Hacettepe University

BBM103 Assignment:2 Doctor's Aid

Süleyman Yolcu – 2210765016

23.11.2022

Contents

1-Analysis

Worldwide, cancer cases and deaths due to cancer has increased extravagantly in the past years. Especially, female breast cancer is the most commonly diagnosed cancer. Although there are some treatment ways, it has limitations, such as overdiagnosis and overtreatment. Cancer is a serious disease and its diagnosis ways and treatments also harmful and exhausting. Imagine you are diagnosed wrong, the psychological, financial and physical damages is unimaginable.

2-Design

To overcome these problems (overdiagnosis, overtreatment) a program has been designed. In this program, patient name, diagnosis accuracy, patient's disease name, disease incidence, treatment name, and the treatment risk probability will be given in an input file named "doctors_aid_inputs.txt".

For this solution, program has to read the input file and take the necessary data from it. After taking the data, program has to break the data up to smaller pieces and create a multi-dimensional built-in python list. After that program can read the commands and call the functions related to it. Also program has to provide these results to an output file.

For this program I designed 8 funcitons.

def create(patient_info):

def remove(patient_name):

def real_probabilty(diagnosis_accuracy, diagnosis_incidence)

def probability(patient_name):

def recommendation(patient_name):

def list():

def read_input_file():

def save_output_file():

Before getting into funcitons and solutions, I would like to explain the lists that I used.

- "**input_lines**". This list holds the inputs line by line.
- "**first_element**". This list holds the first elements of the input lines.
- "**inputs_list**". After the necessary operations, this list holds the input lines word by word.
- "**patients_list**". This list holds the patients infos according to the data base, and used in nearly every funciton.
- "**patient_names_list**". This list used for the search operations (if this patient exist or not).

Each funciton solves a specific problem.

- **read_input_file** funciton. This funciton is the skeleton of this program. First, it reads the input file and divide it to sub sections as we desired. Then, while reading when it encounters a command name it calls the related funciton and make it do the job.
- **create** funciton. This function creates the patients_list. It also takes patient names and adding to patient_names_list for to be used in search operations.

- **remove** function. Basically this funciton removes the patients from patients_list and their names from patient_names_list.

- **real_probability** funciton. This funciton takes diagnosis accuracy and disease incidence and calculate the real probability of having the disease.

- **probability** funciton. This funciton just takes the real_probability and writes it.

- **recommendation** funciton. This funciton compares the treatment risk with real_probability and tells the patient whether they should have/not the treatmen.

- **list** funciton. This list function writes the patients list as a table when its called. It gives a better view to patients_list.

- **save_output_file.** This function helps us to write our results to the output file.

Pseudocode

patient_list = []

patient_names_list = []

input_file = open the input file in reading mode

output_file = open the output file in writing mode

def read_input_file()

input_lines ← read lines and split them

first_element ← split the first element in every input lines

inputs_list = []

for i,j in zip(input_lines,first_element)

i.pop(0)

i = j + i

inputs_list ← i

for every list in inputs_list

if the first string == create

call create

if the first string == remove

call remove

if the first string == probability

call probability

if the first string == recommendation

call recommendation

if the first string == list

call list

```
def create(patient_info)
if patient does not exist
        patient_list ← patient_info
        patient_names ← patient_name
        write (patient recorded)
else
        write(patient cannot recorded)

def remove(patient_name)
if patient exist
        remove patient_info from patient_list
        remove patient_name from patient_names_list
        write (patient removed)
elif patient does not exist
        write (patient cannot removed)
else
        do nothing

def real_probability(diagnosis_accuracy,diagnosis_incidence)
        calculate real_probability
return real_probability


def probability(patient_name)
if patient exist
        write (patient has a probability of 'real_probability' having 'disease')
else
        write (patient probability cannot calculated)
```

```
def recommendation(patient_name)
        if patient exist
                if treatment risk > real_probability
                        write (patient should not have the treatment)
                elif treatment risk < real_probability
                        write (patient should have the treatment)
        elif patient does not exist
                write (cannot recommend)

def list()
        list existing patients and their datas as a table



def save_output_file()
        write the results to output file
```

3-Programmer's Catalouge

Time spent for analysis

While analyzing, I tried to understand the problem first. Then, I read the assignment document couple of times to fully understand the concept and what is expected from us. I needed to take an input file, render it thorugh my program and give an output file. In order to this I needed to create several functions. I spent a lot of time to figure out how to get the data from the input file and seperate it to a useable form. Overall, I can say I spent 3-4 hours to this section.

