Hacettepe University
Department of Computer Engineering

BBM103 Assignment 2 Report

Sudenaz Yazıcı – 2210356008

23.11.2022

Analysis

In this problem, we are asked to code a program that takes the input from a text file and writes the output into a text file. We are designing a clinical decision support system (CDSS) called Doctor’s Aid. With this program, we are able to save a new patient’s informations ,delete patients, list all the patients, calculate patient’s probability of having the disease and recommend treatment according to taken input.

Design

In this section, I will briefly explain how the functions work.

Reading the Input File Function

Reading function gets inputs from a text file and runs other functions according to the taken input.

Creating New Patient Funciton

Creating function saves the patient’s informations into a list. If the patient already exists, it does not save.

Removing Patient Funciton

Removing function deletes the patient from the list. If the patient does not exist, it cannot delete the patient.

Disease Probability Function

Probability function calculates the probability of the patient having the disease according to given input.

Recommendation Function

Recommendation function calculates the probability again. According to that probability, the program gives recommendation to the patient.

Listing Function

Listing functions saves output into a text file as a table.

Saving to the Output File Function

Saving function writes what needs to be written into a text file.

Functions

```
import os

current_dir_path = os.getcwd()
```

Since we are working with files and directories in this problem, we must import os and assign the current working directory.

Reading the Input File Function

```
def read_file():
    reading_file_name = "doctors_aid_inputs.txt"
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
```

The main part of the program is inside of this function. Firstly, we assign the file which we will read. With os.path.join() function, the program combines two path into a single path.

```
    with open(reading_file_path, "r") as f:
        global patient_list
        patient_list = []
```

The program opens and reads the file as f but we will use f just a bit later. To be able to reach the patient_list we define it as global. Since we do not have any patients at first, we assign patient_list as an empty list.

```
for line in "doctors_aid_inputs.txt":
    line_list = f.readline().split(" ",1)
    command = line_list[0]
    if "\n" not in line_list[-1]:
        line_list[-1]=line_list[-1]+"\n" # since there is no \n at the last
line

    if command == "create":
        data = line_list[1].split(", ")
        data[-1]= data[-1][:-1] # removing \n from the last item
        create(patient_list, data)
    elif command == "remove":
        data = [line_list[1]]
        data[-1] = data[-1][:-1] # removing\n from the last item
        remove_patient(patient_list, data)

    elif command == "probability":
        data = [line_list[1]]
        data[-1] = data[-1][:-1] # removing \n from the list item
        probability(patient_list,data)
    elif command == "recommendation":
        data = [line_list[1]]
        data[-1] = data[-1][:-1] # removing \n from the last item
        recommendation(patient_list,data)
    elif command == "list\n":
        show_list()
```

The main part of the program is this for loop. We want the program to stop when there is no line, so we use for loop. The program reads the input file one line at a time with readline() function. We are splitting it from the first space to seperate the commands and the patient informations. Every line has “\n” at the end except the last line, so we are adding “\n” to the last line because we will erase these characters later. Then, the program checks the commands and takes action. In “create” part, we are splitting the patient’s informations. Except the “list\n” part, we are removing “\n” from the last items.

Shortly, the program reads inputs from the text file. According to these inputs, it calls other functions. Inside of this function, we are assigning a list to hold the patients’ informations.

Creating New Patient Funciton

```
def create(x,y):  # x = patient_list   y = data
    if [y] in x:
        save_to_file("Patient {} cannot be recorded due to 
duplication.".format(y[0]))
    else:
        x += [y]
        save_to_file("Patient {} is recorded.".format(y[0]))
```

Create function saves a patient into patient list. If the patient already exists, the program recognizes the patient and does not save it. Since patient’s informations are in a list and we are saving it into another list, we are creating a multi-dimensional list.

Note that save_to_file() function is the function that saves to output file and I will explain that function later.

Removing Patient Funciton

```
def remove_patient(x,y):  # x = patient_list   y = data
    for j in range(len(x)):
        if y[0] == x[j][0]:
            del x[j]
            save_to_file("Patient {} is removed.".format(y[0]))
            return 0
    save_to_file("Patient {} cannot be removed due to 
absence.".format(y[0]))
```

Removing function firstly checks every item of the patient_list with for loop. If it finds a patient with the same name, it deletes the patient and writes to output file. Then, it returns zero to break the loop. If the program cannot find the patient, it writes that the patient cannot be removed.

