SIES (NERUL) COLLEGE OF ARTS, SCIENCE AND COMMERCE

NAAC ACCREDITED 'A' GRADE COLLEGE (ISO 9001:2008 CERTIFIED INSTITUTION) NERUL, NAVI MUMBAI – 400706

PROJECT REPORT ON

Disease Diagnosis Using Chatbot

SUBMITTED BY

Bosco Trevor Dsouza

PROJECT GUIDE

Asst. Professor Manasvi Sharma

SUBMITTED IN PARTIAL FULFILLMENT FOR THE DEGREE OF

MSc. (COMPUTER SCIENCE)

SEMESTER – IV, 2021 - 2022

SIES (NERUL) COLLEGE OF ARTS SCIENCE AND COMMERECE

NAAC ACCERDITED 'A' GRADE COLLEGE

(ISO 9001:2015 CERTIFIED INSTITUTION)

NERUL, NAVI MUMBAI - 400706

Certificate

THIS IS TO CERTIFY THAT THE PROJECT TITLED

DISEASE DIAGNOSIS USING CHATBOT

IS UNDERTAKEN BY

BOSCO TREVOR DSOUZA

Seat No: 04

In partial fulfilment of MSc - IT / CS Degree (Semester _ **IV**)Examination in the academic year **2021-2022** and has not been submitted for any other examination and does not form part of any other course undergone by the candidate. It is further certified that he/she has completed all the required phases of the project.

Project Guide External Examiner

Head of Department Principal

ACKNOWLEDGMENT

I extend my heartfelt gratitude and thanks to Asst. Professor Manasvi Sharma sir for providing me excellent guidance to work on this project and for their understanding and assistance by providing all the necessary information needed for my project topic.

I would also like to acknowledge all the staffs for providing a helping hand to us in times of queries & problems. The project is a result of the efforts of all the peoples who are associated with the project directly or indirectly, who helped us to work to complete the project within the specified time frame. They motivated me in the project and gave a feedback on it to improve my adroitness.
Thanks to all my teachers, who were a part of the project in numerous ways and for the help and inspiration they extended to me and for providing the needed motivation.
With all Respects & Gratitude, I would like to thanks to all the people, who have helped for the development of the Project.

Bosco Trevor Dsouza

MSc. Computer Science (Part-II)

SIES (Nerul) College of Arts, Science, and Commerce.

Table of Content

PROJECT REPORT ON:

Disease Diagnosis Using Chatbot

INTRODUCTION:

A Chatbot is one of the most interesting and a trending A.I. model which is growing in popularity and has been implemented in almost every sector. From Education to Health Care every sector is using Chatbot. With my research in chatbot and disease prediction I had a unique idea of combining both the system into one and make a Heath Care Chatbot system which will help in the diagnosis of the disease based on the symptoms of the patient.

There are various machine learning algorithms and deep learning algorithms that I have used to create the system. I have used simple neural networks for the chatbot and I have used three machine learning algorithms Support Vector Machine, Decision Tree and Random Forest for predicting or diagnosis of the disease that a patient has based on the symptoms he/she has.

I have tried to provide options to the user so that the time of the user is saved and also with such feature it would be easier to get the exact symptoms of the user and the model can give you more accurate results.

My model can be very useful as it uses three models to predict the disease. i.e., it will predict three possible diseases that a user may have.

So, the entire chatbot has been designed using 4 algorithms and 2 datasets. The entire system is built with an idea that it serves and give accurate diagnosis to the patient so that he can get proper diagnosis for the disease.

IMPLEMENTATION DETAIL

Data Collection:

For This Project I required two Datasets. One dataset is in JSON format and the second dataset is in CSV format. The dataset in CSV format contains disease and the possible symptoms which can cause the disease. There are almost 2170 records in the Csv file. The second file I have created in guidance of Dr. Mukesh Bhatija. This json file consist of what are the patterns or what type of question a patient may ask if he or she may have some sort of disease. This file consists of patterns and responses. Not all the diseases can be predicted using the chatbot so I Dr. Mukesh guided me on what are the possible disease that my system might be able to predict.

System Requirement Specification:

1. Python
2. Flask
3. NumPy
4. Pandas
5. Matplotlib
6. Ajax
7. Seaborn
8. Keras
9. Tensorflow
10. Vs Code

Algorithms:

Neural Networks are very widely used. Neural networks are designed similarly like the human brain or we may say its functioning is exactly similar to the functioning of a human brain. Simple neural networks are one of the best suited algorithms that we can use if there is only one tensor input and we are expecting only one tensor output so in such a case the keras sequential api is the best option in which we can just define the dense layer and the number of neurons we need in that layer.

Random Forest Classifier The random forest algorithm is also known as the random forest classifier. The RF algorithm comprises a random collection or a random selection of a forest tree. It creates a random sample of multiple decision trees and merges them together to obtain a more stable and accurate prediction through cross validation. Here it creates various combinations of decision tree based on the symptoms and the one which give the best possible outcome is chosen as a classifier.

Decision Tree Classifier in decision tree there are root nodes and leaf nodes the root nodes are parent class and leaf nodes children class. The parent node split on basis of some condition and connect to the leaf nodes. the symptoms are split and a tree is formed there are many nodes that are connected to the tree and based on that the disease are been predicted or classified.

Support Vector Machine Classifier in this we can divide the values with the help of a hyperplane and the points are plotted and based on the points on the graph the values are classified. There are 18 diseases so there are 18 hyperplane and then the points are plotted and classified.

EXPERIMENTAL SET UP AND RESULTS DATASET

The json file consist of intents tags, patterns and responses.

