**MINI PROJECT REPORT**

**On**

**“MEDICAL CHATBOT”**

**Submitted by**

**Name: DEEPAK SHARMA**

**Roll No: 171500094**

**Name: SURYANSH CHATURVEDI**

**Roll No: 171500349**

Department of Computer Engineering & Applications

**Institute of Engineering & Technology**



**GLA University**

**Mathura- 281406, INDIA**

**2019**

**Acknowledgement**

I thank the almighty for giving me the courage and perseverance in completing the project.

This project itself is acknowledgements for all those people who have give us their heartfelt co-operation in making this project a grand success.

I extend my sincere thanks to Piyush Vashishta, Assistant Professor in C.S.E. Department for providing valuable guidance at every stage of this project work. I am profoundly grateful towards the unmatched services rendered by her.

Last but not least , we would like to express our deep sense of gratitude and earnest thanks giving to our dear parents for their moral support and heartfelt cooperation in doing the main project.

**ABSTRACT**

Obviously, there are countless cases where a digital personal assistant or a chatbot could help physicians, nurses, patients or their families. **Better organization of patient pathways, medication management, help in emergency situations or with first aid, offering a solution for simpler medical issues:** these are all possible situations for chatbots to step in and ease the burden on medical professionals.

The general idea is that in the future, these [talking or texting smart algorithms](https://medicalfuturist.com/chatbots-health-assistants/) [might become the first contact point for primary care](https://medicalfuturist.com/chatbots-health-assistants/). Patients will not get in touch with physicians or nurses or any medical professional with every one of their health questions but will turn to chatbots first. If the little medical helper could not comfortably respond to the raised issues, it will transfer the case to a real-life doctor.

In this way we design a health care chatbot which predict the disease and recommend a special doctor for that particular disease with their contact details.

## CONTENTS

1. Introduction
   1. General Introduction to the topic

1.2 Area of Computer Science

1.3 Types of Chatbots

1. Hardware and Software Requirements
2. Datasets
3. Project Screenshots
4. Applications of Medical Chatbot
5. Conclusion
6. References

## CHAPTER 1 INTRODUCTION

### **General Introduction**

Chatbot who predict disease as well as recommend a doctor.

This project detect the disease as per symptoms and recommend a specialist doctor . Chatbots are gradually being adopted into the healthcare industry and are generally in the early phases of implementation. Healthcare has become an attractive market for companies developing chatbot applications for patients and clinicians. The majority of current and emerging use cases appear to focus on checking patient symptoms. Specifically, natural language processing is used to help diagnose a user based on the symptoms he or she provides. The dataset is made with the help of some websites .In this model we will try to predict disease as per given symptoms and try to recommend a specialist doctor for that disease.

* + **Area of Computer Science**

This project mainly covers the Area of Machine Learning of Computer Science. Through this project, we are trying to climb one step in this vast field of Computer Science. It is totally an Automated Project.

**Machine Learning**: Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. **Machine learning focuses on the development of computer programs** that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. **The primary aim is to allow the computers learn automatically** without human intervention or assistance and adjust actions accordingly.

### **Types of Chatbots**

Chatbots can talk to your customers for you. In this lies their ability to handle various aspects of customer relations, substituting a number of employees with a single bot.In this article, we look at the essence of chatbots: how and where they work, which industries can benefit from them, and where they’ve already been successful.

There are two types of chatbots — those built into messengers (Slack, Telegram, Discord, Kik, etc.) and standalone applications. We advise building a chatbot in a messenger first because there are a lot of people using them already, so your service will be able to receive the recognition it deserves. Just look at these statistics from April 2018 showing the [number of monthly messenger users.](https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/)

## CHAPTER 2

## Hardware and Software Requirements

## Software

**Introduction to Surprise**

[Surprise](http://surpriselib.com/) is a Python [scikit](https://www.scipy.org/scikits.html) building and analyzing recommender systems that deal with explicit rating data.

[Surprise](http://surpriselib.com/) was designed with the following purposes in mind**:**

1. Give users perfect control over their experiments. To this end, a strong emphasis is laid on [documentation](http://surprise.readthedocs.io/en/stable/index.html), which we have tried to make as clear and precise as possible by pointing out every detail of the algorithms.
2. Alleviate the pain of [Dataset handling](http://surprise.readthedocs.io/en/stable/getting_started.html" \l "load-a-custom-dataset). Users can use both *built-in* datasets ([Movielens](http://grouplens.org/datasets/movielens/), [Jester](http://eigentaste.berkeley.edu/dataset/)), and their own *custom* datasets.
3. Provide various ready-to-use [prediction algorithms](http://surprise.readthedocs.io/en/stable/prediction_algorithms_package.html) such as [baseline algorithms](http://surprise.readthedocs.io/en/stable/basic_algorithms.html), [neighborhood methods](http://surprise.readthedocs.io/en/stable/knn_inspired.html), matrix factorization-based ( [SVD](http://surprise.readthedocs.io/en/stable/matrix_factorization.html" \l "surprise.prediction_algorithms.matrix_factorization.SVD), [PMF](http://surprise.readthedocs.io/en/stable/matrix_factorization.html" \l "unbiased-note), [SVD++](http://surprise.readthedocs.io/en/stable/matrix_factorization.html" \l "surprise.prediction_algorithms.matrix_factorization.SVDpp), [NMF](http://surprise.readthedocs.io/en/stable/matrix_factorization.html" \l "surprise.prediction_algorithms.matrix_factorization.NMF)), and [many others](http://surprise.readthedocs.io/en/stable/prediction_algorithms_package.html). Also, various [similarity measures](http://surprise.readthedocs.io/en/stable/similarities.html) (cosine, MSD, pearson…) are built-in.
4. Make it easy to implement [new algorithm ideas](http://surprise.readthedocs.io/en/stable/building_custom_algo.html).
5. Provide tools to [evaluate](http://surprise.readthedocs.io/en/stable/model_selection.html), [analyse](http://nbviewer.jupyter.org/github/NicolasHug/Surprise/tree/master/examples/notebooks/KNNBasic_analysis.ipynb/) and [compare](http://nbviewer.jupyter.org/github/NicolasHug/Surprise/blob/master/examples/notebooks/Compare.ipynb) the algorithms performance. Cross-validation procedures can be run very easily using powerful CV iterators (inspired by [scikit-learn](http://scikit-learn.org/) excellent tools), as well as [exhaustive search over a set of parameters](http://surprise.readthedocs.io/en/stable/getting_started.html" \l "tune-algorithm-parameters-with-gridsearchcv).

