Documentazione

Progetto Brigid AI Chatbot

Catello Martone, Davide Viola, Gabriella Fede, Pasquale Anatriello

5 febbraio 2025

Prof. Fabio Palomba, Corso di Fondamenti di Intelligenza Artificiale
Università degli Studi di Salerno, A.A. 2024/2025

Indice

1 Introduzione

1.1 L'obiettivo della chatbot Brigid

La chatbot sviluppata ha l'obiettivo di fornire un supporto interattivo e intelligente agli utenti, con particolare attenzione all'assistenza psicologica e al dialogo empatico. Grazie all'integrazione di modelli NLP avanzati, come BERT per la comprensione del linguaggio naturale e un classificatore basato su reti neurali, il sistema è in grado di riconoscere gli intenti degli utenti e fornire risposte pertinenti. Inoltre, l'uso di un modello generativo come Gemini consente di gestire input complessi o non previsti, garantendo maggiore flessibilità nella conversazione. La chatbot è progettata per offrire interazioni fluide e contestualizzate, migliorando l'esperienza utente attraverso l'adattamento dinamico delle risposte. L'obiettivo finale è creare un assistente virtuale affidabile, in grado di comprendere e rispondere in modo accurato, favorendo un'interazione naturale ed efficace.

1.2 Scelta nome *Brigid*

Il nome Brigid è stato scelto per la chatbot in riferimento alla figura mitologica e simbolica di Brigid, una divinità celtica associata alla saggezza, alla guarigione e alla poesia. Questa scelta riflette l'intento della chatbot di offrire non solo assistenza e supporto agli utenti, ma anche un dialogo empatico e costruttivo. Brigid è storicamente venerata come un simbolo di protezione e conoscenza, caratteristiche che si allineano con il ruolo della chatbot nel fornire risposte intelligenti e un'interazione rassicurante. Il nome vuole evocare un senso di fiducia e affidabilità, sottolineando la capacità del sistema di comprendere e rispondere alle esigenze dell'utente in modo sensibile ed efficace.

1.3 Metodologia CRISP-DM e Applicazione nel Progetto

Il CRISP-DM (Cross Industry Standard Process for Data Mining) è un modello di sviluppo standardizzato utilizzato nell'ambito dell'analisi dei dati e dell'intelligenza artificiale. Si compone di sei fasi iterative: comprensione del business, comprensione dei dati, preparazione dei dati, modellazione, valutazione e deployment. Nel contesto del nostro progetto, questa metodologia è stata applicata per garantire uno sviluppo strutturato ed efficace della chatbot Brigid.

- **Comprensione del business**: Abbiamo definito gli obiettivi principali della chatbot, ossia fornire assistenza e supporto agli utenti attraverso un'interazione intelligente e contestualizzata.
- **Comprensione dei dati**: È stata analizzata la tipologia di input attesi dagli utenti e strutturato un dataset di intenti (`intents.json`), utile per il riconoscimento delle richieste.
- **Preparazione dei dati**: Sono stati implementati processi di preprocessing NLP per migliorare la qualità dell'input, tra cui tokenizzazione con BERT e pulizia del testo.
- **Modellazione**: È stato sviluppato un modello di classificazione basato su reti neurali con Keras, addestrato sugli intenti, e affiancato da un modello generativo (Gemini) per la gestione di input non previsti.

- **Valutazione**: Il sistema è stato testato su frasi reali e sintetiche, con un'analisi della confidence e metriche di accuratezza per garantire prestazioni affidabili.
- **Deployment**: La chatbot è stato implementato in una modalità di interazione basata esclusivamente su terminale, permettendo agli utenti di avviare una conversazione direttamente tramite riga di comando. Questa scelta semplifica l'accessibilità e facilita i test, consentendo miglioramenti futuri per un'eventuale integrazione in interfacce più avanzate.

L'approccio CRISP-DM ha permesso di sviluppare Brigid in modo metodico, migliorando progressivamente il modello sulla base delle esigenze reali degli utenti e mantenendo una struttura flessibile per future estensioni.

2 Comprensione del Business

La fase di comprensione del business all'interno della metodologia CRISP-DM è fondamentale per definire gli obiettivi del progetto e garantire che il sistema sviluppato soddisfi le esigenze degli utenti finali. In questa sezione vengono analizzati il contesto, gli obiettivi e i requisiti della chatbot Brigid, con un focus sull'impatto che avrà nell'ambito di utilizzo previsto.

