Hacettepe University

Department Of Artificial Intelligence

BBM103 Assignment 2 Report

Making a Patient Data Control System

Adam Sattout - b2220765061

27/11/2022

1 Analysis

Here I will explain the problem we are facing and explain the outcomes needed.

In this assignment we need to make a reliable system for any member of a health department staff to be able to give the essential commands like inserting a new patient's data into the system or calculating the probability of having the diagnosed disease for the specified patient according to some information given beforehand like diagnoses accuracy and incidence rate. This system can be essential in our daily life to make recording different patients and calculating the exact result of a patient much easier than having to write down everything and calculating it manually, which can also be faster and give more accurate results.

2 Design

In this section I will talk about the solution to the problem that I used in this project.

For this problem we first need to take the input from the user through an already specified input text file, put it into a list using the function `re.split()` from native-python library "re" to be able to split the input text by commas and new lines, and use it as the data and commands for our program. Then we tackle the different properties of lists to modify the multidimensional main list of patients, for example, if the user wanted to remove a patient called Sam for example, the program will first check if such patient is in the main list of patients, if not it reports that there is no such patient, if there is then the program will simply remove the list related to this patient with his data.

2.1 Implementation

For this section, I will furthermore expand upon each function I have created in this program before I provide the whole code in the programmer catalog section.

2.1.1 Reading The Input File

```
def fileReading():
    global inputFile
    global outputFile
    f = open('doctors_aid_inputs.txt').read()
    inputFile = re.split('\n|,' , f)
    outputFile = open('doctors_aid_output.txt', "w")
    for n in inputFile:
        innerList.append(n.split(" "))
```

After we create the heading of the main multidimensional list, we define this function that starts by identifying both output file variable and input file list, then it opens the input file speified, reads it, and splits its content into a list called "inputFile", after that it

opens an output file with a specified name, and finally it creates another list deviated from the input list to reach the inputs that were separated by a space instead of a comma or a new line.

Create a variable equal to input file text
Create a list by splitting the input file by \n and ,
Create an output file
Create an inner list split by spaces

2.1.2 Registering a New Patient

```
def create(x):
    patientFound = 0
    for patient in patientsList:
        for data in patient:
            if innerList[x][1] == data:
                patientFound = 1
                outputFile.write('Patient ' + ''.join(innerList[x][1]) + " cannot be recorded due to
duplication.\n")
    if patientFound == 0:
        l = []
        l.append("\n")
        l.append(innerList[x][1])
        l.append("{:.2%}".format(float(inputFile[x+1])))
        for n in range(x+2,x+5):
            l.append(inputFile[n])
        l.append("{:.0%}".format(float(inputFile[x+5])))
        patientsList.append(l)
        outputFile.write('Patient ' + ''.join(innerList[x][1]) + " is recorded.\n")
```

For recording a new patient, we first check if the patient was already registered by simply setting a value to 0 and changing it to 1 in a loop that repeatedly checks if the element of the input file list is equal to the name of the patient that the user wants to create. If so, the value would be equal to 1 and the output would simply inform the user that a user with this name has already been registered. If not, then we will need to create a new sublist with the patient's information and add it to the main list "patientList", I chose to take them by the index of the current list element being read from the input list (being x) and basically taking what comes after the name index, for example innerList[x+1] refers to the name of the patient. inputFile[x+1] indicates the patient's diagnoses accuracy, and since the input of it is taken in the form a decimal number, I needed to change it to a percentage with 2 decimal places, then comes the patient's diagnosed disease name, incidence rate, and treatment method which are pretty straight forward elements being

simple strings, then comes the treatment risk which I also needed change to a percentage. After the new sublist has been added to the main one, a new message will be printed into the output file informing the user that the patient has been recorded.

```
Loop in main list values
If patient is found
  Inform the user that there cant be duplications
If patient is not found
   Add patient information into a sublist
   Add the sublist to the main list
```

2.1.3 Removing a Patient

```
def remove(x):
   patientFound = 0
   for a in patientsList:
         for b in a:
            if innerList[x][1] == b:
               patientsList.remove(a)
               string = 'Patient ' + ''.join(innerList[x][1]) + ' is removed.\n'
               patientFound = 1
   if patientFound != 1:
      string = 'Patient ' + ''.join(innerList[x][1]) + " cannot be removed due to absence.\n"
   outputFile.write(string)
```

