PROJECT REPORT

**TOPIC: BUILDING A CHATBOT USING NLP AND**

**NEURAL NETWORKS IN PYTHON**

**BY: RAJAS V PRASAD**

**Student Of**

**Medical Electronics**

**Engineering**

**M S Ramaiah Institute**

**Of Technology**

**Bangalore 560054**

***TABLE OF CONTENTS:***

- Abstract

- Introduction

- Methodology

- Results

**ABSTRACT:**

Our main idea is to create a chatbot which serves the purpose of suggesting the medications and if further required for user purpose, it will also suggest the related doctors to the symptoms specified by the user. Setting up of the json file is main as it provides the interface of communication between the user and the chatbot. Eg: user can say ‘hey’ to the bot or even can say ‘goodbye’ and mainly they can type in their symptoms like common cold, etc.

This chatbot will mainly look into diseases/infections like :

* Common cold
* Fever
* Diabetes
* Depression
* Asthma

So we provide different tags for each kind of interaction with the bot. Eg: “greeting” tag is mainly used to greet the bot where each tag is specified with patterns like ‘hi’, ‘hello’, etc.

Essentially we provide a couple of examples on how a user can greet the chatbot. The chatbot is going to consume that as training data and understands this is how greeting looks like and will adapt to the data the user provides. Eg: if the chatbot recognizes enough examples like ‘hi’, ‘hello’, ‘hey’ it will generally understand that it is being greeted. We also have responses which are static and are hard coded. These will be the exact messages the bot replies.

While the code is typed in certain concepts are to be understood like:

Parsing- It’s processing a piece of python program and converting the code into a machine language. In general parse is a command for dividing given a program code into small piece of code for analyzing correct syntax.

Json- It’s a java script representation file having arrays and scalar data. It’s easy to read and write also easy for software to parse and generate.

Serialization- It’s the process of converting a data object (i.e. python object, tensorflow model) into a format that allows us to store or transmit data and then recreate the object when needed.

The most important concept of this project is to build a neural network where there are many network of neurons i.e. perceptron which may come from the environment of the information processing system or maybe the O/P of other perceptrons. So a neural network has layers of connected perceptrons and usually in a multilayer perceptron there are 3 layers: I/P layer, Hidden layer & O/P layer.

The main agenda to implement these above mentioned concepts are by importing libraries to our python kernel so that we can accordingly build a chatbot of our requirement. The implementation/methodology will explain the process in detail.

**INTRODUCTION:**

Chatbots are simulators which can understand human language, process it and interact back with humans while performing specific tasks.

Architecture of Chatbots:

Corpus

Chat Window or Session

NLP Model

Interface

Application DB

**NLP model** is either built under neural network or machine learning algorithm which basically understands the question for which it has to pick data and give an answer.

In chatbots data is known as **Corpus/dictionary** which is a repository of information (or) the data the chatbot uses to give answers.

**Application DB** is the database which hosts:

* Application
* Details bridging the gap b/w application to send it to NLP model.

The NLP model uses the **Interface** to push out answers either in a text form (or) voice form.

**Chat window** is where the results are shown in form of texts or voice.

How does a Chatbot work?

**Steps:**

* Import Corpus

Corpus is the training data needed for the chatbot to learn. Without corpus the chatbot will not reply to the user

* Preprocess The Data

This part will handle the cleaning of data where only relevant data is considered.

* Text Case Handling

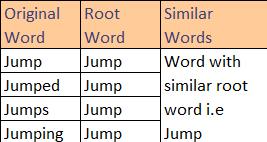
Once the relevant data is provided then this in chatbot will convert all the data coming in as I/P to either upper or lower case. This avoids misinterpretation of words if spelt under lower or upper cases.

* Tokenization

It is the structures process of converting a sentence into individual collection of words. If there is individual collection of words they can be jumbled upon to make a sentence of it’s own which works well in English language seen in python and chatbot gives good answers based upon that. So a lot of sentences in the corpus are broken down into each individual word because the chatbot should answer a specific question rather than answering a random question.

* Stemming

It’s the process of finding similarities b/w words with same root words.



Here we have many words like Jump, Jumping, Jumps & Jumped the common root word for all of them is “Jump”. From this the root word we can call all the 4 words. So instead of finding millions of words we find the root word. We use the process of lemmatization which is almost same as stemming but the only difference is that stem may not be an actual word whereas, lemma is an actual language word.

* Bag Of Words (BOW)

The concept is you are holding a couple of words and you take it all and put in a bag. It is defined as process of converting words into numbers by generating vector embeddings from the tokens generated.

In machine learning also deep learning algorithm the machine does not understand English words as it is, so they should be broken down into mathematical vectors.

