Documentazione

Progetto Brigid AI Chatbot

Catello Martone, Davide Viola, Gabriella Fede, Pasquale Anatriello

28 gennaio 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

L'obiettivo della chatbot **Brigid** è quello di fornire un supporto psicologico, incluso all'interno del progetto *Diversify*. La chatbot offre un sostegno concreto e monitora l'andamento delle conversazioni, raccogliendo dati attraverso l'interazione con l'utente.
Questi dati saranno utilizzati per ottimizzare il funzionamento della chatbot, adattando le conversazioni alle esigenze e alle preferenze specifiche di ciascun utente. In questo modo, l'utente potrà vivere un'esperienza interattiva più personalizzata e coinvolgente, con risposte e contenuti che riflettono in maniera più precisa le sue aspettative e necessità.

2 Descrizione dell'agente

2.1 Obiettivi

Un agente chatbot è un sistema basato sull'intelligenza artificiale, progettato per interagire con gli utenti attraverso conversazioni simulate in linguaggio naturale. 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 contesto, ampliando così la gamma delle interazioni possibili.

2.2 Specifica PEAS

2.3 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. Dal punto di vista tecnico, il sistema deve essere scalabile per gestire molteplici conversazioni senza ritardi.
Abbiamo affrontato queste sfide adottando modelli NLP avanzati, come GPT, personalizzati per riconoscere emozioni e intenzioni. Per garantire risposte empatiche e sicure, abbiamo combinato risposte generative con risposte predefinite per situazioni delicate. L'architettura serverless assicura scalabilità e rapidità.
Queste soluzioni consentono alla chatbot di offrire un supporto psicologico efficace e adattato alle esigenze individuali.

3 Raccolta, analisi e preprocessing dei dati

3.1 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 altamente specifico e 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 response. 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.

3.2 Tecnologie utilizzate

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

3.2.1 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 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()

#Creazione dei Dati di Addestramento
    train_x = []
    train_y = []

```

```
    for pattern, tag in documents:
    # Ottieni l'embedding per la frase
    embedding = get_bert_embedding(pattern)
    train_x.append(embedding)

    # One-hot encoding del tag
    output_row = [0] * len(classes)
    output_row[classes.index(tag)] = 1
    train_y.append(output_row)

    train_x = np.array(train_x)
    train_y = np.array(train_y)
```

Codice 4 Utilizzo di Transformers nel codice

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

```
#Tokenizzazione e Inferenza con BERT:
    inputs = tokenizer(sentence, return_tensors="pt",
       padding=True, truncation=True, max_length=50)
    with torch.no_grad():
    outputs = bert_model(**inputs)

#Estrazione degli embedding di BERT:
    return outputs.last_hidden_state.mean(dim=1).
       squeeze().numpy()
```

Codice 5 Utilizzo di PyTorch nel codice

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

```
#Costruzione del modello di rete neurale
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Dropout
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.metrics import Precision,
       Recall

# Costruzione del modello di classificazione
    model = Sequential()
    model.add(Dense(128, input_shape=(train_x.shape
       [1],), activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(len(train_y[0]), activation='
       softmax'))

# Compilazione del modello
```

```
    model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy', Precision(), Recall()]
    )

#Addestramento del modello
    hist = model.fit(train_x, train_y, epochs=100,
       batch_size=8, verbose=1)

#Salvataggio del modello
    # Salva il modello addestrato
    model.save('../models/chatbot_model.keras')

#Previsioni con il modello addestrato
    y_pred = model.predict(train_x)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true_classes = np.argmax(train_y, axis=1)
```

Codice 6 Utilizzo di TensorFlow e Keras nel codice

.

4 Parti della chatbot

4.1 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]

4.2 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.

4.3 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, il 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, il 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 il chatbot capace di rispondere in modo sia preciso sia naturale, adattandosi alle diverse situazioni conversazionali.

5 Glossario