The name SurPRISE roughly stands for Simple Python RecommendatIon System Engine.

**Installing surprise module**

To install surprise,we have to run “pip install scikit-surprise” in CMD or “conda install –c conda-forge scikit-surprise” while using Anaconda Prompt.

**Modules in Surprise and their functionality**

**SVD(Single Value Decomposition)**

SVD in the context of recommendation systems is used as a collaborative filtering algorithm. For those of you who don’t know, collaborative filtering is a method to predict a rating for a user item pair based on the history of ratings given by the user and given to the item. Most collaborative filtering algorithms are based on user-item rating matrix where each row represents a user, each column an item. The entries of this matrix are ratings given by users to items.

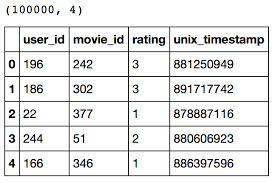
****

Fig 2.1:-Collaborative Filtering is applied on this type of datasets

SVD is a matrix factorization technique.For the purpose of the recommendation systems however, we are only interested in the matrix factorization part keeping same dimensionality. The matrix factorization is done on the user-item ratings matrix. From a high level, matrix factorization can be thought of as finding 2 matrices whose product is the original matrix.



Fig 2.2:- User Product matrix

Fig 2.3 :-Factorised matrix for users and movies with 2 latent features

Now to predict rating of any movie for a user we can find the estimate rating by finding the product of latent features of user with that movie.

**Defining the SVD model**

SVDAlgo=SVD(n\_factors,random\_state)

Parameters:

1. **n\_factors** – The number of factors. Default is 100.
2. **random\_state** (int, RandomState instance from numpy, or None) – Determines the RNG that will be used for initialization. If int,random state will be used as a seed for a new RNG. This is useful to get the same initialization over multiple calls to fit(). If RandomState instance, this same instance is used as RNG. If None, the current RNG from numpy is used. Default is None.

Here,SVDAlgo stores the SVD model which can be trained using training dataset.

**Reader class**

The Reader class is used to parse a file containing ratings.

Such a file is assumed to specify only one rating per line, and each line needs to respect the following structure: (user , item , rating , [timestamp])

where the order of the fields and the separator (here ‘,’) may be arbitrarily defined (see below). brackets indicate that the timestamp field is optional.

reader = Reader(line\_format='user item rating timestamp', sep=',', skip\_lines=1)

reader = Reader(line\_format='user item rating timestamp', sep=',', skip\_lines=1)

Parameters:

1. **line\_format (string)** – The fields names, in the order at which they are encountered on a line. Please note that line format  is always space-separated (use the sep parameter). Default is ‘user item rating’.
2. **sep (*char*)** – the separator between fields. Example : ‘,’.
3. **skip\_lines (int, optional)** – Number of lines to skip at the beginning of the file. Default is 0.

**Dataset module**

The dataset module defines the Dataset class and other subclasses which are used for managing datasets.

**Classmethod:load\_from\_file(**file\_path, reader)

Load a dataset from a (custom) file.

Use this if you want to use a custom dataset and all of the ratings are stored in one file. You will have to split your dataset using the split method.

Parameters:

* **file\_path** (string) – The path to the file containing ratings.
* **reader** (Reader) – A reader to read the file

**Classmethod:build\_full\_trainset()**

Do not split the dataset into folds and just return a trainset as is, built from the whole dataset.It returns TrainSet class object.

Ex:-

reader = Reader(line\_format='user item rating timestamp', sep=',', skip\_lines=1)

dataset=Dataset.load\_from\_file(‘C:\\Desktop\\ratings.csv’,reader=reader)

trainset=dataset.build\_full\_trainset()

**Trainset class**

A trainset contains all useful data that constitute a training set.

It is used by the fit() method of every prediction algorithm. You should not try to build such an object on your own but rather use the Dataset.folds() method or the Dataset.build\_full\_trainset() method.

Trainsets are different from Datasets. You can think of a Dataset as the raw data, and Trainsets as higher-level data where useful methods are defined. Also, a Dataset may be comprised of multiple Trainsets (e.g. when doing cross validation).

Trainset class contains:-

1. **ur**:- The users ratings. This is a dictionary containing lists of tuples of the form (item\_inner\_id, rating). The keys are user inner ids.
2. **ir**:- The items ratings. This is a dictionary containing lists of tuples of the form (user\_inner\_id, rating). The keys are item inner ids.
3. **global\_mean**:-The mean of all ratings.
4. **all\_items()**:-Generate function to iterate over all items.
5. **all\_ratings()**:-Generate function to iterate over all ratings.
6. **all\_users()**:-Generate function to iterate over all users.
7. **to\_inner\_iid(**riid**)**:-Convert an item raw id to an inner id.riid must be string.It returns inner id(int).
8. **to\_inner\_uid(**ruid**)**:-Convert an user raw id to an inner id.ruid must be string.It returns inner id(int).
9. **to\_raw\_iid(**iiid**)**:-Convert an item inner id to an raw id.iiid must be int.It returns raw id(str).
10. **to\_raw\_uid(**iuid**)**:-Convert an user inner id to a raw id.iuid must be int.It returns raw id(str).
11. **n\_users:-**Total number of users.
12. **n\_ratings:-**Total number of ratings.
13. **n\_items:-**Total number of items.

