actions usate nel progetto, il cui nome inizia con "utter_" e per ognuna delle quali è inserita una o più alternative di possibili risposte:

```
utter_greet:
- text: "Ciao, come posso esserti d'aiuto?"
    buttons:
    - title: "Vuoi che ti spieghi cosa puoi chiedermi?"
      payload: '/ask_bot_task'
    - text: "Ciao, cosa posso fare per te?"
    buttons:
    - title: "Vuoi che ti spieghi cosa puoi chiedermi?"
      payload: '/ask_bot_task'

utter_who_am_i:
- text: "Sono un chatbot che aiuta gli studenti del Campus di
    Cesena a trovare informazioni sui corsi di laurea (scheda
    del corso, piano di studi, orario accademico, calendario
    didattico) e sui loro insegnamenti. Posso anche cercare per
    te tra i bandi dell'universita' rivolti agli studenti
    iscritti (borse di studio, mobilita' internazionale,
    tutorato, etc.) e darti informazioni sulla segreteria
    studenti."

utter_you_are_welcome:
- text: "Prego"
```

```
- text: "Non c'e' di che"

utter_student_office:
- text: "Informazioni segreteria:\nOrario di apertura:\nLunedi',
    martedi', mercoledi', venerdi': ore 9:00-11.15\nMartedi',
    giovedi': ore 14:30-15:30\nTelefono: 0547 338850\nEmail:
    segcesena@unibo.it\nUbicazione: via Montalti, 69."

utter_default:
- text: "Scusa non ho ben capito cosa intendi, potresti
    ripetere?"
```

- **custom actions**: sono azione completamente personalizzabili, che consentono altresì di eseguire codice sorgente. Ad esempio possono consistere nell'esecuzione di query in un database per ricercare le informazioni da inviare all'utente. Nel progetto l'unica custom action presente è denominata *action_give_announcements*, impiegata per la ricerca dei bandi di Ateneo. Per lo sviluppo di questa azione è stato impiegato **LSA** (Latent Semantic Analysis, che verrà approfondito in seguito), come metodo per la ricerca dei bandi secondo la query inserita.

- **default actions**: sono azioni preconfigurate, fornite da Rasa, tra cui:

  - *action_listen*: si ferma la predizione di altre azioni e si attende un messaggio dall'utente
  - *action_restart*: riavvia la conversazione
  - *action_default_fallback*: viene attivata quando la confidenza nella classificazione dell'intent o quella nella predizione della prossima azione non superano determinate soglie (**Fallback Policy**). In questo caso viene inviato come messaggio *utter_default*.

### 4.4.6 Form

Il form è molto simile and una custom action e viene preferito ad essa nel caso in cui sia necessario il riempimento di determinati slot per il soddisfacimento della richiesta. Infatti, quando l'esecuzione dell'azione necessita di determinate informazioni dall'utente, i form sopperiscono a questa lacuna eseguendo un'azione che chiede le informazioni mancanti.
Nel progetto è stato definito un form, denominato *give_information_form*, che si occupa di ottenere le informazioni richieste in merito ai corsi di laurea

del campus. Questa scelta è giustificata dal fatto che lo slot *corso* è assolutamente necessario per poter soddisfare la richiesta. Si propone di seguito l'implementazione del form.

```
class giveInformationForm(FormAction):

    def name(self) -> Text:
        return "give_information_form"

    @staticmethod
    def required_slots(tracker: Tracker) -> List[Text]:
        """A list of required slots that the form has to fill"""
        return ["corso"]

    def submit(self, dispatcher: CollectingDispatcher, tracker:
        Tracker, domain: Dict[Text, Any]) -> List[Dict]:
        """Define what the form has to do after all required slots are
            filled"""
        #code for satisfaying the request. Executed only if the slot
            'corso' is already filled
```

Il prossimo riquadro invece rappresenta l'azione eseguita se viene richiesta l'esecuzione del form nel caso lo slot richiesto non sia stato riempito.

```
utter_ask_corso:
- text: "Potresti dirmi precisamente su quale corso di laurea
    vorresti informazioni?"
```

## 4.5 Metodi per la ricerca dei bandi di Ateneo

Per la funzionalità di ricerca dei bandi di Ateneo è stato impiegato il **Latent Semantic Analysis** (LSA), un metodo di rappresentazione dei documenti che consente di valutarne la similarità.
Altri possibili metodi di rappresentazione che si sarebbero potuti impiegare sono **bag-of-words** e **tf-idf** (term frequency - inverse document frequency). Questi due metodi prevedono entrambi di rappresentare dei documenti con vettori di dimensione N, dove N è il numero di parole presenti nei del vocabolario. La differenza tra i due è che le bag-of-words rappresentano il documento con un vettore contenente le frequenze delle parole, mentre tf-idf assegnano alle frequenze un peso che dipende dalla frequenza di quella parola in tutti gli altri documenti.
Fra i due metodi sopracitati tf-idf è sicuramente più efficace, proprio perchè

evidenza quali sono i termini più caratterizzanti di ogni documento, che assumono un peso maggiore nella rappresentaione dei singoli testi. La limitazione di entrambi invece è che si basa solo sulle frequenze delle parole e non sul loro significato. Ad esempio non tengono conto di due fenomeni linguistici, appartenenti al livello semantico:

- **sinonimia**: parole diverse possono assumere lo stesso significato o significati simili;
- **polisemia**: una stessa parola può assumere più significati.