To generate a BOW let’s take an example:

blog

a

Is

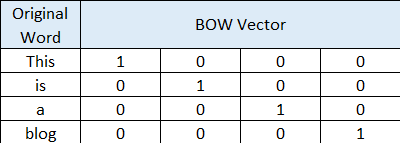
This

This is a blog

(We have broken this sentence into single words)

(Sentence) (Tokens)

BOW



This entire vector tells us the call of words in order from the matrix and there is also something for sticking them together called ‘dot operations’ or ‘matrix multiplication dot operation’ where we bring two vectors together.

* One Hot Encoding

This is the process by which categorical variables are converted into a form that ML algorithms use.

So we know that BOW creates a vector and taking the vector and passing into ML algorithm so that it understands this is a particular sentence. Even the order of words is not important but individual words are important.

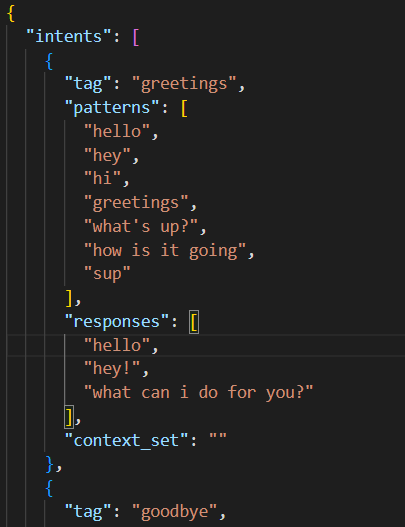
**METHODOLOGY:**

**Step 1: Creating the intents.json file**

The intents.json file consists of “tag” which are categories and each tag has “patterns” & “responses”. The patterns will serve as training data and it is mainly for understanding what a greeting looks like when the user types in. The pattern consists of The uppercase or lowercase letters which may not be cautious as we reduce those words to stems and later tokenize them. The responses are a collection of static responses which is typed into the json.file which will be the exact messages the chatbot returns.

So we create carefully a number of tag’s like “greetings” tag, “goodbye” tag, etc.

Here’s a preview of the intents.json file :



As you can see there are different tag’s and each of them have patterns and responses.

**Step 2: Creating the training data by importing libraries**

We use ‘import random’ which chooses random responses.

We use ‘import json’ so that our model understands a json file.

We use ‘import pickle’ for serialization (converts a python object into a byte stream to store in a file/directory)

We use ‘import numpy’ which is a python library used for working with arrays.

And finally ‘import nltk’ which is the natural language toolkit consisting of human language data and contains text processing libraries for tokenization, parsing, classification, stemming, etc.

‘punkt’ is a sentence tokenizer which divides a text into a list of sentences.

‘wordnet’ is used as a large word database of English nouns, adjectives, adverbs and verbs.

‘omw-1.4’ is an open multilingual wordnet.

nltk.stem is present in the python package which involves process of stemming.

WordNetLemmatizer involves the process of lemmatization which is similar to stemming but it links words with similar meanings to one word.

tensorflow is a software which has particular focus on training and inference of deep neural networks.

‘import Sequential’ allows to create models layer by layer.

‘import Dense’ is the only actual network layer in the model. The dense layer feeds all the O/P’s from the previous layer to all it’s neurons, each neuron provides one O/P to the next layer. In short Dense layer is the most basic layer in neural networks.

‘import Activation’ where activation is a non-linear function applied over I/P data coming to a particular neuron and O/P from the function is served as I/P to the next layers.

‘import Dropout’ this will dropout some nodes of the neural network.

‘import SGD’ is the stochastic gradient descent used to update weight so that there is minimum error b/w actual O/P and predicted O/P. It provides the best fit b/w predicted & actual O/P.

**Step 3: Load The Training Data**

We will first lemmatize the words and then load the json file.

We are reading the json file as text by passing the loads function and as a result we get the json object which is a dictionary in python.

Now we create 3 empty lists. ’words = [ ]’ is For all the words we are going to have. ‘classes = [ ]’ is for classes that we are going to have. ‘documents = [ ]’ will be for the combinations.

Now we have to iterate over the intent by imagining the intents.json file as a dictionary in python where "intents" in intents.json is the object, then we need to access the key intents where in each intent we have sub keys and sub values as sub dictionaries. Then we consider all the pattern in intents. For each of those patterns we give a word list to be tokenized meaning that you're getting a text which is split up into individual words eg: "Hey i am thomas" so it takes here individual words- Hey i am thomas. So this will give us a collection of words where we add this to the words list. The extend() adds all the elements of an iterable list, tuple, string etc to the end of the list. So here all the words list is added to the list ‘words = []’. Then we do the same where we append("*inserts a single element into an existing list. The element will be added to the end of the old list rather than being returned to a new list*") a tuple first for all the word\_list and then also the classes of the particular intent tag nothing but the category of the tag. After that we check if the intent['tag'] is already in the classes = [] list. So if intent['tag'] not in classes then we append the intent['tag'] to the classes. Once we print documents then we get like [(['hello'], 'greetings'), (['hey'], 'greetings'), (['hi'], 'greetings'), (['greetings'], 'greetings'), (['what', "'s", 'up', '?'], 'greetings'), (['how', 'is', 'it', 'going'], 'greetings'), (['sup'], 'greetings')] which represents the greetings tag in intents.json where the patterns are supplied as training data.