**Introduction to Tkinter**

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter outputs the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

To create a tkinter:

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

There are two main methods used you the user need to remember while creating the Python application with GUI:-

1. Tk(): To create a main window, tkinter offers a method ‘Tk()’. To change the name of the window, you can change the className to the desired one. The basic code used to create the main window of the application is:

m=tkinter.Tk() where m is the name of the main window object

1. mainloop(): There is a method known by the name mainloop() is used when you are ready for the application to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event till the window is not closed.

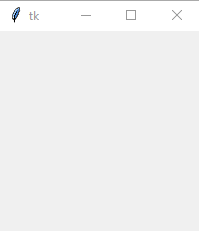
m.mainloop()

fig 2.4 Window appears on executing mainloop()

There are mainly three geometry manager classes class:-

1. **pack() method**:It organizes the widgets in blocks before placing in the parent widget.
2. **grid() method**:It organizes the widgets in grid (table-like structure) before placing in the parent widget.
3. **place() method**:It organizes the widgets by placing them on specific positions directed by the programmer.

There are a number of widgets which you can put in your tkinter application. Some of the major widgets are explained below:

1. **Button**:To add a button in your application, this widget is used.

The general syntax is:

w=Button(master, option=value)

master is the parameter used to represent the parent window.

Parameters:

* activebackground: to set the background color when button is under the cursor.
* activeforeground: to set the foreground color when button is under the cursor.
* bg: to set he normal background color.
* command: to call a function.
* font: to set the font on the button label.
* image: to set the image on the button.
* width: to set the width of the button.
* height: to set the height of the button.

1. **Entry**:It is used to input the single line text entry from the user.. For multi-line text input, Text widget is used.The general syntax is:

w=Entry(master, option=value)

Parameters:

* bd: to set the border width in pixels.
* bg: to set the normal background color.
* cursor: to set the cursor used.
* width: to set the width of the button.
* height: to set the height of the button.

1. **Label**: It refers to the display box where you can put any text or image which can be updated any time as per the code.The general syntax is: w=Label(master, option=value)

Parameters:

* bg: to set he normal background color.
* bg to set he normal background color.
* font: to set the font on the button label.
* image: to set the image on the button.
* width: to set the width of the button.
* height: to set the height of the button.

1. **Text**: To edit a multi-line text and format the way it has to be displayed.The general syntax is: w =Text(master, option=value)

Parameters:

* highlightcolor: To set the color of the focus highlight when widget has to be focused.
* insertbackground: To set the background of the widget.
* bg: to set he normal background color.
* font: to set the font on the button label.
* image: to set the image on the widget.
* width: to set the width of the widget.
* height: to set the height of the widget.

1. **Listbox**: It offers a list to the user from which the user can accept any number of options.The general syntax is: w = Listbox(master, option=value)

Parameters:

* highlightcolor: To set the color of the focus highlight when widget has to be focused.
* bg: to set he normal background color.
* bd: to set the border width in pixels.
* font: to set the font on the button label.
* image: to set the image on the widget.
* width: to set the width of the widget.
* height: to set the height of the widget.

**Messagebox module:-** To show a minimalistic Tkinter message box, use the function showinfo() and showerror() where the parameters are the window title and text.

Ex:-

import tkinter

from tkinter import messagebox

root = tkinter.Tk()

root.withdraw()

messagebox.showerror("Error", "Error message")

messagebox.showwarning("Warning","Warning message")

messagebox.showinfo("Information","Informative message")