Invece LSA non fa uso solo delle mere frequenze, ma è usato per rappresentare documenti tenendo conto anche delle relazioni semantiche tra le parole, rendendo più efficace anche la ricerca di documenti.

### 4.5.1 Latent semantic analysis

Il latent semantic analysis è una tecnica di information retrieval utilizzata per analizzare relazioni tra insiemi di documenti che mira a individuare i concetti e relazioni semantiche latenti presenti all'interno degli stessi [20].
LSA si basa sul principio che la semantica di una parola è fortemente legata al contesto in cui appare, cioè parole diverse che compaiono in contesti molto simili potrebbero essere simili e, al contrario, una stessa parola usata in contesti totalmente diversi potrebbe essere un indizio del fatto che essa assuma più significati. Dunque, studiando i contesti in cui le parole si trovano, cerca di individuare le relazioni tra di esse e tra i documenti, ma soprattutto gli argomenti e gli ambiti in cui si articolano i testi in questione.
Il meccanismo che conferisce a LSA queste caratteristiche è il **singular value decomposition (SVD)**, una tecnica di fattorizzazione di matrici; data una matrice M reale o complessa (nella fattispecie una **term-document matrix**) di dimensione $n \times m$ essa può essere rappresentata dal prodotto di tre matrici:

$$M = U\Sigma V^T$$

dove U (**SVD term matrix**) è la matrice composta dagli autovettori della matrice $MM^T$ dimensioni $n \times k$, $\Sigma$ è una matrice diagonale di dimensioni $k \times k$ composta dai valori singolari ordinati in ordine e $V^T$ (**SVD document matrix**) è la trasposta della matrice V composta dagli autovettori della matrice $M^TM$ di dimensioni $k \times m$. Il fattore di decomposizione k, che corrisponde alla grandezza del nuovo spazio utilizzato per rappresentare frasi e documenti, deve essere scelto in modo da ottimizzare le prestazioni e dipende dalla quantità di dati a disposizione e dal loro contenuto.
Con questo meccanismo avviene dunque una riduzione della dimensionalità che serve a far emergere relazioni semantiche latenti tra parole e documenti.

### 4.5.2 Implementazione LSA

Come precedentemente esposto è possibile creare il modello di LSA, col quale si rappresentano query e documenti nello stesso spazio e si possono calcolare le similarità tra essi, ottenendo i documenti più pertinenti. Ciò, nel progetto è stato implementato utilizzando la matrice U, che rappresenta i termini nel nuovo spazio di dimensione k: con essa è possibile creare vettori che rappresentino le query e i documenti da cercare, dopodichè applicando la **similarità coseno** si ottiene un indice (che va da 0 a 1) che consente di selezionare i bandi maggiormente correlati alla query.
Le principali scelte fatte nell'implementazione di questo componente di ricerca dei bandi sono state due: la prima riguarda il valore di decomposizione k (pari a 12), che è stato definito empiricamente, esaminando con quale valore si otteneva una migliore precisione; la seconda scelta riguarda la selezione dei risultati, basandosi sulle valutazioni ottenute con LSA. Per quest'azione si è optato di scartare i documenti con una similarità inferiore ad una soglia minima predefinita (pari a 0,85), sotto alla quale si può assumere che un documento non sia rilevante (questo aver svolto una normalizzazione sui valori delle similarità dei documenti).
Nell'impiego di questa funzione di ricerca nel chatbot si è infine deciso di mostrare non più di sette risultati per non appesantire eccessivamente la chat.

### 4.5.3 Valutazione LSA

Le prestazioni di questo componente sono state testate su un insieme di 33 query, di cui sono stati definiti i relativi documenti correlati. In questo test la precisione si è attestata a 0,524. Ciò che è emerge dai test è che il modello si comporta bene nell'estrazione di documenti relativi ad alcuni argomenti, ad esempio quelli che hanno maggior rappresentanza nel corpus dei bandi a discapito di quelli meno rappresentati. Inoltre un fattore importante è dato dalla composizione della query, da quanto essa è specifica e dalla eventuale presenza di termini distintivi.

## 4.6 Addestramento e configurazione del chatbot

Dopo aver definito il dominio e aver implementato le funzioni di estrazione delle informazioni dei corsi di laurea e di ricerca dei bandi è possibile addestrare l'agente di dialogo, il quale in questa fase impara a estrarre le entità, riconoscere gli intent e a decidere le azioni da intraprendere.

### 4.6.1 Dataset

Il dataset si compone di 218 frasi etichettate, ovvero esempi di possibili messaggi che l'utente potrebbe inviare di cui si conosce intent ed eventuali entità. Esso è stato diviso in due componenti:

- **training set**: composto da 166 esempi, impiegato per l'addestramento del componente di NLU;
- **test set**: composto da 52 esempi, impiegato per valutare il modello ottenuto.