**Step 4: Preparing The Training Data**

Here we first lemmatize the word for every word in words list and if this word is not in the ignore letters list then remove the duplicates where ‘set’ eliminates the duplicates and ‘sorted’ turns it back into a list and sorts it. now if we print this we should get a list of lemmatized words. Similarly we print the classes too.

Then we save the classes and words in separate files using pickle. Now we create the file in writing binary mode i.e. 'wb'.

So we have a lot of characters, words and classes but they are not numerical values. Hence we cannot feed anything to the neural network and expect it to work. We need to represent these words as numerical values by using Bag Of Words(BOW) where we set individual words indices (or) individual word values to either 0 or 1 depending on, if it's occurring in that particular pattern and then we do the same thing for the classes.

Now we create ‘training = [ ]’ as an empty list. Then the ‘output\_empty = [0]\*len(classes)’ will be the template of zeros and we need as many zeros as there are for classes.

Now we create an empty bag for each of those combinations (i.e. documents list) where we create an empty BOW. Next we set the word\_patterns which are what we find in document[] list at index 0. Next we want to know for each word if it occurs in the pattern. ‘bag.append(1)’ is used where if this word occurs in word patterns and otherwise we're going to say bag.append(0).

‘output\_row = list(output\_empty)’ we do this to copy the list so we are not type casting it instead copying it. Next we want to know the class at index 1 once we know the index we set the index in the output row as 1 and then we append the whole thing to the training list.

Now once we run this loop the document data is going to be in the ‘training = []’ list and we can work with this training list to train the neural network.

Then we use ‘random.shuffle(training)’ to shuffle the training data. Then we turn the training data into a numpy array. After that we split the training data into x & y values.

**Step 5: Building The Neural Network**

First we create a simple sequential model. Then we are going to add a couple of layers where 1st layer is I/P layer then the dense layer with 128 neurons of shape dependent on the size of the training data for x and we also specify the activation function to be a rectified linear unit. After that we use dropout to prevent overfitting of nodes of neurons. Then we add another dense layer with 64 neurons and activation function as 'relu' and another dropout layer. After that we add another dense layer where we want to have as many neurons as much as there are training data inputs and we use the activation function here as 'softmax' which allows us to sum up the results in the output layer so that they all add up to one. We can sort out the percentages of how likely it is to have the O/P or that result.

Then we define a stochastic gradient descent optimizer where we enter a couple of parameters where 0.01 is the learning rate and we have a decay of (1e-6) and we specify a momentum of 0.9. Now we compile them all where we choose a loss function of 'categorical\_crossentropy' and optimizer is set to the SGD that we defined and the metrics we are interested in are accuracy.

Then we use model.fit where we use the training data without specifying the index then we have some output which is the train\_y data and we specify epochs of 200 where we feed the same data 200 times into the neural network with a batch size of 5. Then we specify verbose as 1 so that we get a medium amount of information. Once we ‘print’ it here it trains the neural network for 200 times.

**Step 6: Building The Chatbot**

Here we mainly build the chatbot application itself which uses the training model.

We import the same libraries again, but instead of ‘from tensorflow import Sequential’ we use ‘from tensorflow import load\_model’ as our model has been fed with the training data already.

After this we again create a lemmatizer. Now we have to load words classes and model which we have saved already. Then we load the file in reading binary mode i.e 'rb'. Next load the model we saved i.e. ‘chatbotmodel.h5’.

Now we use different functions because the model is trained but we need to use it the right way because what we get is numerical data but we want to end up with words. For that we use 4 different functions.

First is the function for cleaning up the sentences. Here Essentially we are Tokenizing the sentence, then we lemmatize the words. This is how we clean up a sentence.

Second is the function for BOW. This will convert a sentence into a list full of zeros and ones that indicate if the word is there or not. So We create an initial bag of zeros as many as there are individual words that can be because we need to set for each word either zero or one. bag[i] is one otherwise it is zero. Then we return a numpy array of the bag.

