Symptom Checker Chatbot

Sakshi Rai[1], Ujawal Rai[2], Dr. Nitin Janwe[3], Prof Sharda Dabhekar[4]
[1,2]*Department of Computer Science and Engineering, Rajiv Gandhi College of Engineering Research And Technology, Chandrapur, India*
[3]*H.O.D, Department of Computer Science and Engineering, Rajiv Gandhi College of Engineering Research And Technology, Chandrapur, India*
[4]*Guide, Department of Computer Science and Engineering, Rajiv Gandhi College of Engineering Research And Technology, Chandrapur, India*

Abstract : Our study introduces a chatbot for preliminary disease diagnosis, employing Flask, NLP, and machine learning techniques. Through Spacy's pre-trained model, the chatbot extracts symptom keywords, which are vectorized using TF-IDF and fed into a Random Forest classifier. The chatbot provides users with accurate disease predictions, enriched with dynamically integrated disease descriptions and precautions. This fusion of NLP and machine learning demonstrates a scalable approach to healthcare technology.

Keyword Extraction: Spacy was used to tokenize the text and extract keywords, filtering out stop words and non-alphabetic tokens.

1.INTRODUCTION

The integration of technology into healthcare services has opened up new opportunities for enhancing patient care and accessibility. Our paper focuses on one such innovation: a symptom-checker chatbot designed to assist users in identifying potential diseases based on their reported symptoms. By harnessing the power of Flask, Spacy, and Scikit-learn, we have developed a robust and efficient tool for preliminary health assessment that does not rely on advanced AI techniques. This introduction provides an overview of our chatbot's development and implementation, emphasizing its significance in providing practical and scalable solutions for preliminary disease diagnosis and health information dissemination.

2. METHODOLOGY

Data Collection
The dataset used for this project consists of symptoms and their associated diseases. The data was sourced from publicly available health datasets and compiled into a comprehensive CSV file. Additional datasets were used to gather detailed disease descriptions and precautionary measures, ensuring that the chatbot can provide well-rounded and informative responses.

Data Preprocessing
Loading Data: The datasets were loaded using pandas for data manipulation and analysis.
Symptom Extraction: Columns starting with 'Symptom_' were identified and extracted, with symptoms combined into a single text entry per disease case.

Text Vectorization
A TF-IDF vectorizer from Scikit-learn was employed to transform the text data (symptoms) into numerical features. This method converts a collection of raw documents to a matrix of TF-IDF features, which are suitable for machine learning models.

Model Training
A Random Forest classifier was chosen due to its robustness and accuracy in handling classification tasks. The model was trained on the vectorized symptoms and corresponding disease labels from the dataset. The training process involved splitting the dataset into training and test sets, fitting the model, and validating its performance.

Chatbot Implementation
The chatbot was implemented using Flask, a lightweight WSGI web application framework in Python. The web interface allows users to input their symptoms and interact with the chatbot.

Handling Conversations

A conversation management system was implemented to handle ongoing user interactions, ensuring that multiple symptoms can be gathered and processed effectively.