Disease_Category_intenets_.json × app.py Disease_predict.py index.html

HealthCare > static > Disease_Category_intenets_.json

```
{"intents":[
    {"tag":"fever",
     "patterns":["I have fever"],
     "responses":[
            {
            "type":["radio"],
            "question": ["You said you have fever, What is your body Temperature?"],
            "options": ["Above 100 degree F","Below 100 degree F"]
            }

                ]
    },
    {"tag":"chemical_infection",
     "patterns":["my skin has turned silver"],
     "responses":[
            {
            "type":["radio"],
            "question": ["You said yur skin has turned silver, are you or were you exposed to any chemicals? "],
            "options": ["Yes I was","No I was not"]
                    }
                ]
    },
    {"tag":"loss_of_balance",
     "patterns":["I often loose my balance"],
     "responses":[
            {
            "type":["radio"],
            "question": ["You said you often loose your balance, Is it regularly or sometimes?"],
            "options": ["Sometimes","Regularly"]
                    }
                ]
    },
    {"tag":"cold_and_cough"
```

```
[2] df.shape
```

```
(2169, 90)
```

The disease and symptoms dataset consist of 2169 rows and 90 columns.

Data Cleaning and Preprocessing:

The data is already clean and only requires one hot encoding format. The list of disease that we are predicting is given below. The dataset is in one hot encoding format so it has to be processed and converted into that format.

```
disease=['Fungal infection','Allergy','Drug Reaction',
'Peptic ulcer diseae',
' Migraine',
'Paralysis (brain hemorrhage)','Jaundice','Chicken pox',
'Common Cold','Dimorphic hemmorhoids(piles)',
'Varicoseveins','Hypothyroidism',
'Arthritis','(vertigo) Paroymsal  Positional Vertigo','Acne','Urinary tract infection','Psoriasis',
'Impetigo']
```

```
for k in range(0,len(l1)):
        for z in psymptoms:
            if(z==l1[k]):
                l2[k]=1
```

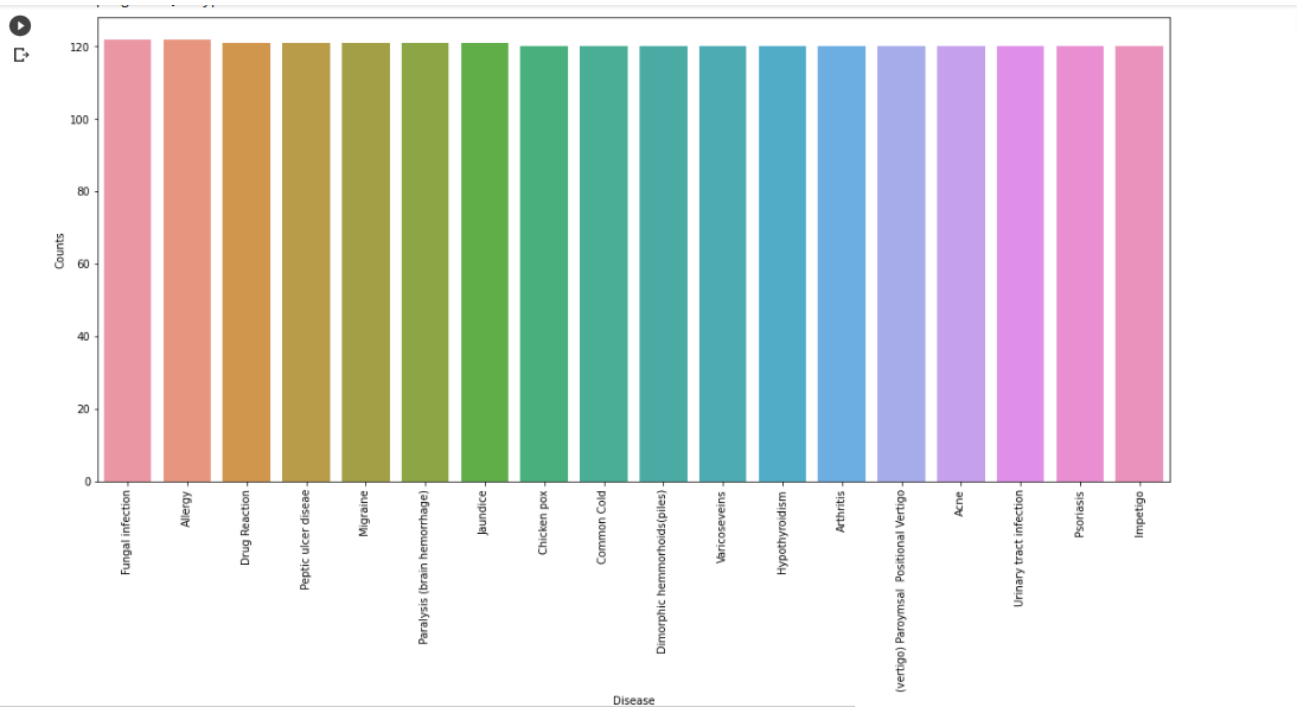
For chatbot we require to do plenty of cleaning we have used nltk that is natural Language tool kit as we have to use NLP natural language processing to create the chatbot.

```
for intent in intents['intents']:
    for pattern in intent['patterns']:
        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        print(words)
        #add documents in the corpus
        documents.append((w, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
print(documents)
```

We need to tokenize the words that is treat each word as a separate entity as computer does not understand sentences. The next step is to lemmatize the word and change it into lower case. Lemmatizing means reduction of words which have no importance, remove punctuations and remove duplicate values. We transform the data in the lowercase format so that there is a common format. We create a bag of words for lemmatizing the words.

```
# lemmatize, lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
type(classes)
classes = sorted(list(set(classes)))
print (len(documents), "documents")
print (len(classes), "classes", classes)
print (len(words), "unique lemmatized words", words)
pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
```

The disease and the counts of records for each disease are similar it can be shown with the help of a bar plot.



Here we can clearly see all the records are of length 120 and we can see there are 18 disease we are classifying or training as these are the only diseases that can be categorized by a chatbot.

