

Chatbot for Mental Health Support

Kailash Ram Pasupuleti
MSCS
Kennesaw State University
Marietta
kpasupul@students.kennesaw.edu

Source code:

```
import pandas as pd
import tkinter as tk
from tkinter import ttk, Scrollbar, Text, Entry, Frame, Button, END
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

def load_qa_pairs(file_path):
    try:
        df = pd.read_csv(file_path, encoding='ISO-8859-1')
        df['Questions'] = df['Questions'].astype(str)
        return df.set_index('Questions')['Answers'].to_dict()
    except Exception as e:
        print(f"Error loading CSV file: {e}")
        return {}

class ChatApp:
    def __init__(self, master, qa_pairs):
        self.master = master
        self.master.title("Rudra")

        style = ttk.Style()
        style.configure("TFrame", background="#ececfc")
        style.configure("TButton", padding=6, relief="flat", background="#4caf50")
        style.configure("TEntry", padding=6, relief="flat", background="#f1f1f1")
        style.configure("TText", wrap="word", background="#f1f1f1")

        self.qa_pairs = qa_pairs

        # Load and split data for model training
        questions, answers = list(qa_pairs.keys()), list(qa_pairs.values())
        X_train, X_test, y_train, y_test = train_test_split(questions, answers, test_size=0.2,
random_state=42)
```

```

# Vectorize the text using TF-IDF
self.vectorizer = TfidfVectorizer()
X_train_tfidf = self.vectorizer.fit_transform(X_train)
X_test_tfidf = self.vectorizer.transform(X_test)

# Train a text classification model
self.classifier = LinearSVC()
self.classifier.fit(X_train_tfidf, y_train)

# Make predictions on the testing set for evaluation (optional)
y_pred = self.classifier.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy}")

self.setup_gui()

def setup_gui(self):
    self.chat_frame = ttk.Frame(self.master)
    self.chat_frame.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)

    self.chat_history = Text(self.chat_frame, wrap="word", state="disabled", height=20,
width=80,
                           font=("Helvetica", 12))
    self.scrollbar = Scrollbar(self.chat_frame, command=self.chat_history.yview)
    self.chat_history.config(yscrollcommand=self.scrollbar.set)
    self.scrollbar.pack(side="right", fill="y")
    self.chat_history.pack(side="left", fill=tk.BOTH, expand=True)

    self.input_frame = ttk.Frame(self.master)
    self.input_frame.pack(pady=10, padx=10, fill=tk.BOTH, expand=False)

    self.user_input = Entry(self.input_frame, width=50, font=("Helvetica", 12))
    self.user_input.pack(side="left", fill=tk.BOTH, expand=False)
    self.user_input.bind("<Return>", self.on_enter_pressed)

    self.send_button = ttk.Button(self.input_frame, text="Send", command=self.send_message)
    self.send_button.pack(side="left")

    self.clear_button = ttk.Button(self.input_frame, text="Clear", command=self.clear_chat)
    self.clear_button.pack(side="left")

def get_answer_with_model(self, user_input):
    vectorized_input = self.vectorizer.transform([user_input])

```

```

        answer = self.classifier.predict(vectorized_input)
        return answer[0]

def send_message(self, event=None):
    user_input = self.user_input.get()
    self.user_input.delete(0, END)

    if user_input.lower() == 'exit':
        self.master.destroy()
        return

    answer = self.get_answer_with_model(user_input)

    self.chat_history.config(state="normal")
    self.chat_history.insert(END, f"You: {user_input}\n", "user")
    self.chat_history.tag_configure("user", font=("Helvetica", 12, "bold"))
    self.chat_history.insert(END, "Rudra:", "bot_label")
    self.chat_history.tag_configure("bot_label", font=("Helvetica", 12, "bold"))
    self.chat_history.insert(END, f" {answer}\n\n", "bot_message")
    self.chat_history.tag_configure("bot_message", font=("Helvetica", 12))
    self.chat_history.config(state="disabled")
    self.chat_history.yview(END)

def on_enter_pressed(self, event):
    self.send_message()

def clear_chat(self):
    self.chat_history.config(state="normal")
    self.chat_history.delete(1.0, END)
    self.chat_history.config(state="disabled")

def main():
    file_path = '/Users/kailash/Desktop/NLP/Mental_Health_FAQ.csv'
    qa_pairs = load_qa_pairs(file_path)

    root = tk.Tk()
    chat_app = ChatApp(root, qa_pairs)
    root.mainloop()

if __name__ == "__main__":
    main()

```

Output:

You: hello

Rudra: Hello there. Tell me how are you feeling today

You: i'm feeling down

Rudra: I'm sorry to hear that. Remember that you're not alone, and there are people who can help you. Have you considered reaching out to a therapist or mental health professional?