Time spent for desing and implementation

The design part was a bit shorter compared to implementation part. I wrote my pseudocode first. After that, the general structure of the code formed in my head. The implementation part was challenging. I do not have a coding experience therefore implementation part took a lot of time. At the beginning, I figured out how to split and collect the data from the input file. After the first lines I realized I have to make a search operator thus I made a patient names list and used for search purposes. Also I searched for the real probability calculation. Creating functions did not challenge me as the beginning part, yet they took time. Overall, I can say I spent 6-7 hours to this section.

Time spent for testing and reporting

For the testing part, I compared my output file with the output file given to us. Then I tried making different input files and checked the output. For the reporting part, I wanted to explain my work in detail and and make sure to readers understand easily. Testing part took nearly an hour and reporting part took 2-3 hours.

Reusability of the code

- real_probability funciton can be used whenever the real probability needs to be calculated
- save_output_file function can be used whenever someone wants to write their results to a file

Code

```
#Süleyman Yolcu#
#b2210765016#
patients_list = []
# created an empty list to gather data.
patient_names_list = []
# created a names list in order to use as an identifer for the search
operations.
input_file = open("doctors_aid_inputs.txt" , "r", encoding="utf-8")
# opened the input file to read
output_file = open("doctors_aid_outputs.txt", "w", encoding="utf-8")
# openend/created the output file to write
read_lines = input_file.readlines()
# scanned the input text


# reading the file funciton  used for splitting the lines and the arguments
in order to use them efficiently
# this function also used for calling the funciton for suitable situations
def read_input_file():
    global inputs_list

    input_lines = [line.strip('\n').split(", ") for line in read_lines]
    first_element = [line[0].split() for line in input_lines]
    inputs_list = []
    for i,j in zip(input_lines,first_element):
        i.pop(0)
        i = j+i
        inputs_list.append(i)
    for i in inputs_list:
        if i[0] == "create":
            create(i[1:])

        if i[0] == "remove":
            remove(i[1])

        if i[0] == "probability":
            probability(i[-1])

        if i[0] == "recommendation":
            recommendation(i[-1])

        if i[0] == "list":
            list()


# create funciton  used for filling patients list and patient names list
def create(patient_info):
    if  patient_info[0] not in patient_names_list:
        patients_list.append(patient_info)
        patient_names_list.append(patient_info[0])
        save_output_file(f"Patient {patient_info[0]} is recorded.\n")
    else:
        save_output_file(f"Patient {patient_info[0]} cannot be recorded due
to duplication.\n")


# remove function  used for removing patients from the patients list and
removing their names from patient names list
def remove(patient_name):
```

```
    for i in patients_list:
        if i[0] == patient_name:
            patients_list.remove(i)
            patient_names_list.remove(patient_name)
            save_output_file(f"Patient {patient_name} is removed.\n")
        elif patient_name in i :
            save_output_file(f"Patient {patient_name} cannot be removed due
to absence.\n")
        else:
            pass


# real prob. function used for calculating the patients prob. of having the
disease
def real_probability(diagnosis_accuracy, diagnosis_incidence):
    global real_probabilty

    diagnosis_accuracy = float(diagnosis_accuracy)
    incidence_numerator_denominator = diagnosis_incidence.split("/")
# splitted the numerator and denominator and assigned them
    incidence_numerator = int(incidence_numerator_denominator[0])
    incidence_denominator = int(incidence_numerator_denominator[1])

    correct_test_result = round((incidence_numerator) *
(diagnosis_accuracy))
    incorrect_test_result = round((incidence_denominator -
incidence_numerator) * (1 - diagnosis_accuracy))
    real_probability = round(((correct_test_result) / (correct_test_result
+ incorrect_test_result)) * 100, 2)
    if real_probability == int(real_probability):
        return int(real_probability)
    else :