Disease Probability Function

```
def probability(x,y):  # x = patient_list   y = data
    for k in range(len(x)):

        if y[0] == x[k][0]:
            floats = []
            floats.append(x[k][1]) # adding diagnosis accuracy
            floats.append(x[k][3]) # adding disease incidence
            floats.append(x[k][5]) # adding treatment risk
            floats = [eval(number) for number in floats]
            prob = (floats[1]/(floats[1] + 1-floats[0]))*100 # (rate of 
people having the disease)/(rate of people having the disease + probability 
of wrong calculation)
```

```
            prob = ("{:.2f}".format(prob))
            prob = float(prob)
            prob = str(prob) + "%"
            save_to_file("Patient {} has a probability of {} of having
{}.".format(x[k][0],prob,x[k][2]))
            return 0
    save_to_file("Probability for {} cannot be calculated due to
absence.".format(y[0]))
```

Probability function firstly checks every item of the patient_list with for loop. If it finds a patient with the same name, it converts some of the string informations to floats. Then, it calculates the probability of having the disease of the patient and writes it. Then, we are using “return 0” to break the loop. If the program cannot find the function, it writes that the probability cannot be calculated.

Recommendation Function

```
def recommendation(x, y): # x = patient_list   y = data
    for k in range(len(x)):
        if y[0] == x[k][0]:
            floats = []
            floats.append(x[k][1]) # adding diagnosis accuracy
            floats.append(x[k][3]) # adding disease incidence
            floats.append(x[k][5]) # adding treatment risk
            floats = [eval(number) for number in floats]
            prob = (floats[1]/(floats[1] + 1-floats[0]))*100
            prob = ("{:.2f}".format(prob))
            prob = float(prob)
            floats[2]=floats[2]*100
            if prob > floats[2]:
                save_to_file("System suggests {} to have the
treatment.".format(x[k][0]))
            else:
                save_to_file("System suggests {} NOT to have the
treatment.".format(x[k][0]))
```

Recommendation function calculates the probability of having the disease again with the same steps. If probability is greater than the treatment risk, function suggests patient to have the treatment. Otherwise, it does not suggests.

Listing Function

```
def show_list():

save_to_file("Patient\tDiagnosis\tDisease\t\tDisease\tTreatment\t\tTreatmen
t")
    save_to_file("Name\tAccuracy\tName\t\tIncidence\tName\t\tRisk")
    save_to_file("-"*75)
    for patient in patient_list:
        floats = []
        floats.append(patient[1])
        floats.append(patient[5])
        floats = [eval(number) for number in floats]
        floats[0] = floats[0] * 100
        floats[1] = floats[1] * 100
        floats[0] = str(floats[0]) + "%"
        floats[1] = str(floats[1]) + "%"

save_to_file(patient[0]+"\t"+floats[0]+"\t"+patient[2]+"\t\t"+patient[3]+"\
t"+patient[4]+"\t\t"+floats[1])
```

Listing function basically writes the informations of the patients as a table. Before doing that, the program converts some strings to floats and then to strings again so that we can get the output as the wanted form(99.9%, 40.0% etc.)

Saving to the Output File Function

```
def save_to_file(i):
    writing_file_name = "doctors_aid_outputs.txt"
    writing_file_path = os.path.join(current_dir_path, writing_file_name)

    with open(writing_file_path, "a") as f:
        f.write(i+"\n")
```

Saving function firstly assigns the file whic the program will write into. With os.path.join() function, the program combines two path into a single path. Then, it opens as f and writes what needs to be written inside.

Programmer's Catalogue

Time Spent for Analysis: Analysis section includes reading and understanding the problem, understanding the calculation of probability. Since the problem is not very hard to understand this section took me about half an hour to one hour.

Time Spent for Design: Design section icludes writing a pseudocode and thinking about the data structures I used. This section took me about one hour.

Time Spent for Implementation: Implementation section took more time than other sections since I tried to learn some new things and I ran into some errors. I figured out how to read a text file, write to a text file and calculate the probability of the patient having the disease. Then, I coded everything according to my pseudocode. Overall, I spent 10-15 hours on this section.

Time Spent for Testing: Since the possible situations we must control are given, this part did not take long time. Overall I can say I spent half an hour on this section.