Di seguito si propone un frammento del file contenente il dataset.

```
## intent:ask_information
- ciao, vorrei la [scheda](tipo_info) del corso di laurea
    [triennale](tipo_laurea) in [ingegneria e scienze
    informatiche](corso)
- salve, potrei visionare il [calendario](tipo_info) del corso di
    laurea [magistrale](tipo_laurea) in [psicologia cognitiva
    applicata](corso)?
- mi fai vedere l'[orario](tipo_info) delle lezioni del corso di
    [ingegneria biomedica](corso)?
- mandami il [piano didattico](tipo_info) del corso di [scienze e
    tecniche psicologiche](corso)?
- vorrei delle informazioni sul corso di laurea
    [magistrale](tipo_laurea) in [Neuroscienze e riabilitazione
    neuropsicologica](corso)
- inviami l'[orario](tipo_info) delle lezioni di [scienze e
    tecnologie alimentari](corso)?
- hai a disposizione il [piano didattico](tipo_info) della laurea
    [magistrale a ciclo unico](tipo_laurea) in [architettura](corso)
- vorrei saperne di piu' sull'esame di [estetica](insegnamento) del
    corso in [architettura](corso)
```

### 4.6.2 Addestramento ed esecuzione

Per eseguire la fase di addestramento è necessario eseguire il seguente comando:

```
$ rasa train
```

Dopo questo comando il chatbot è pronto per essere lanciato in esecuzione: a questo punto è necessario lanciare il server che si occupa di gestire le azioni personalizzate sviluppate in python, con il comando

```
$ rasa run actions
```

e lanciare l'applicazione, sempre da terminale, che consentirà di conversare con l'agente di dialogo

```
$ rasa shell
```

A questo punto il software è in esecuzione, in attesa di un messaggio da parte dell'utente: appena il messaggio viene ricevuto esso sarà preprocessato, saranno estratte le eventuali entità, verrà classificato l'intent e, infine, eseguita l'azione scelta dal dialog manager.

## 4.7 Valutazione e test del chatbot

Valutare un chatbot significa metterlo alla prova con il test set, ovvero dei dati non usati nella fase di addestramento, e verificare che il comportamento effettivo sia quello previsto. Per svolgere queste valutazioni esistono diversi indici, tra cui **accuracy** (accuratezza), **precision** (precisione), **recall** (richiamo) e **F-measure** (che coniuga precision e recall).
Rasa mette a disposizione dei comandi utili a testare il sistema e nei prossimi paragrafi si mostreranno le performance del componente di NLU e poi del chatbot nel complesso.

### 4.7.1 Valutazione del componente di NLU

Questo test serve a comprendere la qualità dei modelli di estrazione delle entità e classificazione degli intent, fondamentali per il corretto funzionamento dell'intera applicazione.
I risultati mostrati nelle tabelle 4.1 e 4.2 sono stati ottenuti usando il test set di cui si è parlato precedentemente, il quale è composto da 52 messaggi etichettati.

| **Entità** | **Precision** | **Recall** | **F-measure** |
|---|---|---|---|
| tipo_laurea | 1.0 | 1.0 | 1.0 |
| insegnamento | 1.0 | 0.583 | 0.737 |
| corso | 1.0 | 1.0 | 1.0 |
| tipo_info | 1.0 | 1.0 | 1.0 |
| **Media pesata** | 1.0 | 0.946 | 0.966 |

Tabella 4.1: Valutazione accuratezza nell'estrazione delle entità.

Come si può evincere dai risultati mostrati nella precedente tabella non sempre il modello riesce ad estrarre gli insegnamenti (la recall è pari a 0.583), sebbene in generale vengano riconosciute con facilità.

| **Intent** | **Precision** | **Recall** | **F-measure** |
|---|---|---|---|
| ask_bot_task | 1.0 | 1.0 | 1.0 |
| goodbye | 1.0 | 1.0 | 1.0 |
| ask_announcement | 0.875 | 0.7 | 0.778 |
| info_student_office | 1.0 | 1.0 | 1.0 |
| thank | 1.0 | 1.0 | 1.0 |
| ask_information | 0.8 | 0.941 | 0.865 |
| greet | 1.0 | 0.75 | 0.857 |
| **Media pesata** | 0.91 | 0.904 | 0.902 |

Tabella 4.2: Valutazione accuratezza nella classificazione degli intent.

Per quanto riguarda l'intent classification, i valori più critici sono la recall di ask_announcement (l'intent dei bandi) e la precision di ask_information (l'intent delle informazioni sui corsi di laurea), dai quali si evidenzia come a volte messaggi della prima classe vengano categorizzate come appartenenti alla seconda.
Va altresì detto che l'esiguità del test set impone certa cautela nella valutazione di questi dati: è sintomatico di ciò, ad esempio, il fatto che sia presente una grande quantità di indicatori pari a 1. Ciò nonostante è plausibile che in presenza di un dataset più voluminoso, questi indicatori possano anche essere migliorati.