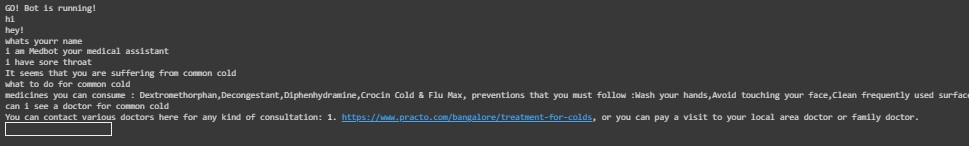
The Third is the function for predicting where we use the previous 2 functions. So we predict a class based on the sentence and we say to give back a bag of words which is bag of words as the sentence because that's what we feed into a neural network. Then we obtain prediction while passing a numpy array of the bag of words. After that ‘ERROR\_THRESHOLD = 0.25’ allows for a certain uncertainty but if that uncertainty is too high we will not take it into a result. So since we have a softmax function where each position in the output is going to be likelihood of that class being the result and if this is below 25 percent we don’t want to take this in order to realize that if the result is larger than the threshold. Then we sort the results where the key or sorting is an anonymous function a lambda expression. So we take the first index everytime and ‘reverse = true’ will sort it in reverse order i.e. descending order. We also have a return list which is empty. We save for each results . So what we did is we created a bag of words, we predict the result based on those bag of words then we have a certain threshold that we don't want to have too much uncertainity so we enumerate all the results to get the index for the class and we want to sort by probability in reverse order so that we have the highest probability first and then we have to return a list full of classes and probabilities.

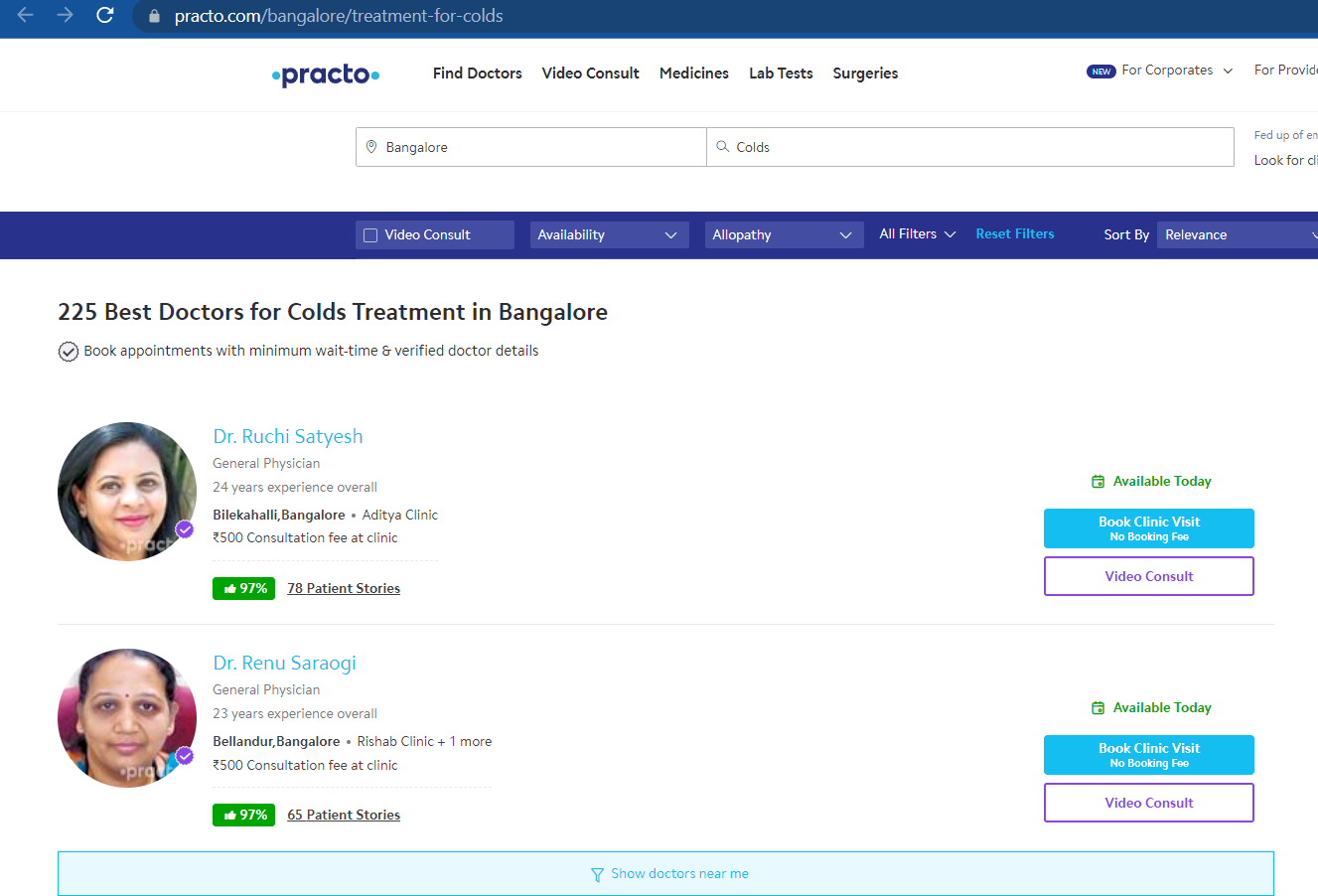
The Forth function is to get responses from the bot. Once we derive responses to be understood by the intents file accordingly, we give a print statement saying ‘*Go! Bot is running’*  and finally we print the results.