2.1 Contesto e Motivazioni

L'idea alla base dello sviluppo di Brigid nasce dalla necessità di creare un assistente virtuale capace di supportare gli utenti attraverso un'interazione naturale e intelligente. La chatbot è stato progettato per offrire risposte pertinenti, basandosi su un modello di classificazione degli intenti e sull'uso di tecnologie avanzate di elaborazione del linguaggio naturale (NLP).

2.2 Obiettivi della Chatbot

Gli obiettivi principali della chatbot Brigid sono:

- Fornire assistenza e supporto agli utenti tramite una conversazione fluida e intuitiva.
- Comprendere il linguaggio naturale utilizzando modelli avanzati come BERT per l'analisi semantica dei messaggi.
- Predire gli intenti dell'utente con un modello di classificazione basato su una rete neurale, addestrato su un dataset di intenti (`intents.json`).
- Gestire input imprevisti grazie all'integrazione con un modello generativo (Gemini), migliorando la capacità di risposta in situazioni non coperte dal dataset.
- Mantenere una struttura modulare e flessibile, con possibilità di espansione futura in interfacce più avanzate.

2.3 Requisiti del Progetto

Per garantire il raggiungimento degli obiettivi sopra elencati, sono stati definiti i seguenti requisiti tecnici e funzionali:

2.3.1 Requisiti Funzionali

Codice	Descrizione
RF_1	La chatbot deve essere in grado di riconoscere e classificare correttamente gli intenti dell'utente.
RF_2	Deve fornire risposte pertinenti in base al contesto della conversazione.
RF_3	Deve gestire input non previsti e rispondere in modo adeguato utilizzando un fallback intelligente (Gemini).
RF_4	L'interazione deve avvenire tramite un'interfaccia a riga di comando (CLI).

2.3.2 Requisiti Tecnici

Codice	Descrizione
RT_1	Utilizzo di BERT per la generazione di embedding delle frasi dell'utente.
RT_2	Implementazione di una rete neurale feedforward con Keras per la classificazione degli intenti.
RT_3	Integrazione con Google Gemini per risposte generative in caso di input fuori dal dominio del dataset.
RT_4	Struttura del codice organizzata in moduli (`pretrain.py`, `chatbot_terminal.py`).

2.4 Benefici Attesi

L'implementazione di Brigid porta diversi vantaggi:

- **Maggiore accessibilità**: grazie alla possibilità di avviare la chatbot direttamente dal terminale senza necessità di un'interfaccia grafica.
- **Migliore comprensione delle richieste dell'utente**: grazie all'integrazione con modelli NLP avanzati.
- **Espandibilità e flessibilità**: con la possibilità di migliorare il dataset, affinare il modello e integrare nuove funzionalità in futuro.

Questa fase di comprensione del business ha guidato tutte le decisioni successive, permettendo di sviluppare la chatbot in modo mirato e orientato agli obiettivi definiti.

3 Comprensione dei dati

La fase di comprensione dei dati all'interno della metodologia CRISP-DM è essenziale per analizzare e strutturare le informazioni su cui il modello si basa. In questa sezione viene descritto il dataset utilizzato per addestrare la chatbot Brigid, il formato dei dati, le categorie di intenti e le loro caratteristiche.

3.1 Struttura del Dataset

Il dataset utilizzato per l'addestramento della chatbot è contenuto nel file `intents.json` e segue un formato standard per chatbot basati su intenti. Ogni intento è definito da:

- Un **tag univoco** che identifica la categoria dell'intento.
- Un insieme di **pattern** (frasi di esempio) che rappresentano possibili input dell'utente per quell'intento.
- Una lista di **risposte predefinite** che la chatbot può fornire se viene rilevato un determinato intento.

3.1.1 Esempio di Struttura di un Intento

```
{
    "intents": [
    {
        "tag": "greeting",
        "patterns": ["Hello", "Hi there!", "Hey", "Good
            morning"],
        "responses": ["Hello! How can I help you?", "Hi!
            What can I do for you?"]
    },
    {
        "tag": "goodbye",
        "patterns": ["Bye", "See you later", "Goodbye"],
        "responses": ["Goodbye! Have a great day.", "See
            you soon!"]
    }
    ]
}
```

3.2 Tipologie di Intenti

Il dataset contiene una serie di intenti suddivisi in diverse categorie, tra cui:

- **Saluti e chiusura** (es. `greeting`, `goodbye`)
- **Richieste informative** (es. `time`, `weather`, `help`)
- **Supporto psicologico** (es. `sadness`, `anxiety`, `stress_relief`)

- **Conversazioni generiche** (es. `smalltalk`, `chitchat`)

L'obiettivo è garantire che la chatbot sia in grado di riconoscere correttamente le intenzioni dell'utente e rispondere in modo pertinente.