### 4.7.2 Valutazione end2end

La valutazione **end2end** è una valutazione complessiva del chatbot, che mostra la qualità del funzionamento dell'intera applicazione e della corretta integrazione dei vari componenti che la compongono. In particolare questa si svolge definendo una serie di storie con i relativi messaggi dell'utente.
Ai fini di questo test sono state definite 35 storie (composte da uno o più messaggi ciascuna), usando tutte le frasi contenute nel test set; esse rappresentano esempi di conversazioni e si compongono di messaggi etichettati con le relative azioni che il chatbot dovrebbe intraprendere. Da questa valutazione si è in grado di stabilire quali conversazioni si sono svolte correttamente, ovvero se il chatbot svolge correttamente i seguenti passaggi:

- estrazione delle entità

- classificazione degli intent
- scelta dell'azione da intraprendere

Svolgendo questo test è emersa un'accuratezza pari all'**80%**, infatti ben 28 conversazioni delle totali 35 si sono svolte correttamente. Nel restante 20% delle conversazioni, ci sono stati uno o più errori da parte dei modelli in gioco, che ne hanno pregiudicato il corretto funzionamento.
Inoltre, questo test mostra anche le prestazioni del dialog manager nella predizione delle azioni. In totale le azioni da predire erano 128, maggiori rispetto alla dimensione del test set perché oltre alle azioni relative alle risposte vi sono da predire azioni accessorie quali action_listen e action_restart, necessarie per il corretto svolgimento della conversazione. Eccone i risultati ottenuti:

- accuracy è 0.88 (113 su 128);
- precision = 0.89;
- recall = 0.88;
- f-measure = 0.89.

### 4.7.3 Altre considerazioni

Come si è visto i risultati ottenuti dai test sono piuttosto positivi, sebbene vi sia una considerazione da fare in merito: il dataset non è di dimensioni sufficientemente grandi da garantire la robustezza al chatbot. Ciò è ancor più vero in applicazioni come queste dove l'utente si interfaccia mediante l'utilizzo del linguaggio naturale: infatti vi sono svariati modi di esprimere una richiesta e ognuno ha modi diversi di interloquire. Pertanto un'ottima rappresentatività è necessaria per l'affidabilità del sistema e poterlo impiegare su larga scala.

## 4.8 Rilascio su Telegram

Negli ultimi anni, di pari passo con la sempre maggior diffusione dei chatbot, servizi di messaggistica e social network (Telegram, Slack, Facebook Messenger, etc.) stanno introducendo modalità per supportare il deployment agenti di dialogo autonomi. Rasa, a sua volta, facilita l'integrazione con questi servizi, tra cui quello scelto in questo progetto, ossia **Telegram**.
Per fare ciò è anzitutto necessario creare un bot Telegram, ottenere le credenziali e inserirle nel file di configurazione adibito nel progetto. Successivamente, si deve mandare in esecuzione il server che gestirà le richieste del bot, con il comando

```
$ rasa run
```

Questo server riceve tutti i messaggi degli utenti e le gestisce; inoltre, assieme al server dedicato alle custom actions, elabora le risposte e invia il risultato di questo processo al bot Telegram.
Queste comunicazioni vengono effettuate mediante protocollo **HTTP**: per lo svolgimento di questo progetto si è scelto di eseguire i server in locale ed impiegare **ngrok**, il quale rende possibile la comunicazione con il bot, mediante la creazione di tunnel sicuri. Con ngrok è possibile generare un url che consenta dall'esterno di comunicare con applicazioni specifiche, definendone la porta.
Certamente ngrok non è una soluzione definitiva, ma è uno strumento adatto per testare sistemi e verificarne il corretto funzionamento, mentre una soluzione definitiva consisterebbe nel fare uso di un server fisico.