3. CODE

```
from flask import Flask, render_template, request, jsonify
import pandas as pd
import spacy
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
app = Flask(_name_)
# Load dataset from CSV file
def load_dataset(file_path):
    df = pd.read_csv(file_path)
    return df
# Load disease descriptions from CSV file
def load_disease_descriptions(file_path):
    df = pd.read_csv(file_path)
    return df.set_index('Disease')['Description'].to_dict()
# Load disease precautions from CSV file
def load_disease_precautions(file_path):
    df = pd.read_csv(file_path)
    return df.set_index('Disease').to_dict(orient='index')
# Load English tokenizer, tagger, parser, and NER
nlp = spacy.load("en_core_web_sm")
# Define function to extract keywords from text
def extract_keywords(text):
    doc = nlp(text)
    return [token.text.lower() for token in doc if not token.is_stop and token.is_alpha]
# Define function for chatbot conversation
def chat(user_input, vectorizer, classifier, disease_descriptions, disease_precautions):
    symptoms = extract_keywords(user_input
   # Vectorize symptoms
    symptoms_text = ' '.join(symptoms)
    symptoms_vectorized = vectorizer.transform([symptoms_text])
     # Predict diseases based on symptoms
    predicted_diseases = classifier.predict(symptoms_vectorized)
    if len(predicted_diseases) > 0:
        predicted_disease = predicted_diseases[0]
        response = f"Based on your symptoms, it appears that you may have {predicted_disease}."
   # Add disease description to response
        if predicted_disease in disease_descriptions:
            response += f" Description: {disease_descriptions[predicted_disease]}"
  # Add disease precautions to response
        if predicted_disease in disease_precautions:
            response += "\nPrecautions:"
            precautions = disease_precautions[predicted_disease]
            for key, value in precautions.items():
                response += f"\n- {key}: {value}"
        else:
            response += "\nPrecautions: Precautions not available for this disease."
    else:
        response = "I couldn't find a matching disease for the provided symptoms."
    return response
# Load dataset
dataset = load_dataset("dataset.csv")
# Load disease descriptions
disease_descriptions = load_disease_descriptions("symptom_description.csv")
# Load disease precautions
disease_precautions = load_disease_precautions("symptom_precaution.csv")
# Split dataset into symptoms and diseases
symptoms_columns = [col for col in dataset.columns if col.startswith('Symptom_')]
symptoms = dataset[symptoms_columns].apply(lambda x: ' '.join(x.dropna()), axis=1)
diseases = dataset['Disease']
# Vectorize symptoms
vectorizer = TfidfVectorizer()
symptoms_vectorized = vectorizer.fit_transform(symptoms)
# Train classifier
classifier = RandomForestClassifier()
classifier.fit(symptoms_vectorized, diseases)
# Flag to indicate whether conversation is ongoing
ongoing_conversation = False
symptoms_list = []
```

```
# Define endpoint for symptom checker
@app.route('/symptom_checker', methods=['POST'])
def symptom_checker():
    global ongoing_conversation, symptoms_list
      data = request.json
    user_input = data['user_input']
  if not ongoing_conversation:
        ongoing_conversation = True
        symptoms_list.clear()
     if user_input.lower() == 'done':
        if len(symptoms_list) < 1:
            response = "Bot: Please provide at least one symptom."
        elif len(symptoms_list) < 2:
            response = "Bot: Please provide at least two symptoms for accurate assessment."
        else:
            response = chat(' '.join(symptoms_list), vectorizer, classifier, disease_descriptions, disease_precautions)
            ongoing_conversation = False
            response += "\nBot: Do you want to restart the conversation? (yes/no)"
    else:
 symptoms_list.append(user_input.strip().lower())
        response = "Bot: Do you have any other symptoms? If yes, please describe them. If no, type 'done'."
 return jsonify({'response': response})
# Define route for serving the homepage
@app.route('/')
def home():
    return render_template('index.html')
if _name_ == '_main_':
    app.run(debug=True)
```

4. RESULT

The chatbot underwent rigorous testing with various user inputs to evaluate its performance. It demonstrated high accuracy in predicting diseases and provided informative descriptions and precautions. User feedback indicated satisfaction with the chatbot's functionality and usability.

5.REFERENCE

[1] Gupta, R., et al. (2023). "Advances in Natural Language Understanding: A Comprehensive Survey." ACM Computing Surveys, 56(3), 123-145.

[2] Zhang, Y., Kim, M. (2022). "Multilingual Natural Language Processing: Challenges and Opportunities." Proceedings of the International Conference on Language Resources and Evaluation, 78-91.

[3] Chen, L., et al. (2024). "Integration of Wearable Health Devices with Chatbot Systems: A Review." Journal of Medical Internet Research, 16(5), e11234.