3.3 Analisi della Distribuzione dei Dati

Per assicurare un buon addestramento del modello, è stata effettuata un'analisi della distribuzione degli intenti all'interno del dataset:

- **Bilanciamento**: Un dataset equilibrato evita che il modello sia troppo incline a predire alcuni intenti rispetto ad altri.
- **Varietà dei pattern**: Ogni intento deve avere un numero sufficiente di frasi di esempio per garantire una generalizzazione efficace.
- **Gestione di sinonimi e variazioni linguistiche**: Frasi diverse con lo stesso significato devono essere incluse per migliorare la capacità del modello di riconoscere input variati.

Se il dataset risultasse sbilanciato, potrebbero essere necessarie tecniche di *data augmentation*, come la generazione di frasi alternative tramite modelli generativi (es. `GPT`) o l'uso di sinonimi.

3.4 Tipologia di Dati Utilizzati

I dati presenti nel dataset sono **dati testuali non strutturati**, che vengono elaborati e trasformati in vettori numerici tramite `BERT`. Le fasi di elaborazione comprendono:

- **Tokenizzazione e conversione in embedding**: Il testo viene trasformato in rappresentazioni numeriche (vettori).
- **Assegnazione delle etichette**: Ogni frase viene associata al suo intento corrispondente.
- **Training del modello**: Il modello viene addestrato sui dati preprocessati per imparare a classificare gli intenti.

3.5 Problematiche del Dataset Iniziale

Il dataset iniziale, recuperato da Kaggle, conteneva 75 intenti, un numero limitato per coprire una vasta gamma di interazioni utente. Questo ha presentato diverse problematiche che avrebbero potuto compromettere le prestazioni della chatbot Brigid:

- **Copertura insufficiente degli intenti**: Il numero ridotto di intenti limitava la capacità della chatbot di rispondere efficacemente a un'ampia varietà di domande.
- **Scarso numero di pattern per intento**: Ogni intento aveva pochi esempi di frasi (*pattern*), il che poteva portare il modello a non riconoscere correttamente input leggermente diversi da quelli presenti nel dataset.

- **Rischio di overfitting**: Un dataset con pochi esempi per ogni classe può portare il modello ad adattarsi troppo ai dati di training, risultando poco flessibile su input reali.
- **Mancanza di diversità nelle risposte**: Il numero limitato di risposte per ogni intento poteva rendere le interazioni con la chatbot ripetitive e meno naturali.

Per risolvere queste problematiche, è stato necessario espandere e migliorare il dataset attraverso l'applicazione di tecniche di *data augmentation*.

3.6 Applicazione della Data Augmentation e Benefici

Per migliorare la qualità del dataset e garantire un addestramento più efficace della chatbot, è stato applicato un processo di **data augmentation**, ossia la generazione artificiale di nuovi dati per arricchire il dataset esistente.

3.6.1 Espansione del Dataset

- Il numero di intenti è stato ampliato da **75 a 223**, aumentando la varietà delle interazioni possibili.
- Ogni intento è stato arricchito con una media di **50 pattern di input**, aumentando la capacità del modello di riconoscere input con diverse formulazioni.
- Ogni intento è stato dotato di **10 risposte fisse**, garantendo maggiore diversità nelle risposte generate.

3.6.2 Ottimizzazione dei Pattern

Per migliorare la classificazione degli intenti, i **pattern di input** sono stati **ridotti in lunghezza**, evitando eccessiva variabilità e garantendo maggiore uniformità. Questa ottimizzazione ha reso il modello più preciso nel riconoscere le richieste degli utenti, riducendo **ambiguità** tra intenti simili. Inoltre, la rimozione di formulazioni ridondanti ha reso l'**addestramento più efficiente**, limitando il rischio di **sovrapposizione tra classi** e migliorando la capacità di generalizzazione della chatbot.

3.6.3 Generazione di Nuovi Pattern con ChatGPT

Per **arricchire il dataset** e migliorare la capacità della chatbot di gestire input variabili, è stato impiegato **ChatGPT** per la generazione di nuovi **pattern naturali e realistici**. Questo approccio ha ampliato la **diversità delle espressioni linguistiche**, rendendo le interazioni più fluide e l'esperienza utente più naturale. L'automazione ha inoltre consentito un'espansione **rapida ed efficace** del dataset, migliorando la copertura delle possibili varianti linguistiche utilizzate dagli utenti.