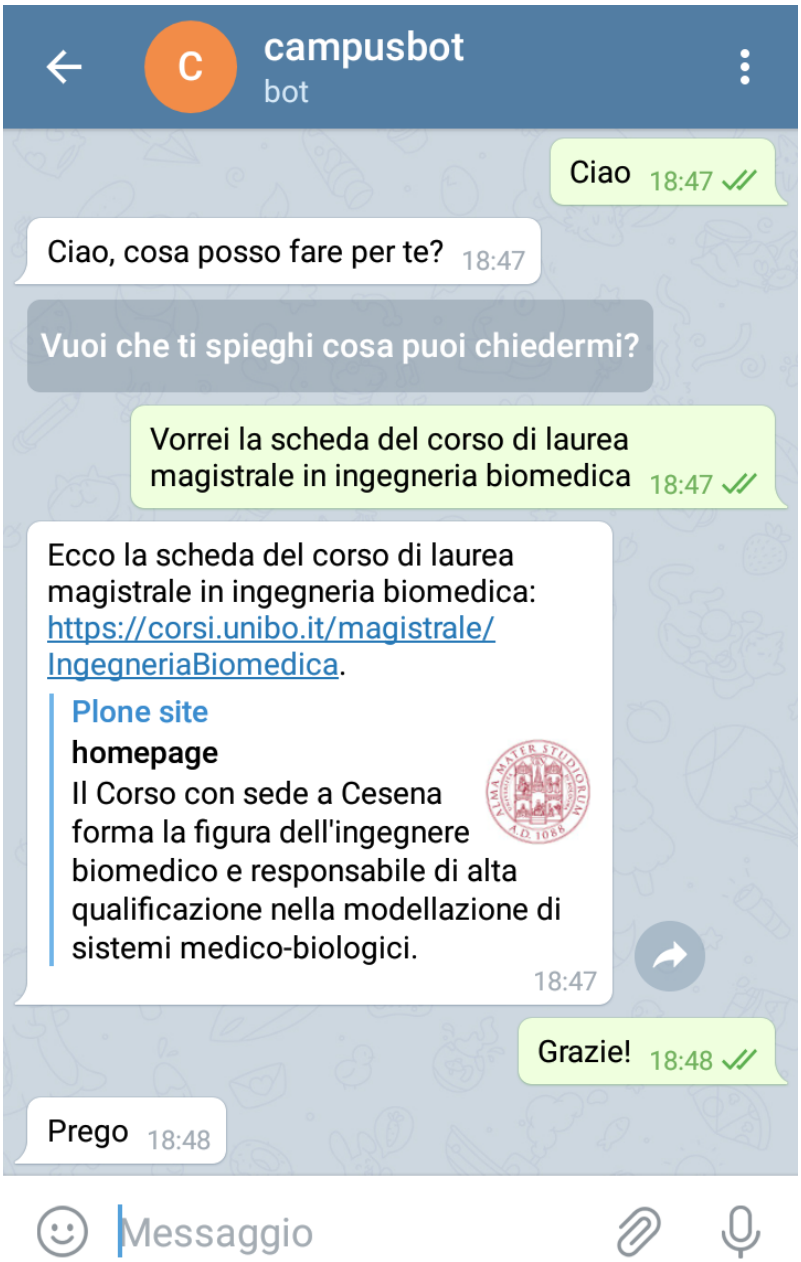


Figura 4.3: Esempio 1: una semplice richiesta riguardante informazioni su un corso di laurea.

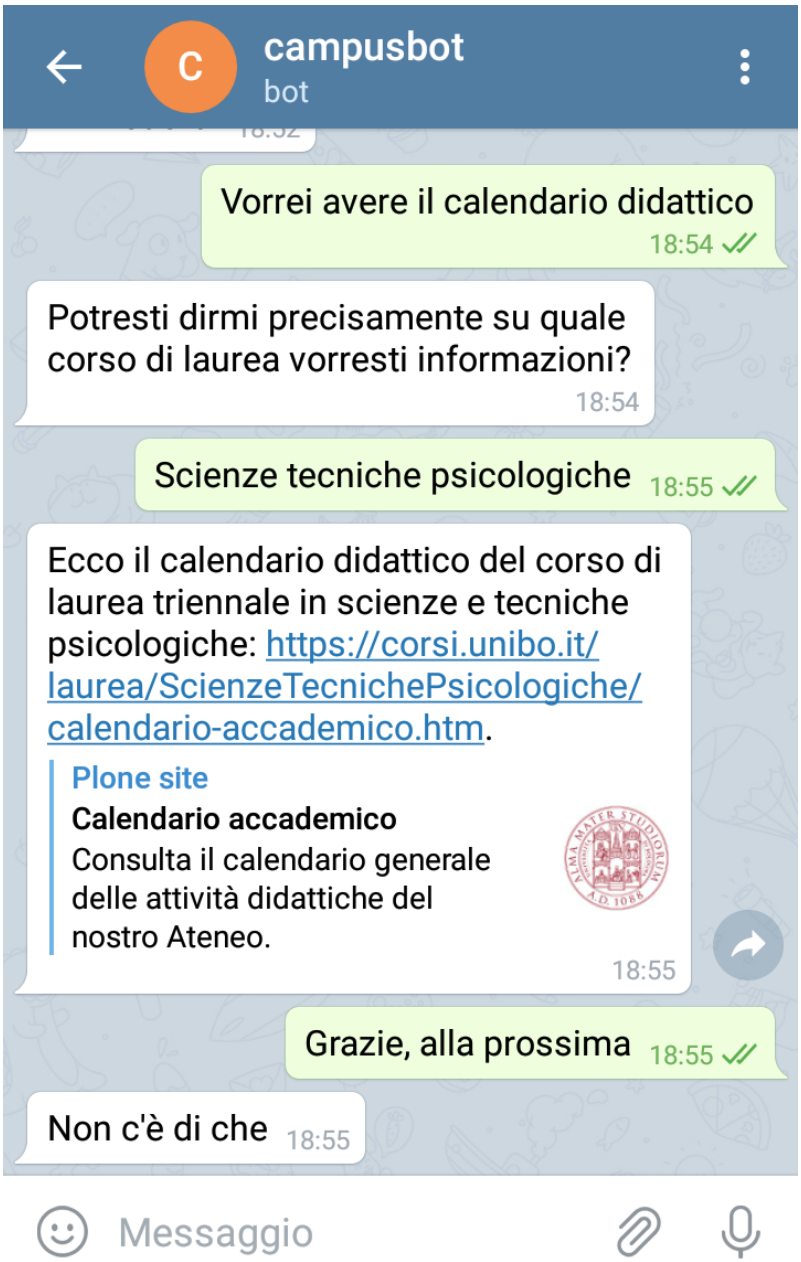


Figura 4.4: Esempio 2: si chiede un'informazione senza specificare il corso di laurea; il chatbot se ne capacita e lo chiede successivamente all'utente.