For removing the patient, we first use a similar way to detect if the patient the user wants to remove is in the main list of patients. If not, the program will again inform the user that the patient isn't in the list of patients. If the patient was found, the program would remove the sublist corresponding to the patient and print a message confirming that the patient was removed from the system.

```
Loop in main list
    If patient is found
      Remove the patient
      Inform the user
If patient is not found
   Inform the user
```

2.1.4 Listing The Patients

```
def list():
    for patient in patientsList:
        for data in patient:
            if data != "\n":
                outputFile.write("{:<20}".format(data))
            else:
                outputFile.write(data)
    outputFile.write("\n")
```

If the user enters the command “list”, the program will print each bit of information into a column with 20 characters width with getting into a new line after each patient to create an organized and readable list into the specified text file

Loop in main list
    If data isn’t a new line
        Print the data into a 20 space column
    Else
        Print the data normally
Print a new line after each patient

2.1.5 Calculating a Patient’s Probability of Having a Disease

```
def probability(x):
    global recommendationBool
    patientFound = 0
    newLine = 0
    for patient in patientsList:
        for data in patient:
            if data == "\n":
                newLine = 1
            if innerList[x][1] == data:
                patientFound = 1
                if newLine == 0:
                    calculateList = a[3].split("/")
                    accuracy = float(a[1].strip("%"))
                else:
                    calculateList = patient[4].split("/")
                    accuracy = float(patient[2].strip("%"))
                infected = float(calculateList[0])
                healthy = float(calculateList[1])
                probability = 100-(100*(1-(accuracy*infected/((100 - accuracy)*healthy+accuracy*infected))))
                probabilityString = "{:.2f}".format(probability)
                probabilityString = probabilityString.replace(".00", "")
```

```
            string = "Patient " + ''.join(innerList[x][1]+ " has a probability of " + probabilityString + "% of
having" + patient[3].lower() + ".\n")
            if recommendationBool == False:
              outputFile.write(string)
            return probability
    if patientFound == 0:
       string = 'Probability for ' + '' join(innerList[x][1]) + " cannot be calculated due to absence.\n"
    outputFile.write(string)
```

To calculate a patient's probability of having the diagnosed disease, after checking if the patient is in the list, we take 2 important information about the patient, his diagnoses accuracy and healthy to infected ratio of the disease he or she might have, then we use this equation to calculate the value we need and insert it to the string message that the user will receive:-

100-(100*(1-(accuracy*infected/((100- accuracy)*healthy+accuracy*infected))))

After that we check if this function was called directly by the user or indirectly by the recommendation function which I will cover in the next subsection.

Loop in main list
  If patient is found
    Get patient's diagnoses accuracy and disease's incidence rate
    Apply the variables into the equation
    If this function wasn't called by recommendation
      Inform the user about the probability
  If patient wasn't found
    Inform the user

2.1.6 Recommending Treatment to a Patient

```
def recommendation(x):
    global recommendationBool
    recommendationBool = True
    patientFound = 0
    for a in patientsList:
       for b in a:
          if b == innerList[x][1]:
             patientFound = 1
             if probability(x) > float(a[6].strip("%")):
                string = "System suggests " + ''.join(innerList[x][1]) + " to have the treatment.\n"
             elif probability(x) < float(a[6].strip("%")):
                string = "System suggests "+ ''.join(innerList[x][1]) + " NOT to have the treatment.\n"
    if patientFound == 0:
       string = "Recommendation for " + innerList[x][1] + " cannot be calculated due to absence.\n"
    outputFile.write(string)
    recommendationBool = False
```

To recommend if the patient should have the specified treatment we simply compare the treatment risk percentage with the probability that the patient has the specified disease, if the risk percentage is larger than the probability, then the program will tell the user that its better to do the treatment, and vice versa.

```
Make recommendationbool = 1 to differentiate the probability function call method
Loop in main list
  If patient is found
    Compare probability percentage and treatment risk percentage
    If probability percentage > treatment risk
      Inform the user that it is better to have the treatment
    Else
      Inform the user that it is worse to have the treatment
  If patient is not found
    Inform the user that there is no such patient
```