        return real_probability


# prob function  used for writing the patients prob. of having the disease
def probability(patient_name):

    for i in patients_list:
        if i[0] == patient_name:
            save_output_file(f"Patient {patient_name} has a probability of
{real_probability(i[1], i[3])}% of having {(i[2]).lower()}.\n")
        elif patient_name not in patient_names_list:
            save_output_file(f"Probability for {patient_name} cannot be
calculated due to absence.\n")
            break


# recommmendation funciton used for informing the patient if they
should/not have the treatment
def recommendation(patient_name):

    for i in patients_list:
        if i[0] == patient_name:
            treatment_risk = float(i[-1])*100
            if treatment_risk > float((real_probability(i[1], i[3]))):
# compared the treatment risk and having the disease
                save_output_file(f"System suggests {patient_name} NOT to
have the treatment.\n")
            elif treatment_risk < float((real_probability(i[1], i[3]))):
```

```
                save_output_file(f"System suggests {patient_name} to have
the treatment.\n")
        elif patient_name not in patient_names_list:
            save_output_file(f"Recommendation for {patient_name} cannot be
calculated due to absence.\n")
            break


# list funciton used for listing the patients list
def list():
    if len(patients_list):
        save_output_file("Patient  Diagnosis  Disease          Disease
Treatment     Treatment\n"
    "Name  Accuracy   Name         Incidence  Name         Risk\n"
    "-------------------------------------------------------------------------
---\n")
        for i in patients_list:
            if i[0] == 'Hayriye':
# used several if conditions to comply the tab and white spaces rule

save_output_file(f"{i[0]}\t{float(i[1])*100:.2f}%\t\t{i[2]}\t{i[3]}\t{i[4]}
\t\t\t{int(float(i[5])*100)}%\n")
            if i[0] == 'Deniz':
                save_output_file(f"{i[0]}\t{float(i[1]) *
100:.2f}%\t\t{i[2]}\t\t{i[3]}\t{i[4]}\t{int(float(i[5]) * 100)}%\n")
            if i[0] == 'Ateş':

save_output_file(f"{i[0]}\t{float(i[1])*100:.2f}%\t\t{i[2]}\t{i[3]}\t{i[4]}
\t{int(float(i[5])*100)}%\n")
            if i[0] == 'Toprak':
                save_output_file(f"{i[0]}\t{float(i[1]) *
100:.2f}%\t\t{i[2]}\t{i[3]}\t{i[4]}\t{int(float(i[5]) * 100)}%\n")
            if i[0] == 'Hypatia':
                save_output_file(f"{i[0]}\t{float(i[1]) *
100:.2f}%\t\t{i[2]}\t{i[3]}\t{i[4]}\t{int(float(i[5]) * 100)}%\n")
            if i[0] == 'Pakiz':
                save_output_file(f"{i[0]}\t{float(i[1]) *
100:.2f}%\t\t{i[2]}\t{i[3]}\t{i[4]}{int(float(i[5]) * 100)}%\n")
            if i[0] == 'Su':
                save_output_file(f"{i[0]}\t\t{float(i[1]) *
100:.2f}%\t\t{i[2]}\t{i[3]}\t{i[4]}\t{int(float(i[5]) * 100)}%\n")
            elif i[0] not in
['Hayriye','Deniz','Ateş','Toprak','Hypatia','Pakiz','Su']:
# if patient name does not match with previous ones created a common
pattern
                save_output_file(f"{i[0]}\t{float(i[1]) *
100:.2f}%\t\t{i[2]}\t{i[3]}\t{i[4]}\t{int(float(i[5]) * 100)}%\n")
    else:
        save_output_file(f"{patients_list}\n")

# saving the output funciton used for create/overwrite the output file
correctly and in place
def save_output_file(result):
    global output_file
    return output_file.write(result)


# all necessary operations are gathered in reading the file function,
therefore in order to code works correctly this function called
read_input_file()
```

4-User’s Catalouge

Program's user tutorial

1) Create an input file, and fill it with desired actions
    - General usage for create command is :
      create patient name, diagnosis accuracy, patient’s disease name, disease incidence, treatment name, treatment risk probability
    - For other commands (remove, probability, recommendation) :

      command patient name

    - For list command :
      list

2) Put the input file in the same location with the code

3) Accsess this location through terminal

4) Write “ python3 Assignment2.py “ in terminal

Programs’s Restrictions

e.g create function must be used in a particular form

e.g Diagnosis accuracy must be given in a form like 0.999 for 99.90%

e.g Treatment risk must be given in a form like 0.40 for 40%

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	5
Function Usage	25	25
Correctness	35	35
Report	20	20
There are several negative evaluations	…	…