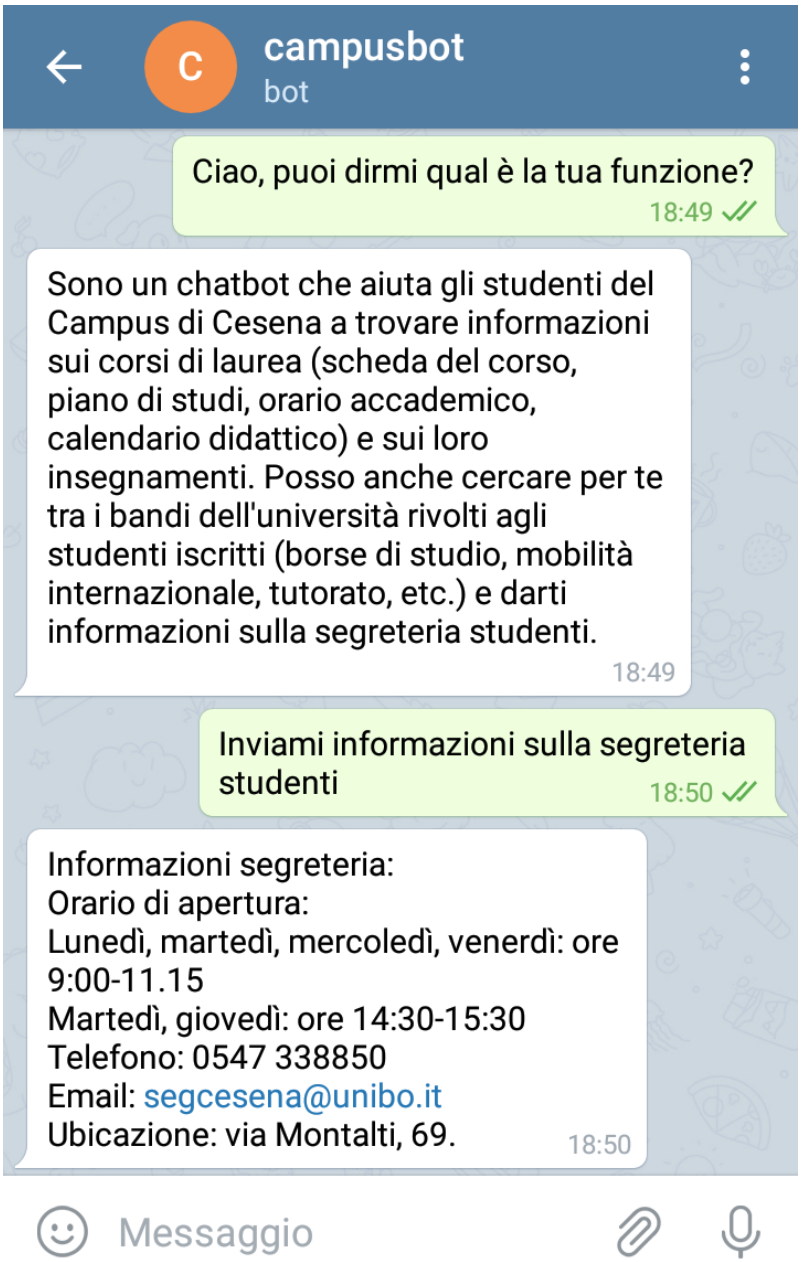


Figura 4.5: Esempio 3: si richiede la funzione del chatbot (si noti che è possibile conoscere ciò anche mediante il pulsante suggerito nell'esempio 1 dopo il saluto) e informazioni sulla segreteria.

