### 4.1.1 IMPLEMENTATION OF HEALTH CARE CHATBOT WITH SYMPTOM PREDICTION AND ADVISIORY SYSTEM USING SVM AND DECISION TREE CLASSIFIER ALGORITHM IN PYTHON

import re

import pandas as pd

from sklearn import preprocessing

from sklearn.tree import DecisionTreeClassifier, \_tree

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.svm import SVC

import numpy as np

import csv

import warnings

warnings.filterwarnings("ignore", message="X does not have valid feature names")

training = pd.read\_csv('Training.csv')

testing = pd.read\_csv('Testing.csv')

cols = training.columns

cols = cols[:-1]

x = training[cols]

y = training['prognosis']

y1 = y

# Grouping data based on 'prognosis' column

reduced\_data = training.groupby(training['prognosis']).max()

# Mapping strings to numbers

le = preprocessing.LabelEncoder()

le.fit(y)

y = le.transform(y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.33, random\_state=42)

testx = testing[cols]

testy = testing['prognosis']

testy = le.transform(testy)

clf = DecisionTreeClassifier()

clf = clf.fit(x\_train, y\_train)

scores = cross\_val\_score(clf, x\_test, y\_test, cv=3)

print("Decision Tree Classifier Mean Score:", scores.mean())

model = SVC()

18

model.fit(x\_train, y\_train)

print("SVM Score:")

print(model.score(x\_test, y\_test))

importances = clf.feature\_importances\_

indices = np.argsort(importances)[::-1]

features = cols

severityDictionary = {}

description\_list = {}

precautionDictionary = {}

symptoms\_dict = {}

for index, symptom in enumerate(x):

symptoms\_dict[symptom] = index

def calc\_condition(exp, days):

sum = 0

for item in exp:

sum = sum + severityDictionary[item]

if (sum \* days) / (len(exp) + 1) > 13:

print("You should take the consultation from doctor. ")

else:

print("It might not be that bad but you should take precautions.")

def getDescription():

global description\_list

with open('symptom\_Description.csv') as csv\_file:

csv\_reader = csv.reader(csv\_file, delimiter=',')

line\_count = 0

for row in csv\_reader:

\_description = {row[0]: row[1]}

description\_list.update(\_description)

def getSeverityDict():

19

global severityDictionary

with open('Symptom\_severity.csv') as csv\_file:

csv\_reader = csv.reader(csv\_file, delimiter=',')

line\_count = 0

try:

for row in csv\_reader:

\_diction = {row[0]: int(row[1])}

severityDictionary.update(\_diction)

except:

pass

def getprecautionDict():

global precautionDictionary

with open('symptom\_precaution.csv') as csv\_file:

csv\_reader = csv.reader(csv\_file, delimiter=',')

line\_count = 0

for row in csv\_reader:

\_prec = {row[0]: [row[1], row[2], row[3], row[4]]}

precautionDictionary.update(\_prec)

def getInfo():

print("-----------------------------------HealthCare ChatBot-----------------------------------")

print("\nYour Name? \t\t\t\t", end="->")

name = input("")

print("Hello, ", name)

def check\_pattern(dis\_list, inp):

pred\_list = []

inp = inp.replace(' ', '\_')

patt = f"{inp}"

regexp = re.compile(patt)

pred\_list = [item for item in dis\_list if regexp.search(item)]

if(len(pred\_list) > 0):

return 1, pred\_list

else:

return 0, []

20

def tree\_to\_code(tree, feature\_names):

tree\_ = tree.tree\_

feature\_name = [

feature\_names[i] if i != \_tree.TREE\_UNDEFINED else "undefined!"

for i in tree\_.feature

]

chk\_dis = ",".join(feature\_names).split(",")

symptoms\_present = []

print("\nEnter the symptoms you are experiencing (separated by commas): ")

symptoms\_input = input("").strip().split(',')

symptoms\_input = [symptom.strip() for symptom in symptoms\_input]

# Check if all symptoms are valid

invalid\_symptoms = [symptom for symptom in symptoms\_input if symptom not in chk\_dis]

if invalid\_symptoms:

print(f"The following symptoms are not valid: {', '.join(invalid\_symptoms)}. Please try again.")

return

while True:

try:

num\_days = int(input("Okay. From how many days ? : "))

break

except ValueError:

print("Please enter a valid number of days.")

def recurse(node, depth):

indent = " " \* depth

if tree\_.feature[node] != \_tree.TREE\_UNDEFINED:

name = feature\_name[node]

threshold = tree\_.threshold[node]

if name in symptoms\_input:

val = 1

else:

21

val = 0

if val <= threshold:

recurse(tree\_.children\_left[node], depth + 1)

else:

symptoms\_present.append(name)

recurse(tree\_.children\_right[node], depth + 1)

else:

present\_disease = print\_disease(tree\_.value[node])

red\_cols = reduced\_data.columns

symptoms\_given = red\_cols[reduced\_data.loc[present\_disease].values[0].nonzero()]

print("Are you experiencing any ")

symptoms\_exp = []

for syms in list(symptoms\_given):

if syms not in symptoms\_input: # Check if symptom has not already been specified

inp = ""

print(syms, "? : ", end='')

while True:

inp = input("")

if inp == "yes" or inp == "no":

break

else:

print("Provide proper answers i.e. (yes/no) : ", end="")

if inp == "yes":

symptoms\_exp.append(syms)

second\_prediction = sec\_predict(symptoms\_exp)

calc\_condition(symptoms\_exp, num\_days)

if present\_disease[0] == second\_prediction[0]:

print("You may have ", present\_disease[0])

print(description\_list[present\_disease[0]])

else:

print("You may have ", present\_disease[0], "or ", second\_prediction[0])

print(description\_list[present\_disease[0]])

print(description\_list[second\_prediction[0]])

precaution\_list = precautionDictionary[present\_disease[0]]

print("Take following measures : ")

for i, j in enumerate(precaution\_list):

22

print(i + 1, ")", j)

recurse(0, 1)

def print\_disease(node):

node = node[0]

val = node.nonzero()

disease = le.inverse\_transform(val[0])

return list(map(lambda x: x.strip(), list(disease)))

def sec\_predict(symptoms\_exp):

df = pd.read\_csv('Training.csv')

X = df.iloc[:, :-1]

y = df['prognosis']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=20)

rf\_clf = DecisionTreeClassifier()

rf\_clf.fit(X\_train, y\_train)

symptoms\_dict = {symptom: index for index,

symptom in enumerate(X)}

input\_vector = np.zeros(len(symptoms\_dict))

for item in symptoms\_exp:

input\_vector[[symptoms\_dict[item]]] = 1

return rf\_clf.predict([input\_vector])

getSeverityDict()

getDescription()

getprecautionDict()

getInfo()

tree\_to\_code(clf, cols)

print("----------------------------------------------------------------------------------------")

23