ANALYSIS OF RESULT

Random Forest:

- Classification Report: -
  Here the classification report for test data and predictions

```
print(classification_report(y_test, y_pred2))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        17
           1       0.93      1.00      0.97        14
           2       1.00      1.00      1.00        13
           3       1.00      0.93      0.96        14
           4       1.00      1.00      1.00        13
           5       0.92      0.92      0.92        13
           6       1.00      0.89      0.94        19
           7       0.86      0.92      0.89        13
           8       0.94      1.00      0.97        15
           9       1.00      1.00      1.00        14
          10       1.00      1.00      1.00        13
          11       1.00      1.00      1.00        13
          12       1.00      1.00      1.00        15
          13       1.00      1.00      1.00        13
          14       1.00      1.00      1.00        12
          15       1.00      1.00      1.00        12
          16       0.93      1.00      0.97        14
          17       0.93      0.93      0.93        15

    accuracy                           0.97       252
   macro avg       0.97      0.97      0.97       252
weighted avg       0.97      0.97      0.97       252
```

Here we have 1.00 precision of 0's (Fungal Infection) with 0.94 of recall it means we have 100% of 0 actual data out of that algorithm is able to predict 94% of data. Similarly, we have 0.92 precision of 5's with 0.92 of recall it means we have 92% of actual data out of that algorithm is able to predict 92% of data.

- Accuracy: -

```
print("Accuracy",accuracy_score(y_test, y_pred2))
```

```
Accuracy 0.9722222222222222
```

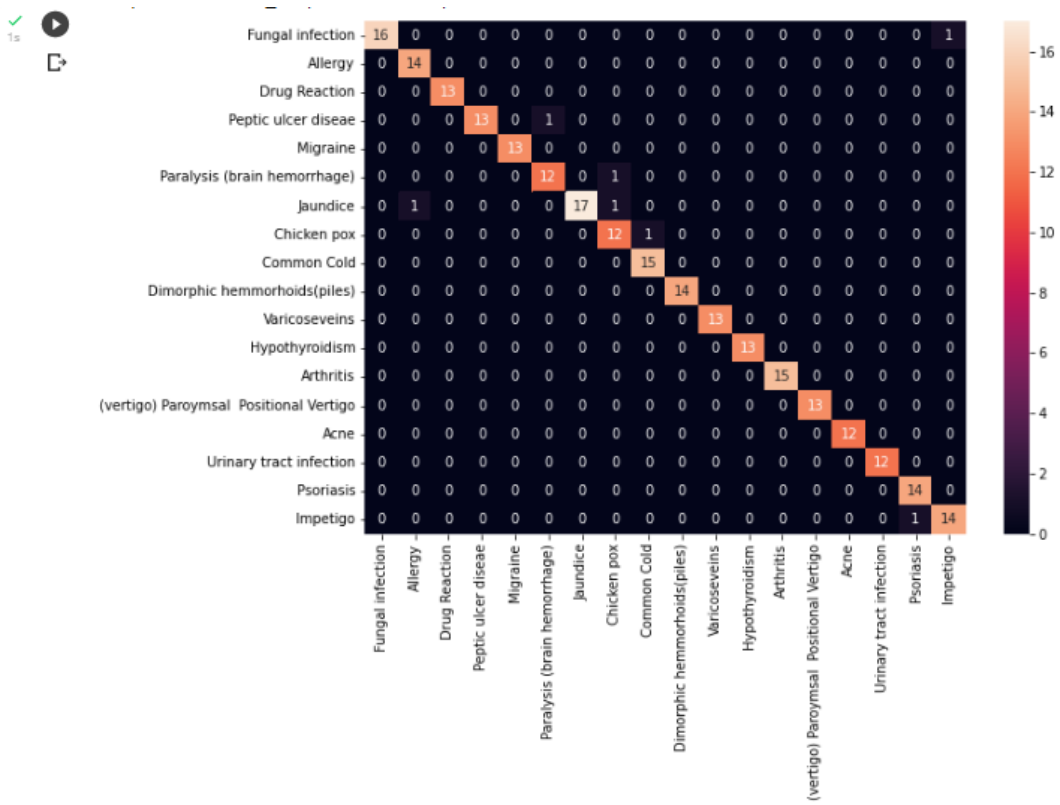
The accuracy of this model is 0.9722 which is 97.22%.

➢ Confusion Matrix: -

```
[48] conf_mat = confusion_matrix(y_test, y_pred2)
     print(conf_mat)
```

```
[[16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1]
 [ 0 14  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 13  0  1  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 13  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 12  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  1  0  0  0  0 17  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 12  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 15  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 14  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 15  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 13  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 14  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 14]]
```

The confusion matrix in a heatmap format can be represented as:



Decision Tree Classifier:

- Classification Report: -
  Here the classification report for test data and predictions

```
print(classification_report(y_test, y_pred3))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        17
           1       0.93      1.00      0.97        14
           2       1.00      0.92      0.96        13
           3       1.00      0.93      0.96        14
           4       0.87      1.00      0.93        13
           5       0.92      0.92      0.92        13
           6       0.94      0.89      0.92        19
           7       0.92      0.92      0.92        13
           8       0.94      1.00      0.97        15
           9       1.00      1.00      1.00        14
          10       1.00      1.00      1.00        13
          11       1.00      1.00      1.00        13
          12       1.00      1.00      1.00        15
          13       0.92      0.92      0.92        13
          14       1.00      1.00      1.00        12
          15       1.00      0.92      0.96        12
          16       0.93      1.00      0.97        14
          17       0.93      0.93      0.93        15

    accuracy                           0.96       252
   macro avg       0.96      0.96      0.96       252
weighted avg       0.96      0.96      0.96       252
```

Here we have 1.00 precision of 0's (Fungal Infection) with 0.94 of recall it means we have 100% of 0 actual data out of that algorithm is able to predict 94% of data. Similarly, we have 0.87 precision of 4's with 1.00 of recall it means we have 87% of actual data out of that algorithm is able to predict 100% of data.