3.6.4 Benefici della Data Augmentation

L'applicazione della *data augmentation* ha portato diversi vantaggi:

- **Migliore generalizzazione** del modello, riducendo il rischio di overfitting.

- **Aumento della robustezza** nella comprensione di input variati.
- **Miglior bilanciamento del dataset**, evitando intenti con pochi esempi.
- **Maggiore adattabilità** a input reali, grazie alla varietà dei pattern generati.

Grazie a queste ottimizzazioni, la chatbot Brigid è ora in grado di comprendere e rispondere con maggiore accuratezza e naturalezza agli input degli utenti.

4 Preparazione dei dati

5 Modellazione

6 Valutazione

7 Deployment

8 Descrizione dell'agente

8.1 Obiettivi

L'agente chatbot è un sistema basato sull'intelligenza artificiale, progettato per interagire con gli utenti attraverso conversazioni simulate in linguaggio naturale ed è in grado di rispondere a domande relative al supporto psicologico. Questo agente è programmato per comprendere input testuali, elaborare le richieste e fornire risposte pertinenti, coerenti e contestualizzate. La chatbot opera attraverso due modelli distinti: il primo, personalizzato, si focalizza sul fornire supporto emotivo; il secondo, *Gemini*, genera risposte fuori dal contesto precedente, ampliando così la gamma delle interazioni possibili.

8.2 Specifica PEAS

8.3 Analisi del problema

La creazione di una chatbot per il supporto psicologico presenta diverse complessità. Comprendere il linguaggio umano, spesso ambiguo o emotivamente carico, richiede modelli avanzati di elaborazione del linguaggio naturale. Inoltre, generare risposte empatiche e contestuali è una sfida, poiché la chatbot deve bilanciare personalizzazione e coerenza, evitando risposte inappropriate o influenzate da bias. Abbiamo affrontato queste sfide adottando modelli NLP, come BERT, personalizzati per riconoscere emozioni e intenzioni. Per garantire risposte empatiche e sicure, abbiamo combinato risposte generative con risposte predefinite per situazioni delicate. Queste soluzioni consentono alla chatbot di offrire un supporto psicologico efficace e adattato alle esigenze individuali.

9 Raccolta, analisi e preprocessing dei dati

9.1 Scelta del dataset

Abbiamo iniziato utilizzando un dataset in formato JSON trovato online sul sito Kaggle, che poi abbiamo ampliato e perfezionato attraverso un lavoro mirato, avvalendoci anche del supporto di ChatGPT per integrare ulteriori dati. Questo processo ci ha permesso di sviluppare un dataset perfettamente adattato alle esigenze della chatbot. Grazie a questa personalizzazione, siamo riusciti a garantire un'esperienza più efficace e rilevante per gli utenti che interagiscono con il sistema.
I dati presenti nel dataset di Kaggle seguono una struttura composta da tag, pattern e responses. Questo ci ha permesso di mantenere una coerenza strutturale, applicando la stessa organizzazione anche ai dati aggiunti successivamente durante il nostro lavoro.
Il dataset in questione è reperibile a questo link.

9.2 Tecnologie utilizzate

Abbiamo utilizzato come ambiente di sviluppo PyCharm, che ci ha permesso di utilizzare come linguaggio di programmazione Python.

9.2.1 Librerie utilizzate

Le librerie che abbiamo utilizzato sono:

- **Numpy**: Per manipolazione di array numerici, utilizzati da BERT.

```
    # Conversione degli embedding e delle etichette in
        array NumPy:
    train_x = np.array(train_x)
    train_y = np.array(train_y)

    #Calcolo degli argmax per le previsioni e le
       etichette vere:
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true_classes = np.argmax(train_y, axis=1)

    #Calcolo delle confidence scores:
    confidences = np.max(y_pred, axis=1)

    #Conversione degli embedding BERT in array NumPy:
    return outputs.last_hidden_state.mean(dim=1).
       squeeze().numpy()
```

Codice 1 Utilizzo di Numpy nel codice

- **JSON**: Per leggere il file intents.json (contiene i pattern e i tag associati).

```
    "tag": "sad",
    "patterns": [
    "I feel sad",
```

```
    "I am not feeling good",
    "I feel down today",
    "I am feeling low",
    "I am upset",
    "I feel so lonely"
    ],
    "responses": [
    "I am sorry you are feeling this way. It is okay to be sad sometimes.",
    "Take your time to process these feelings. You are not alone.",
    "It is important to be kind to yourself during tough moments.",
    "I am here to listen if you need someone to talk to."
    ]
```