**RESULTS:**

The results shown were conversations with the bot as well as the remedies provided for each symptom the user shared.

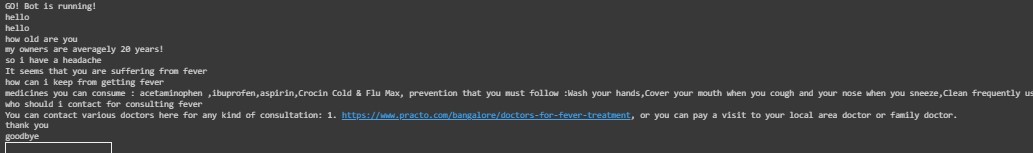
1. Common cold

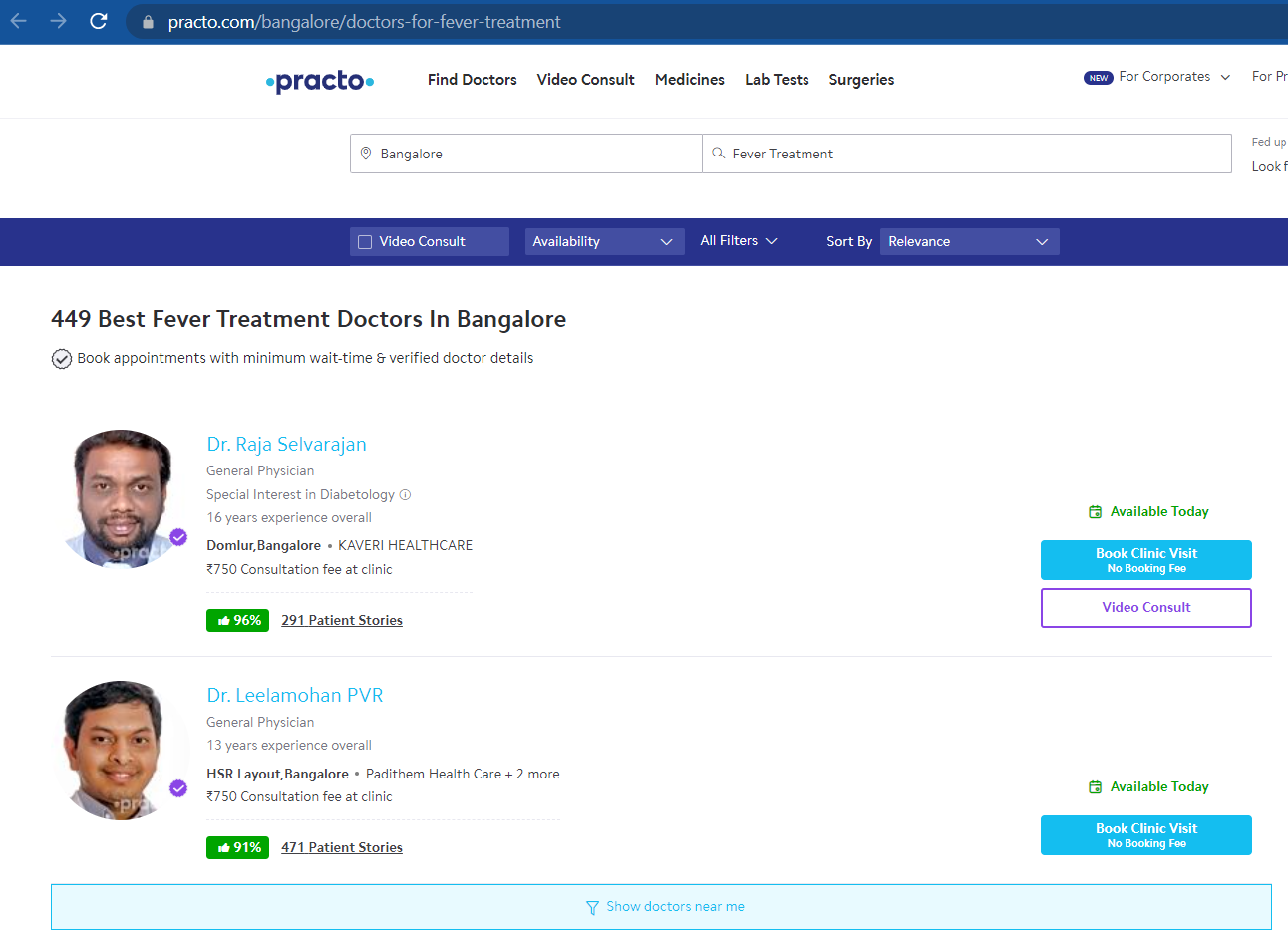




The above results show the preventions for common cold and also once asked for consultation will show the relevant doctors after clicking the website link provided by the bot.