- Accuracy: -

```
print("Accuracy",accuracy_score(y_test, y_pred3))
```

```
Accuracy 0.9603174603174603
```

The accuracy of this model is 0.9603 which is 96.03%.

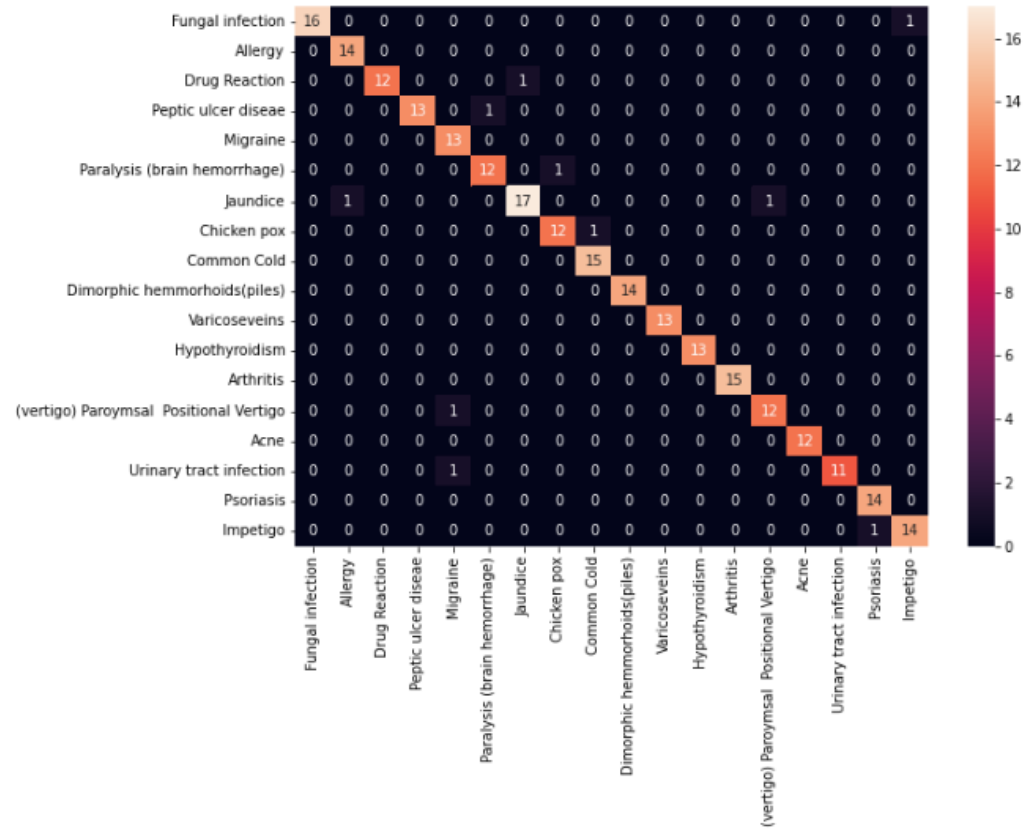
➢ Confusion Matrix: -

```
conf_mat = confusion_matrix(y_test, y_pred3)
print(conf_mat)
```

```
[[16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1]
 [ 0 14  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 12  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 13  0  1  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 13  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 12  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  1  0  0  0  0 17  0  0  0  0  0  0  1  0  0  0  0]
 [ 0  0  0  0  0  0  0 12  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 15  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 14  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 15  0  0  0  0  0]
 [ 0  0  0  0  1  0  0  0  0  0  0  0  0 12  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0]
 [ 0  0  0  0  1  0  0  0  0  0  0  0  0  0  0 11  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 14  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 14]]
```

The confusion matrix in a heatmap format can be represented as:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3894970310>
```



Support Vector Machine Classifier:

- Classification Report: -
  Here the classification report for test data and predictions

```
print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        17
           1       0.93      1.00      0.97        14
           2       1.00      1.00      1.00        13
           3       1.00      0.93      0.96        14
           4       1.00      1.00      1.00        13
           5       0.86      0.92      0.89        13
           6       1.00      0.89      0.94        19
           7       0.86      0.92      0.89        13
           8       0.94      1.00      0.97        15
           9       1.00      1.00      1.00        14
          10       1.00      1.00      1.00        13
          11       1.00      1.00      1.00        13
          12       1.00      1.00      1.00        15
          13       1.00      0.92      0.96        13
          14       1.00      1.00      1.00        12
          15       1.00      1.00      1.00        12
          16       0.93      1.00      0.97        14
          17       0.93      0.93      0.93        15

    accuracy                           0.97       252
   macro avg       0.97      0.97      0.97       252
weighted avg       0.97      0.97      0.97       252
```

- Accuracy: -