**Hardware**

1. Personal computer with 4 GB RAM

2. Internet Connection

**CHAPTER 3**

**DATASET**

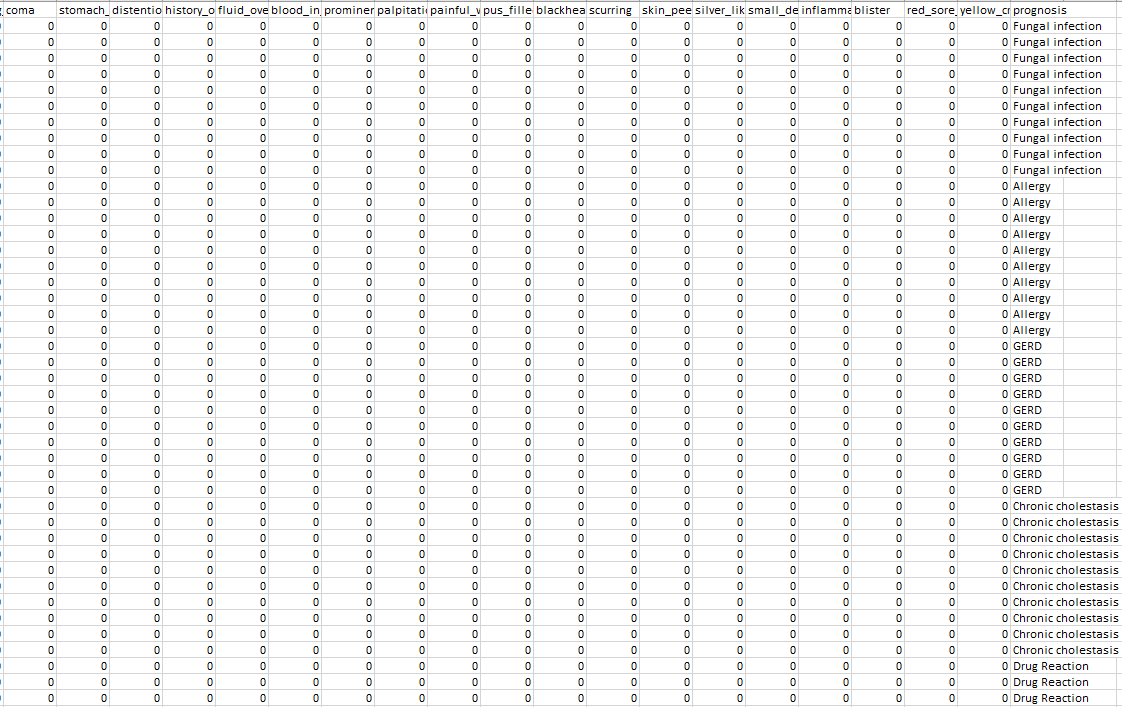


fig 3.1 DATASET FOR SYMPTOMS

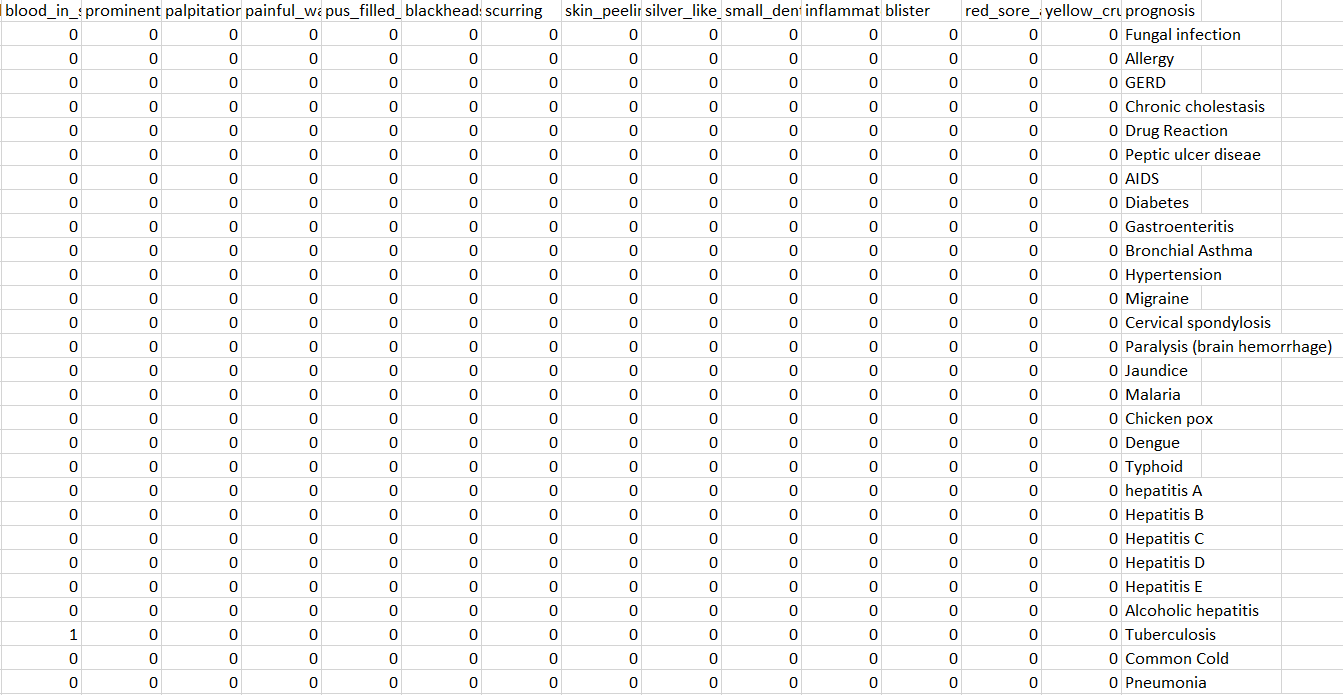
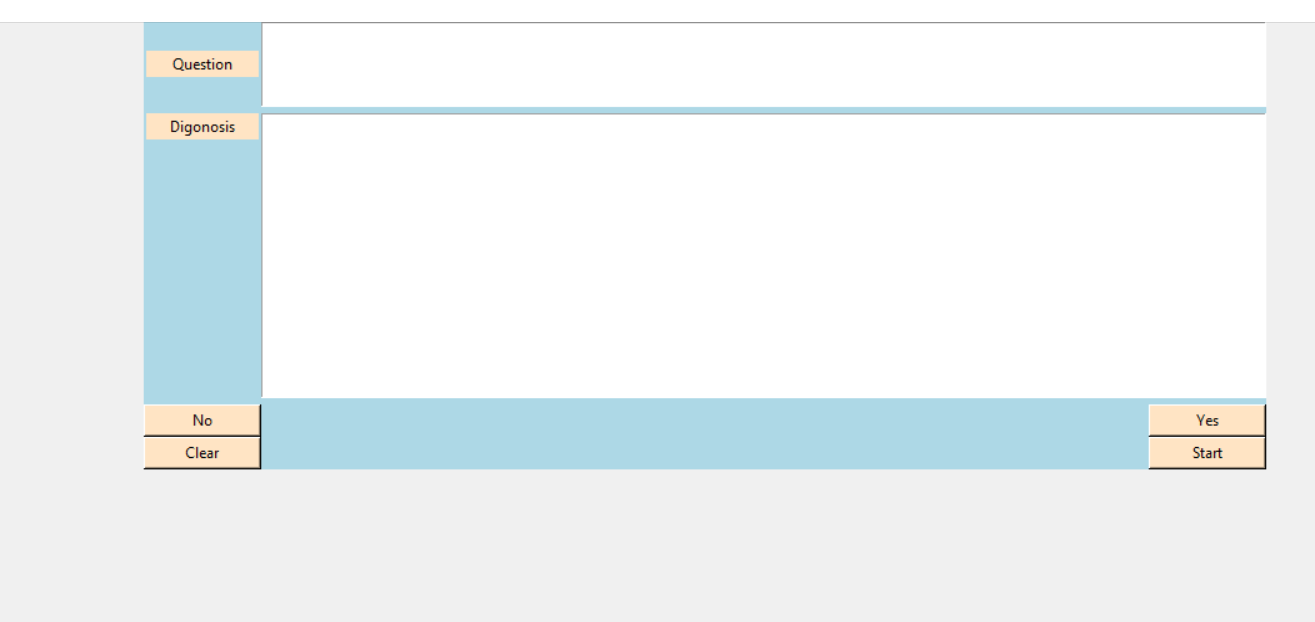