2.1.7 Giving The Results

```
def outputWriting():
    for i in range(len(inputFile)):
        innyList = inputFile[i].split(" ")
        if innyList[0] == "create":
            create(i)
        elif innyList[0] == "remove":
            remove(i)
        elif innyList[0] == "list":
            list()
        elif innyList[0] == "probability":
            probability(i)
        elif innyList[0] == "recommendation":
            recommendation(i)
```

This function can be considered as the main distributer of the whole program. In this function, the program will check the input file for any of the main commands that the user can give, and call the function associated with that command with the index of the list needed to operate.

3 Programmer's Catalogue

In this section I will provide the code I've written as well as some information about the time spent making this assignment

```
import re
#First, we define the lists we are gonna work with, inputFile being the input list splitted by new lines and commas, innerList being a list to reach
#names of patients that dont have a comma before them, and patientlist being the main list that has the patients added to/removed from
innerList = []
inputFile = []
recommendationBool = False
patientsList = [
   ["patient ", "Diagnoses ", "Disease ","Disease ","Treatment ", "Treatment ", "\n"],
   ["Name ", "Accuracy ", "Name ","Incidence ","Name ", "Risk ", "\n"],
   ["------------------------------------------------------------------------"]
   ]
#Then, we define the input file reading function that arranges input list and creates the output file
def fileReading():
   global inputFile
   global outputFile
   f = open('doctors_aid_inputs.txt').read()
   inputFile = re.split('\n|,' , f)
   outputFile = open('doctors_aid_output.txt', "w")

fileReading()
for n in inputFile:
   innerList.append(n.split(" "))

#here we define the create function, we check if the patient is found in the main patient list then if abscent add him/her with his/her data into the main list
def create(x):
   patientFound = 0
   for patient in patientsList:
      for data in patient:
         if innerList[x][1] == data:
            patientFound = 1
            outputFile.write('Patient ' + ''.join(innerList[x][1]) + " cannot be recorded due to duplication.\n")
   if patientFound == 0:
      l = []
      l.append("\n")
      l.append(innerList[x][1])
```

```
        l.append("{:.2%}".format(float(inputFile[x+1])))
        for n in range(x+2,x+5):
            l.append(inputFile[n])
        l.append("{:.0%}".format(float(inputFile[x+5])))
        patientsList.append(l)
        outputFile.write('Patient ' + ''.join(innerList[x][1]) + " is recorded.\n")

#defining the list function to list each patient and adding a new line after each one.
def list():
    for patient in patientsList:
        for data in patient:
            if data != "\n":
                outputFile.write("{:<20}".format(data))
            else:
                outputFile.write(data)
    outputFile.write("\n")

#defining the list of removing a patient, first checking if the patient is in the main list then removing the
patient with his data from the main list.
def remove(x):
    patientFound = 0
    for a in patientsList:
            for b in a:
                if innerList[x][1] == b:
                    patientsList.remove(a)
                    string = 'Patient ' + ''.join(innerList[x][1]) + ' is removed.\n'
                    patientFound = 1
                elif innerList[1] != b:
                    print()
    if patientFound != 1:
        string = 'Patient ' + ''.join(innerList[x][1]) + " cannot be removed due to abscence.\n"
    outputFile.write(string)

#defining the probability function, which checks whether this function was called independently first or
called from the recommendation function and if the patient
#exists then calculates the probability of having the disease mentioned.
def probability(x):
    global recommendationBool
    patientFound = 0
    newLine = 0
    for patient in patientsList:
        for data in patient:
            if data == "\n":
                newLine = 1
            if innerList[x][1] == data:
                patientFound = 1
                if newLine == 0:
                    calculateList = a[3].split("/")
```

```
                accuracy = float(a[1].strip("%"))
            else:
                calculateList = patient[4].split("/")
                accuracy = float(patient[2].strip("%"))
            infected = float(calculateList[0])
            healthy = float(calculateList[1])
            probability = 100-(100*(1-(accuracy*infected/((100 - accuracy)*healthy+accuracy*infected))))
            probabilityString = "{:.2f}".format(probability)
            probabilityString = probabilityString.replace(".00", "")
            string = "Patient " + ''.join(innerList[x][1]+ " has a probability of " + probabilityString + "% of
having" + patient[3].lower() + ".\n")
            if recommendationBool == False:
                outputFile.write(string)
            return probability
    if patientFound == 0:
        string = 'Probability for ' + ''.join(innerList[x][1]) + " cannot be calculated due to absence.\n"
    outputFile.write(string)

#defining the recommendation function which calculates the probability of the mentioned patient then
compares it with the risk of treatment and gives a suggestion whether to
#take the treatment or not.
def recommendation(x):
    global recommendationBool
    recommendationBool = True
    patientFound = 0
    for a in patientsList:
        for b in a:
            if b == innerList[x][1]:
                patientFound = 1
                if probability(x) > float(a[6].strip("%")):
                    string = "System suggests " + ''.join(innerList[x][1]) + " to have the treatment.\n"
                elif probability(x) < float(a[6].strip("%")):
                    string = "System suggests "+ ''.join(innerList[x][1]) + " NOT to have the treatment.\n"
    if patientFound == 0:
        string = "Recommendation for " + innerList[x][1] + " cannot be calculated due to absence.\n"
    outputFile.write(string)
    recommendationBool = False

#The main function of reading the input file and printing the output according to the commands of the
user.
def outputWriting():
    for i in range(len(inputFile)):
        innyList = inputFile[i].split(" ")

        if innyList[0] == "create":
            create(i)
        elif innyList[0] == "remove":
            remove(i)
```

```
        elif innyList[0] == "list":
            list()
        elif innyList[0] == "probability":
            probability(i)
        elif innyList[0] == "recommendation":
            recommendation(i)
outputWriting()
outputFile.close()
```