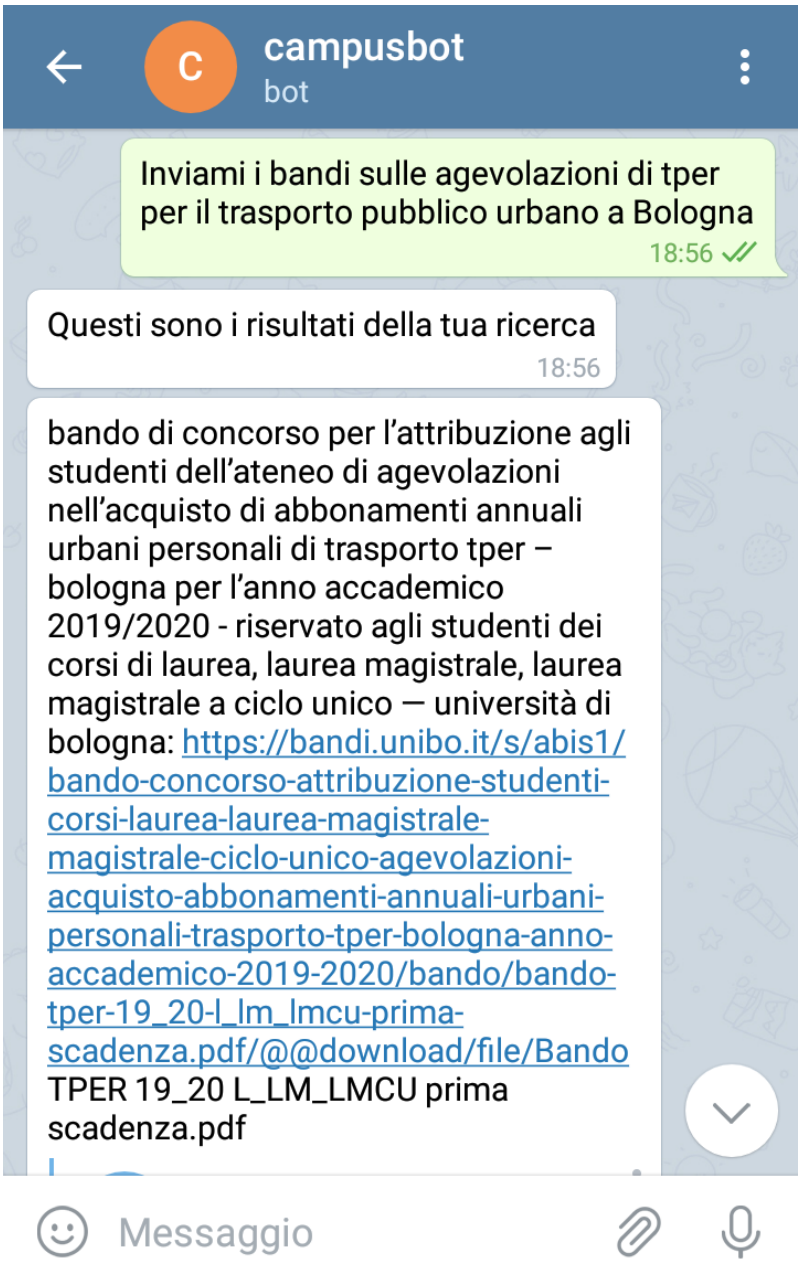


Figura 4.6: Esempio 4, parte 1: ricerca tra i bandi che produce come risultato i due bandi più pertinenti.

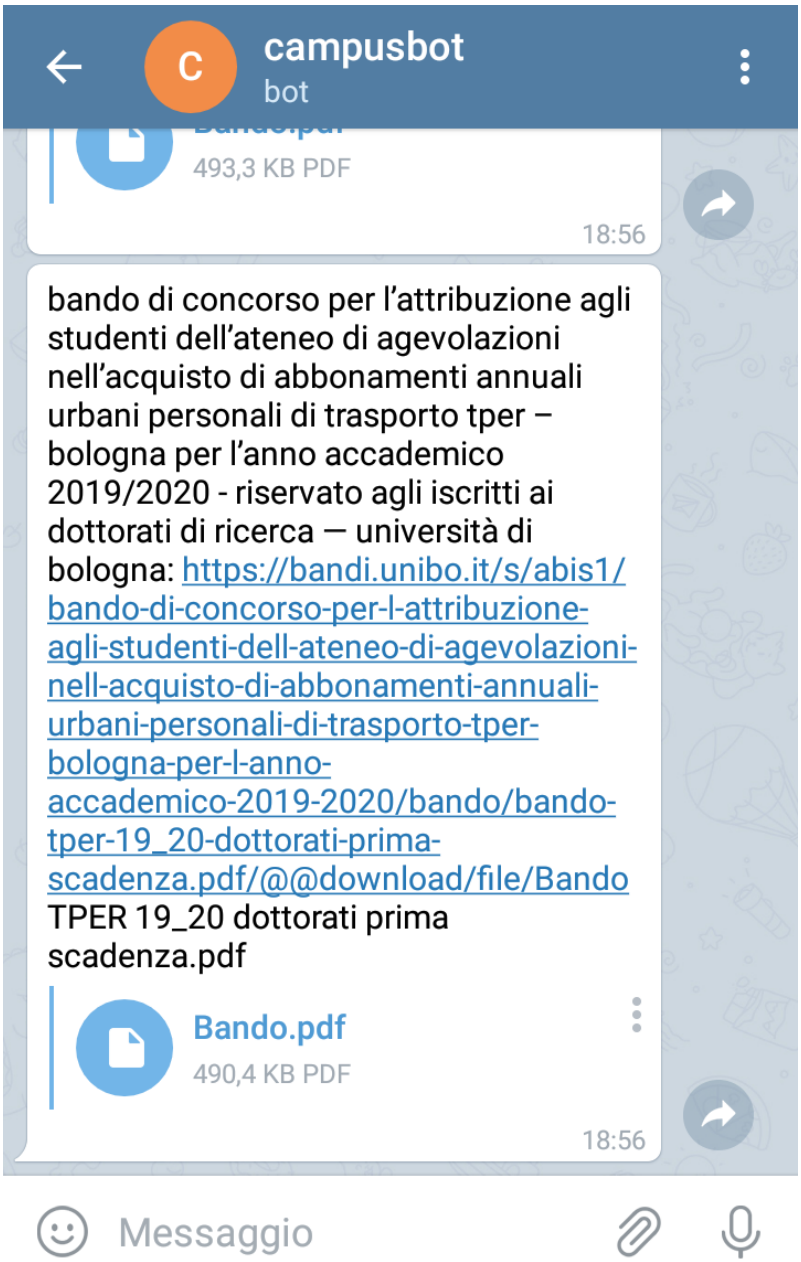


Figura 4.7: Esempio 4, parte 2.