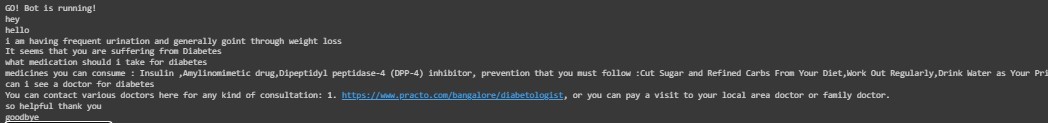
1. Fever

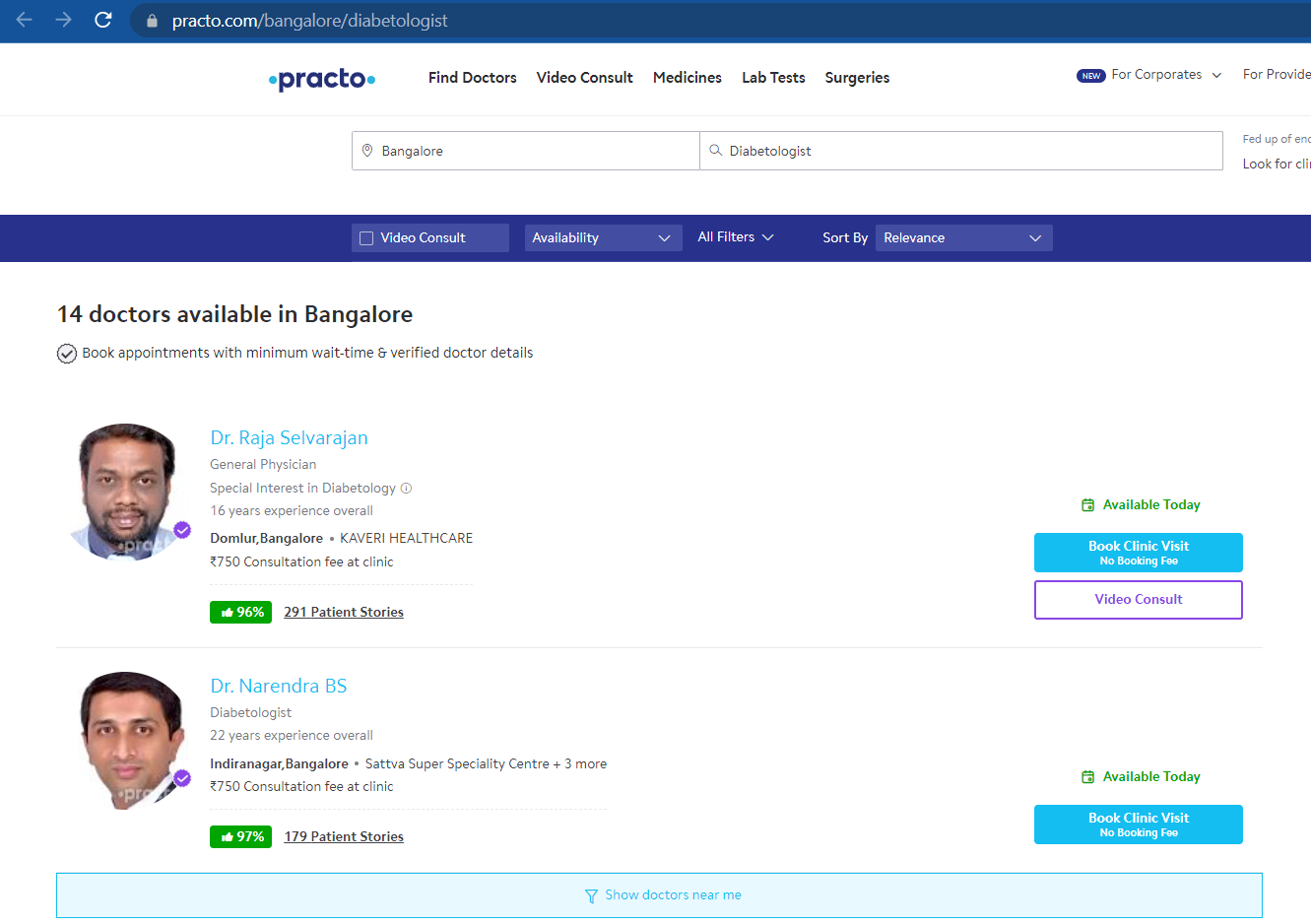




The above results show the preventions for fever and also once asked for consultation will show the relevant doctors after clicking the website link provided by the bot.