Codice 2 esempio breve di file JSON

- **Pickle**: Per salvare oggetti Python (come le classi) su disco.

```
# Salva le classi e i dati
    pickle.dump(classes, open('../models/classes.pkl', 'wb'))
```

Codice 3 Utilizzo di Pickle nel codice

- **Transformers** (da Hugging Face): Per caricare BERT e il suo tokenizer.

```
#Caricamento del Tokenizer e del Modello BERT
    from transformers import BertTokenizer, BertModel

    # Carica il tokenizer e il modello pre-addestrato di BERT
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    bert_model = BertModel.from_pretrained('bert-base-uncased')

#Funzione per ottenere l'Embedding con BERT
    def get_bert_embedding(sentence):
    inputs = tokenizer(sentence, return_tensors="pt", padding=True, truncation=True, max_length=50)
    with torch.no_grad():
    outputs = bert_model(**inputs)
    # Usa la media degli hidden states per rappresentare la frase
    return outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
```

```
15
16 #Creazione dei Dati di Addestramento
17     train_x = []
18     train_y = []
19
20     for pattern, tag in documents:
21     # Ottieni l'embedding per la frase
22     embedding = get_bert_embedding(pattern)
23     train_x.append(embedding)
24
25     # One-hot encoding del tag
26     output_row = [0] * len(classes)
27     output_row[classes.index(tag)] = 1
28     train_y.append(output_row)
29
30     train_x = np.array(train_x)
31     train_y = np.array(train_y)
```

Codice 4 Utilizzo di Transformers nel codice

- **PyTorch**: Utilizzato per il modello BERT.

```
1 #Tokenizzazione e Inferenza con BERT:
2     inputs = tokenizer(sentence, return_tensors="pt",
         padding=True, truncation=True, max_length=50)
3     with torch.no_grad():
4     outputs = bert_model(**inputs)
5
6 #Estrazione degli embedding di BERT:
7     return outputs.last_hidden_state.mean(dim=1).
         squeeze().numpy()
```

Codice 5 Utilizzo di PyTorch nel codice

- **codice/TensorFlow/Keras**: Per costruire e addestrare il modello di classificazione.

```
1     #Costruzione del modello di rete neurale
2     from tensorflow.keras.models import Sequential
3     from tensorflow.keras.layers import Dense, Dropout
4     from tensorflow.keras.optimizers import Adam
5     from tensorflow.keras.metrics import Precision,
         Recall
6
7     # Costruzione del modello di classificazione
8     model = Sequential()
9     model.add(Dense(128, input_shape=(train_x.shape
         [1],), activation='relu'))
10    model.add(Dropout(0.5))
11    model.add(Dense(64, activation='relu'))
```

```
12      model.add(Dropout(0.5))
13      model.add(Dense(len(train_y[0]), activation='
            softmax'))
14
15      # Compilazione del modello
16      model.compile(
17      optimizer=Adam(learning_rate=0.001),
18      loss='categorical_crossentropy',
19      metrics=['accuracy', Precision(), Recall()]
20      )
21
22      #Addestramento del modello
23      hist = model.fit(train_x, train_y, epochs=100,
            batch_size=8, verbose=1)
24
25      #Salvataggio del modello
26      # Salva il modello addestrato
27      model.save('../models/chatbot_model.keras')
28
29      #Previsioni con il modello addestrato
30      y_pred = model.predict(train_x)
31      y_pred_classes = np.argmax(y_pred, axis=1)
32      y_true_classes = np.argmax(train_y, axis=1)
```

Codice 6 Utilizzo di TensorFlow e Keras nel codice

.

10 Parti della chatbot

10.1 Pretrain

[AGGIUNGERE Immagine Codice]

Il codice rappresenta un processo per creare una chatbot basato su machine learning. Si avvale di BERT per generare rappresentazioni numeriche (embedding) delle frasi e poi costruisce un modello di classificazione per determinare i tag delle frasi in input. Il modello creato presenta tre strati:

- **Input**: La dimensione degli embedding di BERT.
- **Strati nascosti**: Due strati densi con ReLU e Dropout per evitare overfitting.
- **Output**: Softmax per classificazione multi-classe.

.