3.1 Time Spent On This Assignment

Time spent for analysis	This section includes reading the assignment pdf thoroughly, understanding the main problem, and identifying the main problem with the skill tested in this assignment. Overall I would say I have spent 3 hours in this section.
Time spent for design and implementation	This part might have been the longest. It includes finding a solution to the main problem with setting an algorithm to how the program would work. Whether it is how to take the input from a text file or make a list and get the right indexes for the element that you need. This part took me nearly 6 hours.
Time spent for testing and reporting	The final part of this assignment was a crucial part of this project to be able to send it flawlessly. Also the reporting to inform the user and other developers on how the program would work, so I put a lot of effort in the section as well adding up to nearly 5 hours.

For reusability, this code may be used to create any kind of system that has registering and removing clients involved. All the function utilities have been discussed in the 4.1 section.

4 User Catalogue

To be able to use this program, you must insert your inputs into the associated text file and run it with python3. Otherwise you may face difficulties running your program. As for the input syntax, you will have to mimic the syntax I am about to provide.

4.1 Input Syntax

To record a patient:

create [name], [diagnoses accuracy], [disease name], [incidence rate], [treatment method], [treatment risk]

To remove a patient:

remove [name]

To list all patients:

list

To get the probability of a patient:

probability [name]

To get the recommendation of a patient:

recommendation [name]

4.2 Output

For the output, you will be able to find it in the same folder of the program and the input file as a text file after running the program.

5 Conclusion

In this section, I would like to go through a short summary of this whole assignment and give my thoughts about it.

At the start, I gave a brief summary about what I thought the problem was and my analysis on it with the goals of my program.

Then in section 2, I gave a detailed explanation upon the used functions in the program as well as a simple pseudocode to furthermore illustrate how they would work.

In section 3, the original code was given as well as how the work was done from my prospective as a programmer and how other programmers might use the code for a different usage.

And lastly I gave the instructions for the users that will use this program so it can be used correctly and flawlessly

Evaluate	Points	Evaluate Yourself
Indented and Readable Codes	5	5
Using Meaningful Naming	5	5
Using Explanatory Comments	5	5
Efficiency	5	4
Function Usage	25	25
Correctness	35	32
Report	20	19
Total	100	95