Technical Specification Document

Project Overview

This project is a Doctor Recommendation System designed to help patients find the most relevant doctors based on their natural language queries. It leverages machine learning, natural language processing (NLP), and similarity search techniques to match patient queries with doctor profiles stored in a database. The system is exposed as a REST API built using Flask, allowing seamless integration with other applications.

Key Functionalities

Natural Language Query Processing:
Accepts unstructured patient queries in plain English.
Converts the query into a structured format comparable to the doctor database.

Doctor Matching:
Uses embeddings to represent doctor profiles and patient queries as dense vectors.
Performs similarity searches to find the most relevant doctors.

Attribute-Based Scoring:
Enhances matching by scoring specific attributes such as specialty, location, and languages spoken.

REST API:
Provides an endpoint (/find_doctors) to accept patient queries and return the top 5 doctor recommendations.

Technologies Used

Programming Language
Python: The primary language for implementing the backend logic, machine learning, and API.
Frameworks and Libraries

Flask:
Used to create the REST API for handling patient queries and returning results.

Sentence Transformers:
Provides the all-mpnet-base-v2 model for generating embeddings of doctor profiles and patient queries.

FAISS:
A library for efficient similarity search and clustering of dense vectors. It is used to index and search doctor embeddings.

Transformers:
Utilized for the google/flan-t5-base model, which restructures patient queries into a structured format.

FuzzyWuzzy:
Used for fuzzy string matching to calculate attribute-based scores (e.g., matching specialties or languages).

NumPy:
Handles numerical operations, including normalization of embeddings.

TensorFlow:
Suppresses TensorFlow-related warnings for a cleaner runtime environment.

Data Handling

JSON:
Doctor data is stored in a JSON file (doctors_us.json), which contains detailed information about each doctor, such as name, specialty, location, languages spoken, insurance accepted, and more.

System Workflow

Data Loading:
Doctor data is loaded from a JSON file at startup.
Each doctor profile is validated and cleaned to ensure consistency.

Embedding Creation:
Doctor profiles are converted into descriptive text strings.
These descriptions are encoded into embeddings using the all-mpnet-base-v2 model.

FAISS Index Construction:
The embeddings are normalized for better similarity calculations.
A FAISS index is built using these embeddings, enabling efficient similarity searches.

Query Processing:
Patient queries are processed using the google/flan-t5-base model to generate a structured description.
The structured query is encoded into an embedding and normalized.

Similarity Search:
The patient query embedding is compared against the FAISS index to retrieve the top 5 closest matches.

Attribute-Based Scoring:
Additional scoring is applied based on specific attributes like specialty, location, and languages spoken.
Fuzzy string matching is used to calculate attribute scores.

Result Formatting:
The top 5 doctors are sorted by their combined similarity and attribute scores.
Results are returned as a JSON response.

API Details
Endpoint: /find_doctors
Method: POST

Request Payload:
Response:
Returns a list of the top 5 doctors with their details and scores.
Key Components

Doctor Data:
Stored in a JSON file (doctors_us.json).
Includes fields such as name, specialty, address, practice name, overall rating, languages spoken, insurances, reviews, and highlights.

Embedding Model:
Model: all-mpnet-base-v2 from Sentence Transformers.
Generates dense vector representations of doctor profiles and patient queries.

Similarity Search:
Library: FAISS (Facebook AI Similarity Search).
Performs nearest-neighbor searches to find the most relevant doctors.

Query Restructuring:
Model: google/flan-t5-base from Hugging Face Transformers.
Converts unstructured patient queries into structured descriptions.

Attribute Scoring:
Uses FuzzyWuzzy for flexible string matching.
Scores attributes like specialty, location, and languages spoken.

Challenges Addressed

Handling Unstructured Queries:
The use of a language model ensures that even vague or unstructured queries can be processed effectively.

Efficient Search:
FAISS enables fast similarity searches, even with large datasets.

Flexible Matching:
Fuzzy string matching allows for partial matches, ensuring flexibility in attribute-based scoring.

Future Enhancements

Scalability:
Move the doctor database to a cloud-based solution (e.g., AWS DynamoDB) for scalability.

Real-Time Updates:
Allow real-time updates to the doctor database and embeddings.

User Feedback:
Incorporate patient feedback to improve recommendations over time.

Advanced Query Understanding:
Use more advanced models for better query understanding and restructuring.
This document provides a comprehensive overview of the technical aspects of the Doctor Recommendation System. Let me know if you need further clarification or additional details!