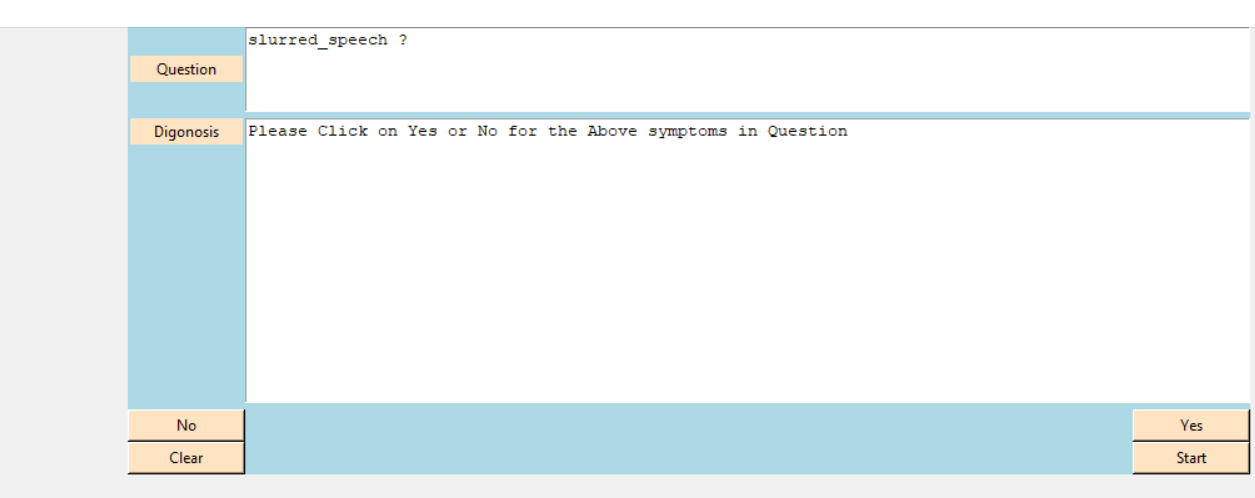
fig 3.2:- DATASET FOR DISEASE PREDICTION