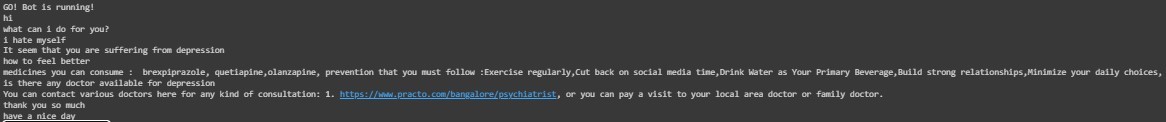
1. Diabetes

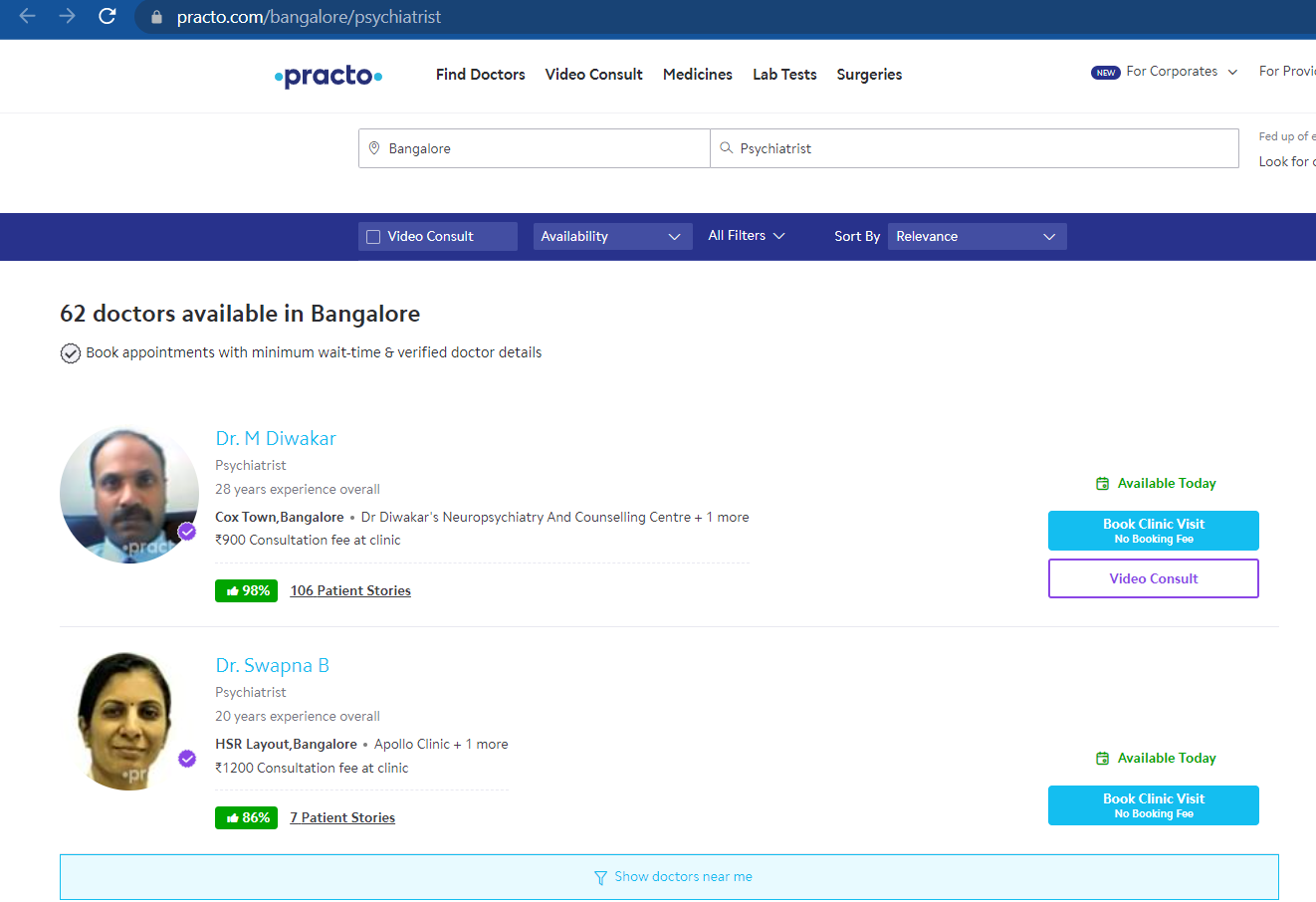




The above results show the preventions for Diabetes and also once asked for consultation will show the relevant doctors after clicking the website link provided by the bot.