```
print("Accuracy",accuracy_score(y_test, y_pred1))
```

```
Accuracy 0.9682539682539683
```

The accuracy of this model is 0.9682 which is 96.82% which is almost equivalent to 97%

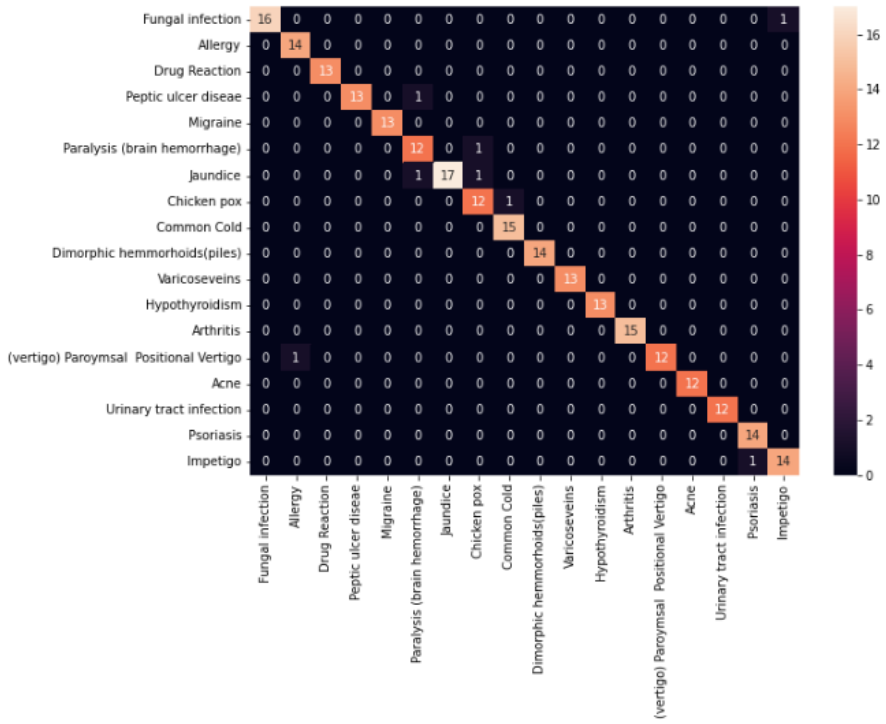
➢ Confusion Matrix: -

```
[57] conf_mat = confusion_matrix(y_test, y_pred1)
     print(conf_mat)
```

```
[[16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1]
 [ 0 14  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 13  0  1  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 13  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 12  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  1 17  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 12  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 15  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 14  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 15  0  0  0  0  0]
 [ 0  1  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 14  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 14]]
```

The confusion matrix in a heatmap format can be represented as:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3894590450>
```



CONCLUSION:

- The Random Forest Classifier gives us maximum accuracy followed by Support Vector Machine and decision tree algorithm.
- This project is to give maximum result to the user and give them results as accurate as possible so that they can get proper treatment without consulting the doctor.
- This project has an easy-to-use interface which can be easily used by the patient.
- The questions are as simplified as possible so that the user can easily understand the options.

FUTURE ENHANCEMENT:

- The Enhancement that needs to be done is to classify the questions as radio button and checkbox
- The next is to improve the GUI.
- The disease can also be predicted based on more algorithms so try more algorithms in order to get more accurate results.
- Add some more feature like direct consulting the doctor and suggest some precautions and ayurvedic medicine.
- Also add the feature to suggest a diet plan to the user.

GUI:

MEDIBOT

Hi, How Are You?

Are You Sick?

Yes No

What are the symptoms?

- [ ] my skin has turned silver
- [ ] I often loose my balance
- [ ] I have cold and cough
- [x] I have thyroid or throat infection
- [x] I have eye related issues
- [x] I have fever
- [ ] I am experiencing pain and stifness
- [ ] I have got skin infection
- [ ] I feel weak and dizzi
- [ ] I am feeling itchy
- [x] I have urinary and anus infection
- [ ] I have issues while digestion
- [ ] I have mental and concenteration issues
- [ ] I have swelling
- [ ] I have weight related issues

I have thyroid or throat infection,I have eye related issues,I have fever,I have urinary and anus infection

send

- [x] My hands and feet are cold
- [ ] I have sticky and stringy mucus
- [ ] I have irritation in my throat
- [ ] I think my thyroid has enlarged

You said you have eye related issues, what exactly are your symtoms?

- [x] I have blurred and disoted vision
- [x] I have visual disturbances
- [ ] My eyes have turned red in color

You said you have fever, What is your body Temperature?

- [ ] Above 100 degree F
- [x] Below 100 degree F

You said you have urinary and anus infection, what exactly are your symtoms?

- [ ] My urine has a foul smell
- [ ] You feel like passing urine many times
- [ ] I have dark colored urine
- [ ] I have blood in urine
- [ ] I have discomfort and burning during urination
- [x] I have experienced inflammation near the bladder
- [ ] I have irritation near the anus area
- [ ] I have blood in my stools
- [ ] I am experiencing abnormal mensuration

You have a high possibility of having Urinary tract infection or you might also be suffering from Migraine

PROGRAM CODE:

App.py:

```
from msilib.schema import RadioButton
from operator import truediv
from pickle import TRUE
from tabnanny import check
import pandas as pd
from click import option
from flask import Flask, jsonify, render_template, request
import pandas as pd
from nltk.stem import WordNetLemmatizer
from keras.models import model_from_json
lemmatizer = WordNetLemmatizer()
import json
import nltk
import numpy as np
import pickle

app = Flask(__name__)

infile = open('words.pkl','rb')
words = pickle.load(infile)
infile2 = open('classes.pkl','rb')
classes = pickle.load(infile2)
data_file = open('static/Disease_Category_intenets_.json').read()
intents = json.loads(data_file)
def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in
sentence_words]
    return sentence_words
# return bag of words array: 0 or 1 for each word in the bag that exists in the
sentence
def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
```

```
                    print ("found in bag: %s" % w)
    return(np.array(bag))
def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list

def get_disease(symptom):
    """symptomp=symptom.split(",")
    lent=len(symptomp)
    for i in range(lent,17):
        symptom+=","""
    symptom=symptom.split(",")
    print(len(symptom))
    disease=[]
    for i in range(0,len(symptom)):
        if("Above 100 degree F" in symptom[i]):
            disease.append("high_fever")
        elif("Below 100 degree F" in symptom[i]):
            disease.append("mild_fever")
        elif("Yes I was" in symptom[i]):
            disease.append("silver_like_dusting")
        elif("Regularly" in symptom[i]):
            disease.append("loss_of_balance")
        elif("I get chills" in symptom[i]):
            disease.append("chills")
        elif("I experience continuous sneezing" in symptom[i]):
            disease.append("continuous_sneezing")
        elif("I cough a lot" in symptom[i]):
            disease.append("cough")
        elif("I have a running nose" in symptom[i]):
            disease.append("runny_nose")
        elif("I shiver a lot" in symptom[i]):
            disease.append("shivering")
        elif("I have lost my sense of smell" in symptom[i]):
            disease.append("loss_of_smell")
        elif("I have sinus" in symptom[i]):
            disease.append("sinus_pressure")
        elif("My eyes continuously water" in symptom[i]):
            disease.append("watering_from_eyes")
```

```
        elif("I have a blocked nose" in symptom[i]):
            disease.append("congestion")

        elif("My hands and feet are cold" in symptom[i]):
            disease.append("cold_hands_and_feets")
        elif("I have sticky and stringy mucus" in symptom[i]):
            disease.append("phlegm")
        elif("I have irritation in my throat" in symptom[i]):
            disease.append("throat_irritation")
        elif("I think my thyroid has enlarged" in symptom[i]):
            disease.append("enlarged_thyroid")

        elif("I have blurred and disoted vision" in symptom[i]):
            disease.append("blurred_and_distorted_vision")
        elif("I have visual disturbances" in symptom[i]):
            disease.append("visual_disturbances")
        elif("My eyes have turned red in color" in symptom[i]):
            disease.append("redness_of_eyes")

        elif("I am experiencing pain during bowel movements" in symptom[i]):
            disease.append("pain_during_bowel_movements")
        elif("I have pain in my joints" in symptom[i]):
            disease.append("joint_pain")
        elif("I have pain in my abdomen" in symptom[i]):
            disease.append("abdominal_pain")
        elif("I have pain in my stomach" in symptom[i]):
            disease.append("stomach_pain")
        elif("I have a muscle pain" in symptom[i]):
            disease.append("muscle_pain")
        elif("I have pain in my anal region" in symptom[i]):
            disease.append("pain_in_anal_region")
        elif("I have pain in my chest" in symptom[i]):
            disease.append("chest_pain")
        elif("I experience pain while walking" in symptom[i]):
            disease.append("painful_walking")
        elif("I am experiencing stifness in my movements" in symptom[i]):
            disease.append("movement_stiffness")
        elif("I have got a stiff neck" in symptom[i]):
            disease.append("stiff_neck")

        elif("I have a skin rash" in symptom[i]):
            disease.append("skin_rash")
        elif("I have skin erruptions" in symptom[i]):
            disease.append("nodal_skin_eruptions")
        elif("My skin is peeling" in symptom[i]):
            disease.append("skin_peeling")
        elif("My skin has turned to yellow" in symptom[i]):
            disease.append("yellowish_skin")
```

```
        elif("I have blackheads" in symptom[i]):
            disease.append("blackheads")
        elif("I have got pus filled pimples" in symptom[i]):
            disease.append("pus_filled_pimples")
        elif("I have patches on my body" in symptom[i]):
            disease.append("dischromic _patches")
        elif("I have got a red sore around my nose" in symptom[i]):
            disease.append("red_sore_around_nose")
        elif("I have yellow pus filled blisters" in symptom[i]):
            disease.append("yellow_crust_ooze")
        elif("I have inflamation around my nails" in symptom[i]):
            disease.append("inflammatory_nails")
        elif("I have red spots over the body" in symptom[i]):
            disease.append("red_spots_over_body")
        elif("I have blisters" in symptom[i]):
            disease.append("blister")
        elif("I have small dents in my nails" in symptom[i]):
            disease.append("small_dents_in_nails")
        elif("There are brusies on my leg" in symptom[i]):
            disease.append("bruising")

        elif("I am experiencing scurring" in symptom[i]):
            disease.append("scurring")
        elif("I feel dizzi" in symptom[i]):
            disease.append("dizziness")
        elif("I have weakness on one side of the body" in symptom[i]):
            disease.append("weakness_of_one_body_side")
        elif("I have muscle weakness" in symptom[i]):
            disease.append("muscle_weakness")
        elif("I am expereincing fatigue" in symptom[i]):
            disease.append("fatigue")
        elif("I am feeling lethargic" in symptom[i]):
            disease.append("lethargy")
        elif("I am exeriencing malaise" in symptom[i]):
            disease.append("malaise")
        elif("I get cramps" in symptom[i]):
            disease.append("cramps")

        elif("I have an external itch" in symptom[i]):
            disease.append("itching")
        elif("I have an internal itch" in symptom[i]):
            disease.append("internal_itching")

        elif("My urine has a foul smell" in symptom[i]):
            disease.append("foul_smell_of urine")
        elif("You feel like passing urine many times" in symptom[i]):
            disease.append("continuous_feel_of_urine")
        elif("I have dark colored urine" in symptom[i]):
```

```
            disease.append("dark_urine")
        elif("I have blood in urine" in symptom[i]):
            disease.append("spotting_ urination")
        elif("I have discomfort and burning during urination" in symptom[i]):
            disease.append("burning_micturition")
        elif("I have experienced inflammation near the bladder" in symptom[i]):
            disease.append("bladder_discomfort")
        elif("I have irritation near the anus area" in symptom[i]):
            disease.append("irritation_in_anus")
        elif("I have blood in my stools" in symptom[i]):
            disease.append("bloody_stool")
        elif("I am experiencing abnormal mensuration" in symptom[i]):
            disease.append("abnormal_menstruation")

        elif("I have acidity" in symptom[i]):
            disease.append("acidity")
        elif("I am experiencing vomiting" in symptom[i]):
            disease.append("vomiting")
        elif("I am experiencing nausea" in symptom[i]):
            disease.append("nausea")
        elif("I am experiencing indigestion" in symptom[i]):
            disease.append("indigestion")
        elif("I am having constipation" in symptom[i]):
            disease.append("constipation")
        elif("I fart a lot" in symptom[i]):
            disease.append("passage_of_gases")
        elif("I have lost my appetite" in symptom[i]):
            disease.append("loss_of_appetite")