**CHAPTER 4**

**PROJECT SCREENSHOTS**



Press start to start digonosis



Click yes or no as per your symptoms



**CHAPTER 5**

**Code Implementation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ########ApragmaticApproachforDiagnosis############ | | | | |
| #Importingthelibraries | | | | |
| fromtkinterimport\* | | | | |
| fromtkinterimportmessagebox | | | | |
| importos | | | | |
| importwebbrowser | | | | |
| importnumpyasnp | | | | |
| importpandasaspd | | | | |
| classHyperlinkManager: | | | | |
| def init (self,text): | | | | |
| self.text=text | | | | |
| self.text.tag\_config("hyper",foreground="blue",underline=1) | | | | |
| self.text.tag\_bind("hyper","<Enter>",self.\_enter) | | | | |
| self.text.tag\_bind("hyper","<Leave>",self.\_leave) | | | | |
| self.text.tag\_bind("hyper","<Button-1>",self.\_click) | | | | |
| self.reset() | | | | |
| defreset(self): | | | | |
| self.links={} | | | | |
| defadd(self,action): | | | | |
| #addanactiontothemanager.returnstagstousein | | | | |
| #associatedtextwidget | | | | |
| tag="hyper-%d"%len(self.links) | | | | |
| self.links[tag]=action | | | | |
| return"hyper",tag | | | | |
| def\_enter(self,event): | | | | |
| self.text.config(cursor="hand2") | | | | |
| def\_leave(self,event): | | | | |
| self.text.config(cursor="") | | | | |
| def\_click(self,event): | | | | |
| fortaginself.text.tag\_names(CURRENT): | | | | |
| iftag[:6]=="hyper-": | | | | |
| self.links[tag]() | | | | |
| return | | | | |
| #Importingthedataset | | | | |
| training\_dataset=pd.read\_csv('Training.csv') | | | | |
| test\_dataset=pd.read\_csv('Testing.csv') | | | | |
| #SlicingandDicingthedatasettoseparatefeaturesfrompredictions | | | | |
| X=training\_dataset.iloc[:,0:132].values | | | | |
| Y=training\_dataset.iloc[:,-1].values | | | | |
| #DimensionalityReductionforremovingredundancies | | | | |
| dimensionality\_reduction=training\_dataset.groupby(training\_dataset['prognosis']).max() | | | | |
| #EncodingStringvaluestointegerconstants | | | | |
| fromsklearn.preprocessingimportLabelEncoder | | | | |
| labelencoder=LabelEncoder() | | | | |
| y=labelencoder.fit\_transform(Y) | | | | |
| #Splittingthedatasetintotrainingsetandtestset | | | | |
| fromsklearn.model\_selectionimporttrain\_test\_split | | | | |
| X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state= 0) | | | | |
| #ImplementingtheDecisionTreeClassifier | | | | |
| fromsklearn.treeimportDecisionTreeClassifier | | | | |
| classifier=DecisionTreeClassifier() | | | | |
| classifier.fit(X\_train,y\_train) | | | | |
| #Savingtheinformationofcolumns | | | | |
| cols=training\_dataset.columns | | | | |
| cols=cols[:-1] | | | | |
| importances=classifier.feature\_importances\_ | | | | |
| indices=np.argsort(importances)[::-1] | | | | |
| features=cols | | | | |
| #ImplementingtheVisualTree | | | | |
| fromsklearn.treeimport\_tree | | | | |
| #MethodtosimulatetheworkingofaChatbotbyextractingandformulatingquestions | | | | |
| defprint\_disease(node): | | | | |
| #print(node) | | | | |
| node=node[0] | | | | |
| #print(len(node)) | | | | |
|  | | val | =node.nonzero() |  |
| #print(val) | | | | |
| disease=labelencoder.inverse\_transform(val[0]) | | | | |
| returndisease | | | | |
| defrecurse(node,depth): | | | | |
| globalval,ans | | | | |
| globaltree\_,feature\_name,symptoms\_present | | | | |
| indent=""\*depth | | | | |
| iftree\_.feature[node]!=\_tree.TREE\_UNDEFINED: | | | | |
| name=feature\_name[node] | | | | |
| threshold=tree\_.threshold[node] | | | | |
| yieldname+"?" | | | | |
| # | |  | ans=input() |  |
| ans=ans.lower() | | | | |
| ifans=='yes': | | | | |
| val=1 | | | | |
| else: | | | | |
| val=0 | | | | |
| ifval<=threshold: | | | | |
| yieldfromrecurse(tree\_.children\_left[node],depth+1) | | | | |
| else: | | | | |
| symptoms\_present.append(name) | | | | |
| yieldfromrecurse(tree\_.children\_right[node],depth+1) | | | | |
| else: | | | | |
| strData="" | | | | |
| present\_disease=print\_disease(tree\_.value[node]) | | | | |
| # | |  | print("Youmayhave"+ | present\_disease) |
| # | |  | print() |  |
| strData="Youmayhave:"+str(present\_disease) | | | | |
| QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n') | | | | |
| red\_cols=dimensionality\_reduction.columns | | | | |
| symptoms\_given=  red\_cols[dimensionality\_reduction.loc[present\_disease].values[0].nonzero()] | | | | |
| # | |  | print("symptomspresent | "+str(list(symptoms\_present))) |
| # | |  | print() |  |
| strData="symptomspresent:"+str(list(symptoms\_present)) | | | | |
| QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n') | | | | |
| # | |  | print("symptomsgiven" | +str(list(symptoms\_given))) |
| strData="symptomsgiven:"+str(list(symptoms\_given)) | | | | |
| QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n') | | | | |
| confidence\_level=(1.0\*len(symptoms\_present))/len(symptoms\_given) | | | | |
| # | print("confidencelevelis"+str(confidence\_level)) | | | |
| # | print() | | | |
| strData="confidencelevelis:"+str(confidence\_level) | | | | |
| QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n') | | | | |
| # | print('Themodelsuggests:') | | | |
| # | print() | | | |
| strData='Themodelsuggests:' | | | | |
| QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n') | | | | |
| row=doctors[doctors['disease']==present\_disease[0]] | | | | |
| # | print('Consult',str(row['name'].values)) | | | |
| # | print() | | | |
| strData='Consult'+str(row['name'].values) | | | | |
| QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n') | | | | |
| # | print('Visit',str(row['link'].values)) | | | |
| #print(present\_disease[0]) | | | | |
| hyperlink=HyperlinkManager(QuestionDigonosis.objRef.txtDigonosis) | | | | |
| strData='Visit'+str(row['link'].values[0]) | | | | |
| defclick1(): | | | | |
| webbrowser.open\_new(str(row['link'].values[0])) | | | | |
| QuestionDigonosis.objRef.txtDigonosis.insert(INSERT,strData,  hyperlink.add(click1)) | | | | |
| #QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n') | | | | |
| yieldstrData | | | | |
| deftree\_to\_code(tree,feature\_names): | | | | |
| globaltree\_,feature\_name,symptoms\_present | | | | |
| tree\_=tree.tree\_ | | | | |
| #print(tree\_) | | | | |
| feature\_name=[ | | | | |
| feature\_names[i]ifi!=\_tree.TREE\_UNDEFINEDelse"undefined!" | | | | |
| foriintree\_.feature | | | | |
| ] | | | | |
| #print("deftree({}):".format(",".join(feature\_names))) | | | | |
| symptoms\_present=[] | | | | |
| # | recurse(0,1) | | | |
| defexecute\_bot(): | | | | |
| # | print("Pleasereplywithyes/Yesorno/Noforthefollowingsymptoms") | | | |
| tree\_to\_code(classifier,cols) | | | | |
| #Thissectionofcodetoberunafterscrapingthedata | | | | |
| doc\_dataset=pd.read\_csv('doctors\_dataset.csv',names=['Name','Description']) | | | | |
| diseases=dimensionality\_reduction.index | | | | |
| doctors=pd.DataFrame() | | | | |
| doctors['name']=np.nan | | | | |
| doctors['link']=np.nan | | | | |
| doctors['disease']=np.nan | | | | |
| doctors['disease']=diseases['prognosis'] | | | | |
| doctors['name']=doc\_dataset['Name'] | | | | |
| doctors['link']=doc\_dataset['Description'] | | | | |
| record=doctors[doctors['disease']=='AIDS'] | | | | |
| record['name'] | | | | |
| record['link'] | | | | |
| #ExecutethebotandseeitinAction | | | | |
| #execute\_bot() | | | | |
| classQuestionDigonosis(Frame): | | | | |
| objIter=None | | | | |
| objRef=None | | | | |
| def init (self,master=None): | | | | |
| master.title("Question") | | | | |
| #root.iconbitmap("") | | | | |
| master.state("z") | | | | |
| #master.minsize(700,350) | | | | |
| QuestionDigonosis.objRef=self | | | | |