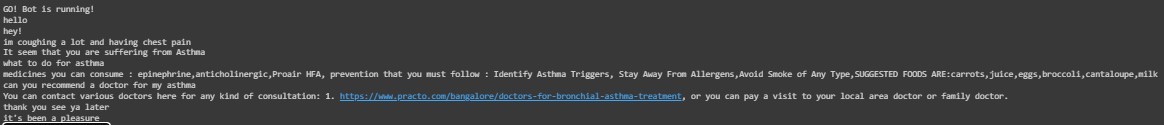
1. Depression

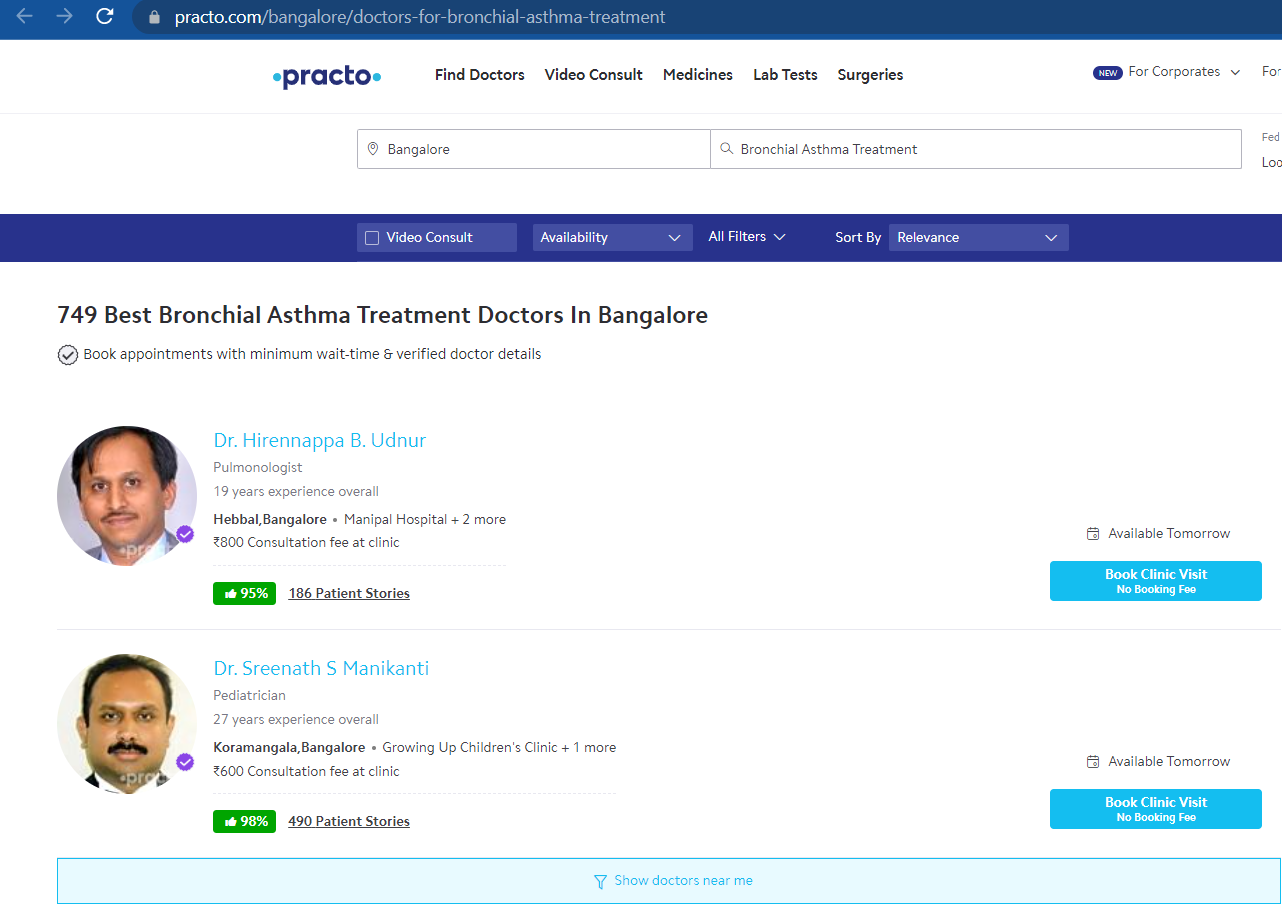




The above results show the preventions for Depression and also once asked for consultation will show the relevant doctors after clicking the website link provided by the bot.

1. Asthma





The above results show the preventions for Asthma and also once asked for consultation will show the relevant doctors after clicking the website link provided by the bot.

Hence this concludes the usage of the bot as it provided relevant details for each symptom of the user once provided. The chatbot was only provided with the knowledge in detecting symptoms for diseases only for Common cold, Fever, Diabetes, Depression & Asthma. This is how a healthcare chatbot works using neural networks in machine learning.