[AGGIUNGERE Immagine Codice]

10.2 Analyze intents

Il codice carica un file JSON che contiene gli intenti di un chatbot, li analizza e restituisce informazioni utili. Inizia leggendo il file e verificando che contenga la chiave "intents". Dopodiché, conta il numero totale di tag presenti, li elenca e analizza ciascun tag. Per ogni intento, stampa il nome del tag, il numero di frasi esempio associate (patterns) e il numero di risposte definite (responses). Se il file non viene trovato o non è un JSON valido, restituisce un messaggio d'errore.

10.3 Chatbot interface

Questo codice descrive un chatbot avanzato che combina l'uso di un modello di classificazione pre-addestrato con l'intelligenza artificiale generativa di Gemini. Il funzionamento si basa su diverse fasi. Per prima cosa, il sistema configura Gemini, un modello AI generativo che può creare risposte dinamiche e contestuali. Viene poi caricato il file intents.json, che contiene una struttura di intenti predefiniti, con frasi esempio e risposte associate, insieme al modello di classificazione e alla lista dei tag.
Quando un utente invia un messaggio, la frase viene elaborata utilizzando il modello BERT, che trasforma il testo in un vettore numerico (embedding) per analizzarne il significato. Questo embedding viene passato al modello di classificazione, che prevede a quale intento (tag) il messaggio corrisponde. Se il modello è sicuro della sua previsione, restituisce una risposta predefinita collegata al tag identificato. Se invece la confidenza è bassa, la chatbot utilizza Gemini per generare una risposta più flessibile e creativa, basandosi anche sul contesto della conversazione grazie alla cronologia memorizzata.
Tutta questa logica è integrata in una pipeline che decide dinamicamente se utilizzare risposte predefinite o dinamiche. Nel frattempo, la cronologia della chat viene costantemente aggiornata per garantire che ogni messaggio tenga conto delle interazioni precedenti. Infine, la chatbot funziona come un sistema interattivo: attende l'input dell'utente, genera una risposta adeguata e la presenta, aggiornando il contesto per i messaggi futuri. Questo approccio ibrido rende la chatbot capace di rispondere in modo sia preciso sia naturale, adattandosi alle diverse situazioni conversazionali.

11 Glossario

Termini	Descrizione
Diversify	Progetto di Ingegneria del Software che prevede la realizzazione di un sito web per combattere le discriminazioni.
Gemini	Modello di linguaggio multimodale sviluppato da Google DeepMind.
PEAS	Performance Measure, Environment, Actuators, Sensors.
Bias	Errore sistematico causato da dati di addestramento sbilanciati o parziali, che porta il modello a produrre risultati distorti o discriminatori. Può influenzare l'equità, l'accuratezza e l'affidabilità delle previsioni.
NLP	Natural Language Processing.
Kaggle	Piattaforma online per data science e machine learning che offre competizioni, dataset pubblici, e strumenti collaborativi per sviluppare modelli e analizzare dati.
ChatGPT	Modello di linguaggio basato sull'architettura GPT (Generative Pre-trained Transformer), progettato per comprendere e generare testo in linguaggio naturale.
Tag	Etichetta che identifica un argomento specifico.
Pattern	Modelli o frasi che il bot cerca di riconoscere, spesso usando espressioni regolari o parole chiave.
Responses	Risposte predefinite che il bot restituisce quando un pattern corrisponde a un tag.
Intent	Gruppo formato da un tag, patterns e responses.
Dataset	Raccolta di dati organizzati in una struttura definita.
Numpy	Libreria Python fondamentale per il calcolo scientifico.
JSON	Formato di scambio dati leggero e leggibile. Allo stesso momento, è una libreria Python che permette la manipolazione di questi ultimi.
Pickle	Libreria Python che permette di serializzare (salvare) e deserializzare (caricare) oggetti Python in un formato binario.
Transformers	Libreria sviluppata da Hugging Face che semplifica l'uso di modelli avanzati di deep learning.
PyTorch	Libreria di machine learning, sviluppata da Facebook, usata per l'addrestramento di reti neurali.
TensorFlow/Keras	Libreria di machine learning open-source sviluppata da Google, utilizzata per costruire e addestrare modelli di deep learning.
Pretrain	Processo di addestramento iniziale di un modello di machine learning su un ampio dataset generico.

Termini	Descrizione
BERT	Bidirectional Encoder Representations from Transformers. Modello di deep learning per il trattamento del linguaggio naturale sviluppato da Google.
Embedding	Tecnica di rappresentazione numerica di parole, frasi o altre unità di dati in uno spazio continuo.