| super(). init (master=master) | | | | |
| self["bg"]="lightblue" | | | | |
| self.createWidget() | | | | |
| self.iterObj=None | | | | |
| defcreateWidget(self): | | | | |
| self.lblQuestion=Label(self,text="Question",width=12,bg="bisque") | | | | |
| self.lblQuestion.grid(row=0,column=0,rowspan=4) | | | | |
| self.lblDigonosis=Label(self,text="Digonosis",width=12,bg="bisque") | | | | |
| self.lblDigonosis.grid(row=4,column=0,sticky="n",pady=5) | | | | |
| #self.varQuestion=StringVar() | | | | |
| self.txtQuestion=Text(self,width=100,height=4) | | | | |
| self.txtQuestion.grid(row=0,column=1,rowspan=4,columnspan=20) | | | | |
| self.varDiagonosis=StringVar() | | | | |
| self.txtDigonosis=Text(self,width=100,height=14) | | | | |
| self.txtDigonosis.grid(row=4,column=1,columnspan=20,rowspan=20,pady=5) | | | | |
| self.btnNo.grid(row=25,column=0) | | | | |
| self.btnYes=Button(self,text="Yes",width=12,bg="bisque",  command=self.btnYes\_Click) | | | | |
| self.btnYes.grid(row=25,column=1,columnspan=20,sticky="e") | | | | |
| self.btnClear = Button(self, text="Clear",width=12,bg="bisque", command=self.btnClear\_Click) | | | | |
| self.btnClear.grid(row=27,column=0) | | | | |
| self.btnStart=Button(self,text="Start",width=12,bg="bisque",  command=self.btnStart\_Click) | | | | |
| self.btnStart.grid(row=27,column=1,columnspan=20,sticky="e") | | | | |
| defbtnNo\_Click(self): | | | | |
| globalval,ans | | | | |
| globalval,ans | | | | |
| ans='no' | | | | |
| str1=QuestionDigonosis.objIter. next () | | | | |
| self.txtQuestion.delete(0.0,END) | | | | |
| self.txtQuestion.insert(END,str1+"\n") | | | | |
| defbtnYes\_Click(self): | | | | |
| globalval,ans | | | | |
| ans='yes' | | | | |
| self.txtDigonosis.delete(0.0,END) | | | | |
| str1=QuestionDigonosis.objIter. next () | | | | |
| #self.txtDigonosis.insert(END,str1+"\n") | | | | |
| defbtnClear\_Click(self): | | | | |
| self.txtDigonosis.delete(0.0,END) | | | | |
| self.txtQuestion.delete(0.0,END) | | | | |
| defbtnStart\_Click(self): | | | | |
| execute\_bot() | | | | |
| self.txtDigonosis.delete(0.0,END) | | | | |
| self.txtQuestion.delete(0.0,END) | | | | |
| self.txtDigonosis.insert(END,"PleaseClickonYesorNofortheAbovesymptomsin  Question") | | | | |
| QuestionDigonosis.objIter=recurse(0,1) | | | | |
| str1=QuestionDigonosis.objIter. next () | | | | |
| self.txtQuestion.insert(END,str1+"\n") | | | | |
| classMainForm(Frame): | | | | |
| main\_Root=None | | | | |
| defdestroyPackWidget(self,parent): | | | | |
| foreinparent.pack\_slaves(): | | | | |
| e.destroy() | | | | |
| def init (self,master=None): | | | | |
| MainForm.main\_Root=master | | | | |
| super(). init (master=master) | | | | |
| master.geometry("300x250") | | | | |
| master.title("AccountLogin") | | | | |
| self.createWidget() | | | | |
| self.lblMsg=Label(self,text="SelectYourChoice",bg="blue",width="300",  height="2",font=("Calibri",13)) | | | | |
| self.lblMsg.pack() | | | | |
| self.btnLogin=Button(self,text="Login",height="2",width="30",  command=self.lblLogin\_Click) | | | | |
| self.btnLogin.pack() | | | | |
| self.btnRegister=Button(self,text="Register",height="2",width="30",  command=self.btnRegister\_Click) | | | | |
| self.btnRegister.pack() | | | | |
| deflblLogin\_Click(self): | | | | |
| self.destroyPackWidget(MainForm.main\_Root) | | | | |
| frmLogin=Login(MainForm.main\_Root) | | | | |
| frmLogin.pack() | | | | |
| defbtnRegister\_Click(self): | | | | |
| self.destroyPackWidget(MainForm.main\_Root) | | | | |
| frmSignUp=SignUp(MainForm.main\_Root) | | | | |
| frmSignUp.pack() | | | | |
| classLogin(Frame): | | | | |
| main\_Root=None | | | | |
| defdestroyPackWidget(self,parent): | | | | |
| foreinparent.pack\_slaves(): | | | | |
| e.destroy() | | | | |
| def init (self,master=None): | | | | |
| Login.main\_Root=master | | | | |
| super(). init (master=master) | | | | |
| master.title("Login") | | | | |
| master.geometry("300x250") | | | | |
| self.createWidget() | | | | |
| defcreateWidget(self): | | | | |
| self.lblMsg=Label(self,text="Pleaseenterdetailsbelowtologin",bg="blue") | | | | |
| self.lblMsg.pack() | | | | |
| self.username=Label(self,text="Username\*") | | | | |
| self.username.pack() | | | | |
| self.username\_verify=StringVar() | | | | |
| self.username\_login\_entry=Entry(self,textvariable=self.username\_verify) | | | | |
| self.username\_login\_entry.pack() | | | | |
| self.password=Label(self,text="Password\*") | | | | |
| self.password.pack() | | | | |
| self.password\_verify=StringVar() | | | | |
| self.password\_login\_entry=Entry(self,textvariable=self.password\_verify,show='\*') | | | | |
| self.password\_login\_entry.pack() | | | | |
| self.btnLogin=Button(self,text="Login",width=10,height=1,  command=self.btnLogin\_Click) | | | | |
| self.btnLogin.pack() | | | | |
| defbtnLogin\_Click(self): | | | | |
| username1=self.username\_login\_entry.get() | | | | |
| password1=self.password\_login\_entry.get() | | | | |
| #messagebox.showinfo("Failure",self.username1+":"+password1) | | | | |
| list\_of\_files=os.listdir() | | | | |
| ifusername1inlist\_of\_files: | | | | |
| file1=open(username1,"r") | | | | |
| verify=file1.read().splitlines() | | | | |
| messagebox.showinfo("Sucess","LoginSucessful") | | | | |
| self.destroyPackWidget(Login.main\_Root) | | | | |
| frmQuestion=QuestionDigonosis(Login.main\_Root) | | | | |
| frmQuestion.pack() | | | | |
| else: | | | | |
| messagebox.showinfo("Failure","LoginDetailsarewrongtryagain") | | | | |
| else: | | | | |
| messagebox.showinfo("Failure","Usernotfoundtryfromanotheruser\norsign  upfornewuser") | | | | |
| classSignUp(Frame): | | | | |
| main\_Root=None | | | | |
| defdestroyPackWidget(self,parent): | | | | |
| foreinparent.pack\_slaves(): | | | | |
| e.destroy() | | | | |
| def init (self,master=None): | | | | |
| SignUp.main\_Root=master | | | | |
| master.title("Register") | | | | |
| super(). init (master=master) | | | | |
| master.title("Register") | | | | |
| master.geometry("300x250") | | | | |
| self.createWidget() | | | | |
| defcreateWidget(self): | | | | |
| self.lblMsg=Label(self,text="Pleaseenterdetailsbelow",bg="blue") | | | | |
| self.lblMsg.pack() | | | | |
| self.username\_lable=Label(self,text="Username\*") | | | | |
| self.username\_lable.pack() | | | | |
| self.username=StringVar() | | | | |
| self.username\_entry=Entry(self,textvariable=self.username) | | | | |
| self.username\_entry.pack() | | | | |
| self.password\_lable=Label(self,text="Password\*") | | | | |
| self.password\_lable.pack() | | | | |
| self.password=StringVar() | | | | |
| self.password\_entry=Entry(self,textvariable=self.password,show='\*') | | | | |
| self.password\_entry.pack() | | | | |
| self.btnRegister=Button(self,text="Register",width=10,height=1,bg="blue",  command=self.register\_user) | | | | |
| self.btnRegister.pack() | | | | |
| defregister\_user(self): | | | | |
| #print(self.username.get()) | | | | |
| #print("Hello") | | | | |
| file=open(self.username\_entry.get(),"w") | | | | |
| file.write(self.username\_entry.get()+"\n") | | | | |
| file.write(self.password\_entry.get()) | | | | |
| file.close() | | | | |
| self.destroyPackWidget(SignUp.main\_Root) | | | | |
| self.lblSucess=Label(root,text="RegistrationSuccess",fg="green",font=("calibri",  11)) | | | | |
| self.lblSucess.pack() | | | | |
| self.btnSucess.pack() | | | | |
| defbtnSucess\_Click(self): | | | | |
| self.destroyPackWidget(SignUp.main\_Root) | | | | |
| frmQuestion=QuestionDigonosis(SignUp.main\_Root) | | | | |
| frmQuestion.pack() | | | | |
| root=Tk() | | | | |
| frmMainForm=MainForm(root) | | | | |
| frmMainForm.pack() | | | | |
| root.mainloop() | | | | |