        elif("I feel excessive hunger" in symptom[i]):
            disease.append("excessive_hunger")

        elif("I have a headache" in symptom[i]):
            disease.append("headache")
        elif("I experience confusion and loss of memory sort of altered
sensorium" in symptom[i]):
            disease.append("altered_sensorium")
        elif("I have mood swings" in symptom[i]):
            disease.append("mood_swings")
        elif("I have feeleing like I will faint" in symptom[i]):
            disease.append("spinning_movements")
        elif("I am having depression" in symptom[i]):
            disease.append("depression")
        elif("I feel irritated" in symptom[i]):
            disease.append("irritability")
        elif("I feel like unsteadiness" in symptom[i]):
            disease.append("unsteadiness")

        elif("I have swelling in my joints" in symptom[i]):
```

```
            disease.append("swelling_joints")
        elif("I have swollen lyph nodes" in symptom[i]):
            disease.append("swelled_lymph_nodes")
        elif("I have a puffy face and eyes" in symptom[i]):
            disease.append("puffy_face_and_eyes")
        elif("I have swollen hands and feets" in symptom[i]):
            disease.append("swollen_extremeties")
        elif("I have prominent veins on my calfs" in symptom[i]):
            disease.append("prominent_veins_on_calf")
        elif("My legs are swollen" in symptom[i]):
            disease.append("swollen_legs")
        elif("I have swollen blood vessels" in symptom[i]):
            disease.append("swollen_blood_vessels")

        elif("I have gained weight" in symptom[i]):
            disease.append("weight_gain")
        elif("I have lost weight" in symptom[i]):
            disease.append("weight_loss")
        elif("I have gain unnnecessary extra weight(obesity)" in symptom[i]):
            disease.append("obesity")

        elif("" in symptom[i]):
            disease.append(0)

    df=pd.read_csv("static/Training_Disease.csv")
    cols=df.columns
    l1=list(cols[:-1])
    disease_values=['Fungal infection','Allergy','Drug Reaction',
    'Peptic ulcer disease',
    ' Migraine',
    'Paralysis (brain hemorrhage)','Jaundice','Chicken pox',
    'Common Cold','Dimorphic hemmorhoids(piles)',
    'Varicoseveins','Hypothyroidism',
    'Arthritis','(vertigo) Paroymsal  Positional Vertigo','Acne','Urinary tract
infection','Psoriasis',
    'Impetigo']

    l2=[]
    for x in range(0,len(l1)):
        l2.append(0)

    for k in range(0,len(l1)):
        for z in disease:
            if(z==l1[k]):
                l2[k]=1

    disease_symp = [l2]
    print(disease_symp)
```

```
    svm_model_file = open("SVM_disease_model","rb")
    model = pickle.load(svm_model_file)
    predict_svm=model.predict(disease_symp)

    rf_model_file = open("RandomForest_disease_model","rb")
    model = pickle.load(rf_model_file)
    predict_rf=model.predict(disease_symp)

    dt_model_file = open("DecisionTreet_disease_model","rb")
    model = pickle.load(dt_model_file)
    predict_dt=model.predict(disease_symp)
    list_Disease=[]
    for a in range(0,len(disease_values)):
        if(predict_dt == a):
            list_Disease.append(disease_values[a])
    for a in range(0,len(disease_values)):
        if(predict_rf == a):
            list_Disease.append(disease_values[a])
    for a in range(0,len(disease_values)):
        if(predict_svm == a):
            list_Disease.append(disease_values[a])

    return list_Disease





def getResponse(ints, intents_json):
    result_question=[]
    result_option=[]
    result_type=[]
    result_tag=[]
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result_tag.append(tag)
            result = i['responses']
            for j in range(0,len(result)):
              result_question.append(result[j]['question'])
              result_option.append(result[j]['options'])
              result_type.append(result[j]['type'])
            break

    return result_question,result_option,result_tag,result_type
def chatbot_response(text):
```

```
    json_file = open('model.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    model = model_from_json(loaded_model_json)
    # load weights into new model
    model.load_weights("chatbot_model.h5")
    ints = predict_class(text, model)
    res = getResponse(ints, intents)
    return res

global symptom_list
global tag_list

@app.route("/")
def hello_world():
    question='Are You Sick?'
    option=["Yes","No"]
    radio=TRUE
    check=False
    return render_template("index.html",start=True,  radio=radio,check=check,
question=question, len_option= len(option), option=option)

@app.route("/chat", methods=['GET','POST'])
def chat():
    button= request.form['input_response']
    if button=="Yes":
        disease_list=pd.read_csv("static/Category.csv")
        #option=disease_list['Category'].tolist()
        option=disease_list['Category_option'].tolist()
        question="What are the symptoms?"
        radio=False
        check=True
        return jsonify({'option':option,'radio':True,'question':question})
        #return
render_template("index.html",start=False,  radio=radio,check=check,
question=question, len_option= len(option), option=option)
    if button=="No":
        user_msg="Lets start the chat"

        return jsonify({'user_msg':user_msg})
    return jsonify({'error':"This is an error mate"})



@app.route("/chat_disease", methods=['GET','POST'])
def Chat_disease():

    type_q= request.form['type']
```

```
    symptom_list=""
    if(type_q=="nothing"):

        symp=""
        disease= request.form['input_response2']
        print(disease)
        print(len(disease))
        symptom= disease.split(",")
        option=[]
        result_ques,result_opt,result_tag,result_type=
chatbot_response(symptom[0])
        for i in range(0,len(result_opt)):
            for j in result_opt[i]:
                option.append(j)
        symptom.remove(symptom[0])
        for i in symptom:
            symp+=str(i)+","
        l=len(symp)
        symp_1 = symp[:l-1]

        return
jsonify({'result_ques':result_ques[0][0],'symptom_list':"",'symptom':symp_1,'resu
lt_opt':option,'result_tag':result_tag[0],'type':result_type[0][0]})
    else:

        option_chosen= request.form['input_response2']
        tag_selected=request.form['type']
        symptom_list_resp=request.form['symptom_list']
        symptom_list+=symptom_list_resp+","+option_chosen
        symptom=request.form['symptom']
        bin=option_chosen.split(",")

        sym=symptom

        if(len(symptom)>0):
            symptom= sym.split(",")
            symp=""
            print(len(symptom))
            option=[]
            result_ques,result_opt,result_tag,result_type=
chatbot_response(symptom[0])
            for i in range(0,len(result_opt)):
                for j in result_opt[i]:
                    option.append(j)
            symptom.remove(symptom[0])
            for i in symptom:
                symp+=str(i)+","
            l=len(symp)
```

```
            symp_1 = symp[:l-1]
            return
jsonify({'result_ques':result_ques[0][0],'symptom_list':symptom_list,'symptom_lis
t':symptom_list,'symptom':symp_1,'result_opt':option,'result_tag':result_tag[0],'
type':result_type[0][0]})
        else:
            option_chosen= request.form['input_response2']
            tag_selected=request.form['type']
            symptom_list_resp=request.form['symptom_list']
            symptom_list_resp=symptom_list_resp+","+option_chosen
            symptom_list_resp=symptom_list_resp[1:]
            disease=get_disease(symptom_list_resp)
            print(disease)
            if(disease[0]==disease[1]==disease[2]):
                stmt="You are at very High Risk Of Having "+disease[0]
            elif(disease[0]==disease[1]):
                stmt="You have a high possibility of having "+disease[0]+" or you
might also be suffering from "+disease[2]
            elif(disease[1]==disease[2]):
                stmt="You have a high possibility of having "+disease[1]+" or you
might also be suffering from "+disease[0]
            elif(disease[0]==disease[2]):
                stmt="You have a high possibility of having "+disease[0]+" or you
might also be suffering from "+disease[1]
            else:
                stmt="There is a possibility you might be suffering from
"+disease[0]+", "+disease[1]+" or "+disease[2]
            return jsonify({'stmt':stmt,'symp':symptom_list_resp})

if __name__ == '__main__':
   app.run(debug=True)
```