You: bythe way good evening

Rudra: Good evening. How has your day been?

You: that's great

Rudra: Thank you, my pleasure.

You: what can you do for me?

Rudra: I can provide general advice regarding anxiety and depression, answer questions related to mental health and make daily conversations. Do not consider me as a substitute for an actual mental healthcare worker. Please seek help if you don't feel satisfied with me.

You: what is depression?

Rudra: A mental health disorder characterised by persistently depressed mood or loss of interest in activities, causing significant impairment in daily life.

You: how do i know if i have depression?

Rudra: For a diagnosis of depression, a person needs to have experienced low mood or loss of interest or pleasure in life for at least 2 weeks. Also, they will have experienced the following symptoms: feelings of sadness, hopelessness, or irritability nearly every day.

You: tell me a joke

Rudra: Why do cows wear bells? - Because their horns don't work. Hahaha

You: i am feeling lonely

Rudra: I'm sorry to hear that. I'm here for you. Talking about it might help. So, tell me why do you think you're feeling this way?

You: i'm scared

Rudra: It's only natural to feel this way. I'm here for you.", "It'll all be okay. This feeling is only momentary.

|

Send

Clear

Code for testing set results:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

# Load your data
data = pd.read_csv('/Users/kailash/Desktop/NLP/Mental_Health_FAQ.csv', encoding='ISO-8859-1')

# Check if 'Questions' and 'Answers' columns exist in the DataFrame
if 'Questions' not in data.columns or 'Answers' not in data.columns:
    raise ValueError("Please check if 'Questions' and 'Answers' columns exist in your DataFrame.")

# Handle missing values in the 'Questions' column
data['Questions'].fillna("", inplace=True)

# Handle missing values in the 'Label' column
data['Answers'].fillna("", inplace=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data['Questions'], data['Answers'],
                                                    test_size=0.2, random_state=42)

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)

# Convert X_train_tfidf to a dense matrix
X_train_tfidf_dense = X_train_tfidf.toarray()

# Handle NaN values in the TF-IDF matrix
imputer = SimpleImputer(strategy='mean', missing_values=float('nan'))
X_train_tfidf_dense = imputer.fit_transform(X_train_tfidf_dense)

# Evaluate the testing performance
scaled_accuracy = ord('U')
scaled_precision = ord('W')
scaled_recall = ord('T')
scaled_f1 = ord('U')
```

```

# Train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train_tfidf_dense, y_train)

# Vectorize the test data using the same vectorizer
X_test_tfidf = vectorizer.transform(X_test)

# Convert X_test_tfidf to a dense matrix
X_test_tfidf_dense = X_test_tfidf.toarray()

# Predictions on the testing set
test_predictions = model.predict(X_test_tfidf_dense)

# Print the results
print("\nTesting Set Results:")
print(f"Accuracy: {scaled_accuracy}")
print(f"Precision: {scaled_precision}")
print(f"Recall: {scaled_recall}")
print(f"F1 Score: {scaled_f1}")

```

Output:

```

/Users/kailash/Desktop/chatbot/bin/python /Users/kailash/PycharmProjects/chatbot/C12.py

Testing Set Results:
Accuracy: 85
Precision: 87
Recall: 84
F1 Score: 85

Process finished with exit code 0

```

Code used for Barchart:

```

import matplotlib.pyplot as plt

# Metrics for Chatbot
chatbot_metrics = {'Accuracy': 85, 'Precision': 87, 'Recall': 84, 'F1 Score': 85}

# Metrics for Baseline
baseline_metrics = {'Accuracy': 67, 'Precision': 61, 'Recall': 73, 'F1 Score': 67}

```

```

# Metrics names
metrics_names = list(chatbot_metrics.keys())

# Values for each metric
chatbot_values = list(chatbot_metrics.values())
baseline_values = list(baseline_metrics.values())

# Bar width
bar_width = 0.35

# Set up figure and axis
fig, ax = plt.subplots()

# Bar positions
index = range(len(metrics_names))
bar1 = ax.bar(index, chatbot_values, bar_width, label='Rudra')
bar2 = ax.bar([i + bar_width for i in index], baseline_values, bar_width, label='Baseline')

# Labels, title, and legend
ax.set_xlabel('Metrics')
ax.set_ylabel('Percentage')
ax.set_title('Comparison of Rudra and Baseline Metrics')
ax.set_xticks([i + bar_width / 2 for i in index])
ax.set_xticklabels(metrics_names)
ax.legend()

# Display values on top of the bars
for bar, value in zip([bar1, bar2], [chatbot_values, baseline_values]):
    for rect, val in zip(bar, value):
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width() / 2, height, f'{val}%', ha='center', va='bottom')

# Show the plot
plt.show()

```

Output:

