DESCRIPTION OF A CHATBOT BASED MEDICAL EXPERT SYSTEM

Sameer Deshmukh, Jaideep Kekre ,Dept. Of Computer Engineering, Sinhgad College Of Engineering

Abstract: Doctors in India have to work long hours and see hundreds of patients a day. A lot of time is spent by doctors for gathering the personal and medical history of patients with respect to their demography and present and past symptoms. Our project enhances the first doctor-patient interaction where the doctor asks the patient certain questions for noting down their history

Index terms: Expert system, Medical decision modelling, Chatbot.

I .Introduction

The aim of this project to develop a chat bot based medical expert system**.** We believe that a patient's basic history taking can be done by having a conversation with a friendly chat bot, so that the doctor is freed from this activity and precious time is saved. This project will only aim to act as a middle man between the patient and doctor. It works by letting the patient chat with the system so that he/she can report their personal and medical history to the computer, which the computer keeps a track of. The system will then try to infer a diagnosis from this interaction and report it directly to the doctor by sending the doctor an email.

Expert system is a computer system that emulates the decision-making ability of a human expert. As described above, most doctors in India are over-burdened by the number of patients that they have to see on a daily basis. We have come up with a novel solution to this problem by creating a chatbot based medical expert system that can have a friendly conversation with a patient for making note of their medical history. For this purpose we have used the Telegram messaging app as a front-end for the patient. Telegram is a widely used, cross-platform, open source and highly secure chat application that lets bots have conversations with users. It does this through a RESTful API which any chat bot can interface with.

What's in the scope?

1. Let's patients chat with the system to extract an accurate history from them.

2. Let's doctors see the medical history of their patients through a similar chat interface.

3. Tries to zero in on the disease that the patient might have by relating symptoms to each other.

And what's out of scope:

1. Does not aim to replace doctors.

2. Cannot detect and accurately extract symptoms of complex diseases like cancer or diseases where manual check-up or tests are essential.

3. Limited by the knowledge base.

There will primarily be 3 classes of users:

Patients

The primary users of the system. Patients will chat with the system through a chat interface.They will communicate their symptoms to the system and the system will ask them relevant questions to figure out a diagnosis. They will chat with the system and engage in a chat for determining the exact disease. Patients are the most important users of the system.

Doctors

The doctors will use the system for accessing data about patients.They will be able to view all the patients whose data they have access to and also view specific details about each patient.

Admin

The admin will be responsible for maintenance and upkeep of the system.The admin will be sole person who will be able to set permissions for doctors who would want to access data of a given patient.

II. Operating Environment and Interfaces

Operating environment will be as follows:

- Any UNIX based operating system. We will be using Debian/Ubuntu.
- Internet connection.
- Python interpreter.
- Various python libraries like python-telegram-api, threading, multiprocess, etc.
- Redis.
- Computer running commodity hardware.

Design and Implementation Constraints

A major concern is maintaining the privacy of user data. Only specific doctors should be able to access the symptoms related data of certain patients and that too with their consent .Admin access to this data will be restricted to the chief maintainers of the system. Thus data will have to be stored in a very secure manner and it cannot be distributed without specific legal permission from patients.The front end of the system (user facing) will depend on the Telegram mobile app. We will need to conform to the standards set by Telegram for chat communication between client and server. All the server side code will be written in Python and the database will be Redis. We will primarily use HTTP get and post requests to send and receive data over the network.

A. Interface between patient and system:

This interface is already created by Telegram, and we will be using the same to take information from the patient and deliver the relevant information back to them.It predominantly features a text box for input of text, a chat window that displays previous conversations, a keyboard for typing responses and a ’send’ button for sending this information to the system.

Interface between doctor and system:

The only information that the doctor expects from the system is information about his/her patients. This will be delivered to them via their email account.

Interface between admin and system:

The admin takes care of assigning doctors to patients. This will be done through a command line interface.

Patient chats with the system:

Stimulus: Patient sends a text message describing his ailment.

Response: Chat bot pulls the users details from the database into active memory for further processing. System will initiate a chat and try to diagnose the patient's disease. This will involve read/write calls to the data base and business logic modules. Once the chat reaches a 'finished' state, an appropriate message will be sent to the user indicating this.

Stimulus: Patient replies with the Telegram in-app keyboard to the question asked by the system.

Response: The chat bot records this response, and after processing is done by the business logic module, an appropriate reply is send back to the user. This might be another question to prompt the patient for more information or a message announcing that the conversation is over.

B. Doctor-system interaction:

Stimulus: A patient whose data has been authorised to be viewed by a particular doctor has just successfully completed a conversation with the system.

Response: The system will fetch the relevant patient details about the patient and return it to the doctor in a readable and understandable format. The information will be sent to the doctor as an email. The responses will typically concern obtaining more information about a particular patient.

C. Admin-system interaction:

Stimulus: Admin wants to authorize a doctor to be able to receive information about a given patient.

Response: System associates doctor credentials with the patient and authorises doctor to access patient information.

III. System Modules and Architecture

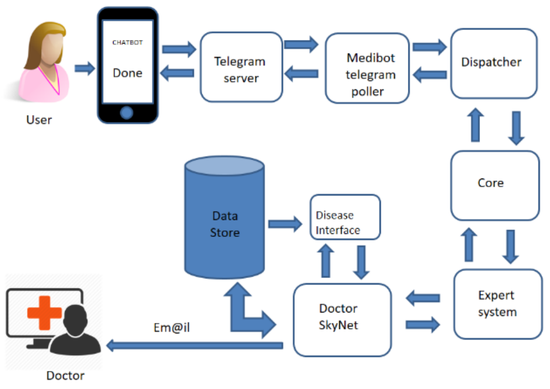


Figure 1. Architecture Diagram of the System

A .Server module

This module is responsible for receiving and sending messages by interacting with the Telegram server. Additionally it is also responsible for dispatching messages to the expert system modules and getting replies from them.

B. Database communication module

This is the module that is completely responsible for read/write calls to the database. The data base module will supply the relevant user data to the chat bot and receive data to be stored in the data base

C. Dispatcher module

The dispatcher acts like a router. It allocates a single object to an independent conversation.

D. Question and response data store

This is a static data store which stores the data about questions and responses.

E. Core module

This module is a state aware module that is spawned on a per conversation basis. It is aware of the state that the conversation is in at any point of time.

F. Expert System module

The Expert System module sits between the core and Doctor Sky Net modules. It takes care of handling conversation states and communication between Core and Doctor SkyNet.

G. Doctor SkyNet module

This is the core module that contains various crucial algorithms for actual functioning of the entire system.

IV. Database schema

The system as a whole relies on two kinds of data – hard coded data and dynamic data. They are represented as follows:

A .Hard coded data

This data either pertains to symptoms, diseases or questions. It highly important data that must be in a particular format. It is the primary knowledge store of the expert system.

It is stored as Python native data structures like dictionaries, lists, strings and numbers – all in dedicated python files. The reason for keeping these hard coded and not in any external database is to completely nullify the possibility of unwanted elements making changes to the data, which might prove disastrous for the system as whole.

The two major types of hard coded data are as follows:

B .Disease data:

Disease data gives information about each disease and its constituent symptoms.

Internally, each symptom for a particular disease is rated as CRITICAL, IMPORTANT and OPTIONAL so as to give the expert system an idea about the most and least important symptoms of a particular disease.

For example, the disease data of *dengue* looks like this:

```
dengue = {
        fever: CRITICAL,
        body_pain:IMPORTANT,
        joint_pain: IMPORTANT,
        pain_behind_eyes: OPTIONAL,
        rash: OPTIONAL,
        body_pain_muscles: OPTIONAL,
        'name': "dengue"
}
```

C. Questions data:

Questions data is the question that will be asked for each symptom in order to confirm the existence of the symptom in the patient.

One question corresponds to one symptom. These questions are divided into top questions and specific questions.

Top questions are for confirming if the patient shows signs of a particular symptom and specific questions are for asking about specific details of a particular symptom.Each question is tagged with the name of the symptom. The tags play an important role in deciding the hierarchy of the symptoms when they are loaded into various expert system data structure like the symptom_validity_table and scratch_pad.

A sample top question and its specific question would look like:

```
# Top question to confirm if body pain exists.
'body_pain':{
'question':"Do you have body pain?",
'response':['Yes','No'],
'response_type':'ruledchar',
}

# Specific question asking where exactly the
#   body pain might be.
'body_pain_head':{
'question':"Does your Head hurt?",
'response':['Yes','No'],
'response_type':'ruledchar',
'serial':0,
}
```

C. Dynamic data

Dynamic data involves the data that is provided to the system by real human users as a result of the conversations that they have with the system. This data is stored in the Redis database as key-value pairs on a per user basis.

There are primarily 4 data tables, or Redis 'hashes', where the data is stored.

D. Per user basic data

This is a Redis hash that is unique to each user.

It is stored in form of chat_id + ":basic_data". This hash contains basic data about the user, it contains the following attributes:

Age , Height, Weight, Gender

E. Per user symptom data

This is a Redis hash that is again, unique to each user.It is stored in the form chat_id + "symptoms". This hash contains keys that are symptoms and the corresponding values of each key corresponds to an integer that that represents the number of times the user has reported that particular symptom till date.The key is not created until the user reports that symptom. An example database looks like this:

Fever – 2

Body_pain – 1

Fatigue – 1

F. Global symptom count

Since we use machine learning algorithms for reporting symtoms, a hash called GLOBAL_SYMPTOM_COUNT keeps a track of the number of times ALL the symptoms have been answered till date.It contains key-value pairs where the keys are symptoms and the values are all initially set to zero.Whenever any user reports a particular symptom, the count of that symptom in this hash is incremented by 1.

G .Global username chat id store

This is a simple key value store that stores the username with the corresponding chat ID.A username-chat ID entry is made into this table as soon as a message arrives in the server.This table helps us keep a track of the number of users we have had and their corresponding chat IDs.

A sample store with two users ‘v0dro’ and ‘jaideepkekre’ will look something like this:

V0dro – 1234

Jaideepkekre - 23456

V.Doctor SkyNet

This module is where the real magic happens. Its primary purpose is to use intelligent algorithms to read the response from the user, keep a track of the symptoms that the user has already answered about, and to serve questions to the user that are most relevant to him based on a data base of diseases that is maintained.Each disease is represented as a ‘bucket’ (using the ‘Buckets’ module as we’ll see later). Each bucket is associated with certain symptoms. The algorithms employed by DoctorSkyNet read the state of these buckets and serve the most relevant question to the user. This helps us narrow down the number of questions that need to be asked by the system in order to reach a possible diagnosis.

The algorithms that are used by DoctorSkyNet can be summarized as follows:

A. Algorithm minus one

- Role: Clustered symptom identifier.
- Finds the symptom that ALL the patients are suffering from.
- Uses Redis.
- Does not interact with buckets.

Steps:

2 Get all symptom, ‘Yes’ count from Redis via ORM.
3 Find symptom with highest Yes count.
4 Return that symptom.

B.Algorithm zero

- Role : Most probable symptom finder , patient history based
- Finds the symptom that the patients is PRONE to.
- Uses Redis.
- Does not interact with buckets.

Steps:

1. Get Patient’s historical symptom, Yes count from Redis via ORM.
2. Find symptom with highest Yes count.
3. Return that symptom.

C. Algorithm one

- Role: Find symptom with highest score across buckets.
- Finds the symptom that will fill largest number of buckets.
- Does not interact directly with buckets.

Steps:

1. Get symptom score dict.
2. Find symptom with highest score.
3. Return that symptom.
4. Update buckets, remove symptom from all score dicts.

D. Algorithm two

- Role : Find Critical symptom with highest score across buckets
- Finds the symptom that will remove largest number of buckets if answered in negative.
- Does not interact with bucket directly

Steps:

1. Get Critical symptom score dict.
2. Find symptom with highest score.
3. Return that symptom.
4. Update buckets, remove symptom from all score dicts.

E . Algorithm three

Role: Find symptom with highest cumulative remaining score in each bucket.

Finds the symptom that will remove largest number of buckets if answered in negative.

Steps:

1. Get Bucket.
2. Find symptom with highest score in Bucket.
3. Record that symptom, score in data structure.
4. If buckets left go to step 1.
5. Find symptom with highest score in data structure
6. Return that symptom

Once the basic data is in place, DoctorSkyNet moves to the second stage and proceeds to query the patient for symptoms based on the above algorithms. It sets the ‘done’ variable to *1* once sufficient data about symptoms has been acquired by the system.

The overall working of DoctorSkyNet can be summarized as follows:

```
whileconversation_in_progress():
update_bucket_fractions()
ifbasic_questions_unanswered():
init_basic_questions_conversation()

ifsymptom_queries_not_over():
init_symptom_query_conversation()
```

Question Interface: The question interface class stores the details of each question and makes it quickly accessible to any other class needing access to questions in a relatable format. It has the same attributes as any question dictionary in the data store.

Disease Buckets: The disease buckets are used for representing diseases. Each disease is assigned its own bucket and each bucket consists of multiple symptoms.

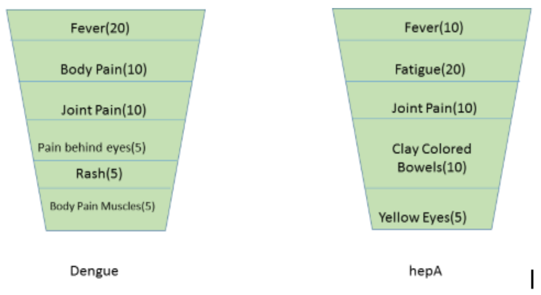
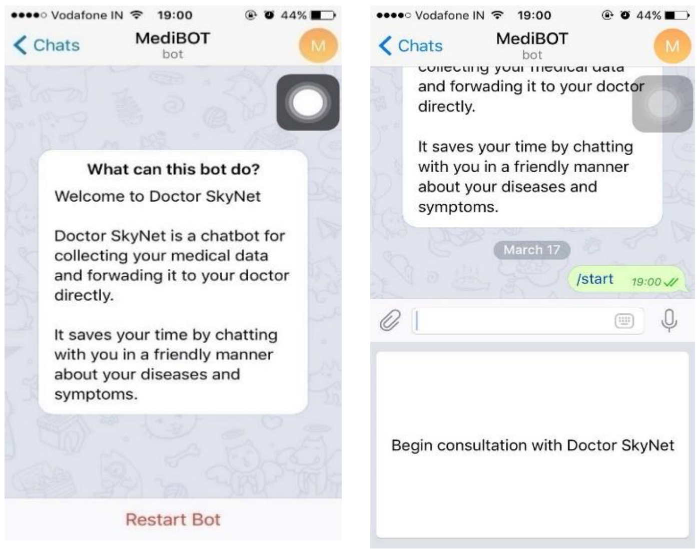


Figure 2. Sample Bucket Design

VI. RESULTS



What can this bot do?

Welcome to Doctor SkyNet

Doctor SkyNet is a chatbot for collecting your medical data and forwarding it to your doctor directly.

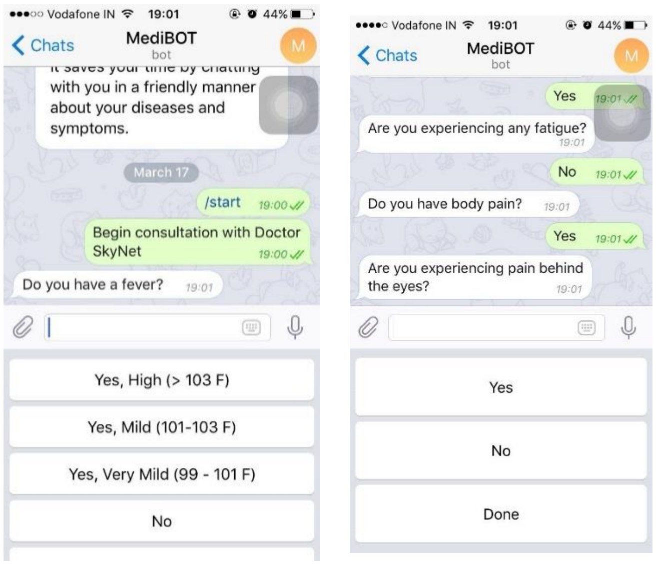
It saves your time by chatting with you in a friendly manner about your diseases and symptoms.

collecting your medical data and forwarding it to your doctor directly.

It saves your time by chatting with you in a friendly manner about your diseases and symptoms.

Begin consultation with Doctor SkyNet

Figure 3. System UI



It saves your time by chatting with you in a friendly manner about your diseases and symptoms.

March 17

/start 19:00

Begin consultation with Doctor SkyNet 19:00

Do you have a fever? 19:01

Yes 19:01

Are you experiencing any fatigue? 19:01

No 19:01

Do you have body pain? 19:01

Yes 19:01

Are you experiencing pain behind the eyes? 19:01

Figure 3 . System GUI

VII. CONCLUSION

After studying various expert systems, chat bots and getting feedback from people involved in the medical profession, it can be concluded that:

- A chat bot based medical expert system can be very useful for reducing the stress that doctors have to go through on a daily basis.
- However, the results of the system should still be verified by a legitimate doctor before they are furnished to the patient.

Such a system can be perfected to incorporate all sorts of diseases

REFERENCES

[1] Hallili, Amine. "Toward an ontology-based chatbot endowed with natural language processing and generation." *26th European Summer School in Logic, Language & Information*. 2014.

[2] Kohane, Isaac Samuel. *Temporal reasoning in medical expert systems*. Boston Univ., MA (USA), 1987.

[3] Shortliffe, Edward H., and Lawrence M. Fagan. "Expert systems research: modeling the medical decision making process." (1982).

[4] Hill, Jennifer, W. Randolph Ford, and Ingrid G. Farreras. "Real conversations with artificial intelligence: A comparison between human–human online conversations and human–chatbot conversations." *Computers in Human Behavior* 49 (2015): 245-250.

[5] Dutta, Soumitra. "Temporal reasoning in medical expert systems."*Engineering of Computer-Based Medical Systems, 1988., Proceedings of the Symposium on the*. IEEE, 1988.

[6] http://www.wikipedia.org/

[7] http://www.obofoundry.org/

[8] https://github.com/python-telegram-bot/python-telegram-bot

[9] https://docs.python.org/2/