Time Spent for Reporting: Reporting section includes writing and explaining the problem and the program. Since this is my first report I looked up some examples and tried to learn basics of writing report. This section took me about 5 hours.

```python
import os

current_dir_path = os.getcwd()

def save_to_file(i):
    writing_file_name = "doctors_aid_outputs.txt"
    writing_file_path = os.path.join(current_dir_path, writing_file_name)

    with open(writing_file_path, "a") as f:
        f.write(i+"\n")

def create(x,y):  # x = patient_list   y = data
    if [y] in x:
        save_to_file("Patient {} cannot be recorded due to
duplication.".format(y[0]))
    else:
        x += [y]
        save_to_file("Patient {} is recorded.".format(y[0]))

def remove_patient(x,y):  # x = patient_list   y = data
    for j in range(len(x)):
        if y[0] == x[j][0]:
            del x[j]
            save_to_file("Patient {} is removed.".format(y[0]))
            return 0
    save_to_file("Patient {} cannot be removed due to
absence.".format(y[0]))

def probability(x,y):  # x = patient_list   y = data
    for k in range(len(x)):

        if y[0] == x[k][0]:
            floats = []
            floats.append(x[k][1]) # adding diagnosis accuracy
            floats.append(x[k][3]) # adding disease incidence
            floats.append(x[k][5]) # adding treatment risk
            floats = [eval(number) for number in floats]
            prob = (floats[1]/(floats[1] + 1-floats[0]))*100 # (rate of
people having the disease)/(rate of people having the disease + probability
of wrong calculation)
            prob = ("{:.2f}".format(prob))
            prob = float(prob)
            prob = str(prob) + "%"
            save_to_file("Patient {} has a probability of {} of having
{}.".format(x[k][0],prob,x[k][2]))
            return 0
    save_to_file("Probability for {} cannot be calculated due to
absence.".format(y[0]))

def recommendation(x, y): # x = patient_list   y = data
    for k in range(len(x)):
        if y[0] == x[k][0]:
            floats = []
            floats.append(x[k][1]) # adding diagnosis accuracy
            floats.append(x[k][3]) # adding disease incidence
            floats.append(x[k][5]) # adding treatment risk
            floats = [eval(number) for number in floats]
            prob = (floats[1]/(floats[1] + 1-floats[0]))*100
            prob = ("{:.2f}".format(prob))
            prob = float(prob)
            floats[2]=floats[2]*100
            if prob > floats[2]:
                save_to_file("System suggests {} to have the
treatment.".format(x[k][0]))
            else:
                save_to_file("System suggests {} NOT to have the
treatment.".format(x[k][0]))

def show_list():

save_to_file("Patient\tDiagnosis\tDisease\t\tDisease\tTreatment\t\tTreatmen
t")
    save_to_file("Name\tAccuracy\tName\t\tIncidence\tName\t\tRisk")
    save_to_file("-"*75)
    for patient in patient_list:
        floats = []
        floats.append(patient[1])
        floats.append(patient[5])
        floats = [eval(number) for number in floats]
        floats[0] = floats[0] * 100
        floats[1] = floats[1] * 100
        floats[0] = str(floats[0]) + "%"
        floats[1] = str(floats[1]) + "%"

save_to_file(patient[0]+"\t"+floats[0]+"\t"+patient[2]+"\t\t"+patient[3]+"\
t"+patient[4]+"\t\t"+floats[1])


def read_file():
    reading_file_name = "doctors_aid_inputs.txt"
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
    with open(reading_file_path, "r") as f:
        global patient_list
        patient_list = []
        for line in "doctors_aid_inputs.txt":
            line_list = f.readline().split(" ",1)
            command = line_list[0]
            if "\n" not in line_list[-1]:
                line_list[-1]=line_list[-1]+"\n" # since there is no \n at
the last line

            if command == "create":
                data = line_list[1].split(", ")
                data[-1]= data[-1][:-1] # removing \n from the last item
                create(patient_list, data)
            elif command == "remove":
                data = [line_list[1]]
                data[-1] = data[-1][:-1] # removing\n from the last item
                remove_patient(patient_list, data)
            elif command == "probability":
                data = [line_list[1]]
                data[-1] = data[-1][:-1] # removing \n from the list item
                probability(patient_list,data)
            elif command == "recommendation":
                data = [line_list[1]]
                data[-1] = data[-1][:-1] # removing \n from the last item
                recommendation(patient_list,data)
            elif command == "list\n":
                show_list()
read_file()
```

Where to use this program ?

This program can be used not only as saving patients’ informations but also as saving informations for other places. For instance, an online shopping website can use some parts of this code and store customer informations. Instead of calculating probability, it can calculate the amount of money that customer owes.

User's Catalogue

Program's user manual/tutorial

1.Open the doctors_aid_inputs.txt file.

2.Write the steps that you want to do.

3.Run the Assignment2.py program.

4.Go to doctors_aid_outputs.txt and see the results.

Restrictions on the program

For the possibility() function, possibility of the user giving invalid values that need to be floats at the create part is ignored.

Evaluation	Points	Evaluate Yourself / Guess Grading
Indented and Readable Codes	5	5
Using Meaningful Naming	5	5
Using Explanatory Comments	5	5
Efficiency (avoiding unnecessary actions)	5	0
Function Usage	25	25
Correctness	35	30
Report	20	15
There are several negative evaluations	…	…