**CHAPTER 5**

**APPLICATIONS OF SENTIMENT ANALYSIS**

# 1.Availability and ongoing health monitoring:-

All healthcare providers are always willing to help their patients and they understand how it is vital to be available if there is urgent need of medical attention. Unfortunately, doctors have limited time and a lot of patients which doesn’t allow them to be available anytime.

# 2.Providing information fast when there is not a moment to lose:-

Emergencies are normal for healthcare which means prompt and correct diagnosis is vitally important. Various healthcare specialists can make the right decision in time in case they are able to obtain the needed patient information easily. That’s why a number of modern medical institutions are creating a chatbot which is responsible for internal record-keeping. This kind of bots is able to immediately receive important patient information such as prescribed and administered medication, allergies, check-up reports, etc.

# 3. Gaining the trust of patients:-

All medical institutions keeping up with advancing technology should be effectively presented online to attract potential patients and maintain their appointments booked. However, the thing is that it might not be easy for patients to look through lots of webpages on health problems and book appointments with no help.

# 4.Scheduling appointments:-

As we just have read in the previous item, virtual assistants for healthcare websites is a great tool to begin communicating with future possible patients. These bots are effective in particular if it comes to booking visits. People are able to schedule an appointment with a medical specialist online almost instantly.

# 5. Providing support and additional information:-

As chatbots in healthcare are highly in demand, medical institutions can offer various services from symptom checking and appointment scheduling to dealing with additional questions. People are able to get answers to their additional questions with the help of chatbot. There is no need for them to call the clinic to clarify some misunderstanding. There are chatbots which can provide information to the following and similar to them questions.“How long is someone infectious after a viral infection?”“How can I get a prescription?”“How can I find out my blood type (blood group)?”

Thereby, clinics building a chatbot for their sites, lower the number of repetitive calls that their specialists have to answer. This, in its turn, enables hospital employees to concentrate on more significant tasks which will lead to better healthcare service quality.

**CHAPTER 6**

**CONCLUSION**

It bid a Machine learning Decision tree map algorithm by using structured and unstructured data from hospital. It also uses Map Reduce algorithm for partitioning the data. To the highest of gen, none of the current work attentive on together data types in the zone of remedial big data analytics. Compared to several typical calculating algorithms, the scheming accuracy of our proposed algorithm reaches 94.8% with an regular speed which is quicker than that of the CNN-based unimodal disease risk prediction (CNN-UDRP) algorithm and produces report. The report consists of possibility of occurrences of diseases.

## 

## CHAPTER 7

## REFERENCES

1. [www.google.com](http://www.google.com/)
2. https:/[/www.pra](http://www.practo.com/)c[to.com/](http://www.practo.com/)
3. https://courses.edx.org/certific ates/fa4fda27bb734dafab5fcdc ca8b36032