Chatbot.py

```
import nltk
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
import tensorflow as tf
from keras.optimizers import SGD
import random
```

```
words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('static/Disease_Category_intenets_.json').read()
intents = json.loads(data_file)
for intent in intents['intents']:
    for pattern in intent['patterns']:
        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        print(words)
        #add documents in the corpus
        documents.append((w, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
print(documents)
# lemmatize, lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
# sort classes
type(classes)
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print (len(documents), "documents")
# classes = intents
print (len(classes), "classes", classes)
# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)
pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)
# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # lemmatize each word - create base word, in attempt to represent related
words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in
pattern_words]
    # create our bag of words array with 1, if word match found in current
pattern
```

```
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)
    # output is a '0' for each tag and '1' for current tag (for each pattern)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1
    training.append([bag, output_row])
# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:,0])
train_y = list(training[:,1])
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))
#use sgd with nestrov gradient descent
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['accuracy'])
#save model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5,
verbose=1)
model.save('chatbot_model.h5', hist)
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("chatbot_model.h5")
print("Saved model to disk")
```

Disease_prediction.py

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
import pickle
df=pd.read_csv("static/Training_Disease.csv")
cols=df.columns
l1=list(cols[:-1])
disease=['Fungal infection','Allergy','Drug Reaction','Peptic ulcer
diseae','Migraine','Paralysis (brain hemorrhage)','Jaundice','Chicken
pox','Common Cold','Dimorphic
```

```
hemmorhoids(piles)','Varicoseveins','Hypothyroidism','Arthritis','(vertigo)
Paroymsal  Positional Vertigo','Acne','Urinary tract
infection','Psoriasis','Impetigo']

l2=[]
for x in range(0,len(l1)):
    l2.append(0)

df.replace({'prognosis':{'Fungal infection':0,'Allergy':1,'Drug Reaction':2,
'Peptic ulcer diseae':3,'Migraine':4,'Paralysis (brain
hemorrhage)':5,'Jaundice':6,'Chicken pox':7,'Common Cold':8,'Dimorphic
hemmorhoids(piles)':9,
'Varicose veins':10,'Hypothyroidism':11,'Arthritis':12,
'(vertigo) Paroymsal  Positional Vertigo':13,'Acne':14,'Urinary tract
infection':15,'Psoriasis':16,
'Impetigo':17}},inplace=True)

# print(df.head())

X= df[l1]
print(X)
y = df[["prognosis"]]
print(y)

np.ravel(y)

tr=pd.read_csv("static/Testing_Disease.csv")
tr.replace({'prognosis':{'Fungal infection':0,'Allergy':1,'Drug Reaction':2,
'Peptic ulcer diseae':3,'Migraine':4,'Paralysis (brain
hemorrhage)':5,'Jaundice':6,'Chicken pox':7,'Common Cold':8,'Dimorphic
hemmorhoids(piles)':9,
'Varicose veins':10,'Hypothyroidism':11,'Arthritis':12,
'(vertigo) Paroymsal  Positional Vertigo':13,'Acne':14,'Urinary tract
infection':15,'Psoriasis':16,
'Impetigo':17}},inplace=True)

X_test= tr[l1]
y_test = tr[["prognosis"]]
np.ravel(y_test)


svm = SVC()
svm=svm.fit(X,np.ravel(y))


y_pred=svm.predict(X_test)
```

```
with open('SVM_disease_model', 'wb') as files:
    pickle.dump(svm, files)

clf=RandomForestClassifier()
clf=clf.fit(X,np.ravel(y))

with open('RandomForest_disease_model', 'wb') as files:
    pickle.dump(clf, files)

classifier=DecisionTreeClassifier()
classifier=classifier.fit(X,np.ravel(y))

with open('DecisionTreet_disease_model', 'wb') as files:
    pickle.dump(classifier, files)
```