# Conclusioni e sviluppi futuri

In questa tesi si sono affrontati i principali aspetti e tecnologie del natural language processing, in particolare in riferimento ai chatbot; dalla trattazione di questo tema sono emerse le difficoltà relative al trattamento automatico del linguaggio, evidenziando come problemi di questa natura richiedano approcci adeguati alla loro complessità, e come molta strada debba ancora essere fatta in questo ambito.
Si è inoltre presentata la realizzazione di un progetto pensato per un contesto universitario, nella fattispecie un chatbot che fornisse informazioni sui corsi di laurea del Campus di Cesena e dei bandi di Ateneo. Si è impiegato il framework Rasa che si è rivelato ottimo per la sua realizzazione, in quanto bene si adatta a chatbot che operano in un dominio specifico e ben definito. Sicuramente ci sono alcune pecche dovute alla complessità del dominio di applicazione e alla complessità intrinseca di applicazioni come questa, ma nel complesso il chatbot si comporta piuttosto bene, soprattutto se l'utente conosce le informazioni che è in grado di ricavare dallo stesso.
È altresì d'uopo sottolineare che potrebbero essere apportate alcune migliorie: ad esempio, per quanto riguarda la ricerca dei bandi, i risultati non sono così ottimi se le query contengono pochissime parole o parole poco distintive, quindi potrebbe essere utile provare altre tecniche e compararle con LSA, l'attualmente impiegata.
Inoltre si potrebbero incrementanre le funzionalità o estendere i corpus di dati per la fase di addestramento. Quest'ultima operazione potrebbe essere semplicemente implementata con il rilascio di una versione beta: in questo modo sarà possibile raccogliere una grande quantità di conversazioni reali che potranno essere usate per riaddestrare il chatbot, conferendogli maggior precisione ed efficacia.

# Ringraziamenti

Vorrei ringraziare tutti i professori incontrati in questo percorso formativo che, con il loro lavoro, mi hanno consentito di accrescere le mie competenze nella disciplina dell'informatica; in particolare il mio relatore, il prof. Gianluca Moro, che mi ha offerto interessanti spunti utili alla realizzazione del mio progetto di tesi.
Un altro grande ringraziamento va inoltre a tutte le persone che mi hanno accompagnato in questo percorso, dunque ai miei familiari e ai miei amici e a tutti coloro che mi hanno incoraggiato nel raggiungimento di questo importante obiettivo.
Infine vorrei altresì ringraziare tutto il personale dell'università, che rende possibile l'erogazione della didattica e dei servizi necessari per l'intero funzionamento dell'Ateneo.

# Bibliografia


[1] Descarte R. *Discorso sul metodo*. Editori Riuniti, 1637.

[2] Hemant Rakesh. Natural language processing, 2018. URL: https://becominghuman.ai/natural-language-processing-in-a-nutshell-a784b9fea849.

[3] Olga Davydova. 7 types of artificial neural networks for natural language processing, 2017. URL: https://medium.com/@datamonsters/artificial-neural-networks-for-natural-language-processing-part-1-64ca9ebfa3b2.

[4] WayBlazer. Task-oriented chatbots at wayblazer, 2018.

[5] Pirrelli V. Lenci A., Montemagni S. *Testo e Computer. Elementi di linguistica computazionale*. Carocci, 2016.

[6] Cerruti M. Berruto G. *La linguistica. Un corso introduttivo*. UTET Università, 2017.

[7] Martin J. H. Jurafsky D. *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson College, 2019.

[8] Michel Kana. Representing text in natural language processing, 2019. URL: https://towardsdatascience.com/representing-text-in-natural-language-processing-1eead30e57d8.

[9] Vibhor Nigam. Understanding neural networks. from neuron to rnn, cnn, and deep learning, 2018. URL: https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90.

[10] Christopher Olah. Understanding lstm networks, 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[11] Jason Brownlee. A gentle introduction to transfer learning for deep learning, 2017. URL: https://machinelearningmastery.com/transfer-learning-for-deep-learning/.

[12] Raj S. *Building Chatbots with Python. Using Natural Language Processing and Machine Learning.* Apress, 2018.

[13] Woebot faqs, 2019. URL: https://woebot.io/faqs/.

[14] Kunal Bhashkar. Build your own chatbot, 2019. URL: https://medium.com/@BhashkarKunal/conversational-ai-chatbot-using-deep-learning-how-bi-directional-lstm-machine-reading-38dc5cf5a5a3.

[15] Harshall Lamba. Word level english to marathi neural machine translation using encoder-decoder model, 2019. URL: https://towardsdatascience.com/word-level-english-to-marathi-neural-machine-translation-using-seq2seq-encoder-decoder-lstm-model-1a913f2dc4a7.

[16] Rasa docs. URL: https://rasa.com/docs/rasa/.

[17] Open data: ecco il portale dei dati aperti dell'università di bologna, 2017. URL: https://magazine.unibo.it/archivio/2017/12/04/open-data-ecco-il-portale-dei-dati-aperti-unibo.

[18] Beautiful soup documentation, 2019. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[19] Selenium webdriver, 2019. URL: https://www.seleniumhq.org/docs/03_webdriver.jsp.

[20] Raghavan P. Manning C. D. and Schütze H. *Introduction to Information Retrieval.* Cambridge University Press, 2019.