Hacettepe University
Department of Computer Engineering

BBM103 Assignment 2 : Doctor's Aid Report

Ezgi Ekin - 2210356029
24.11.2022

Table of Contents

- *Analysis* *3*
- *Design* *4*
  1. *Getting the input from the .txt file* 4
  2. *Create a new patient* 5
  3. *Delete an existing patient* 5
  4. *List all patients with their information* 6
  5. *Probability of having the disease* 7
  6. *Recommendation for patient* 8
  7. *Writing outputs to a .txt file* 8
- *Programmer's Catalogue* *9*
  1. *Description of program and functions* 9
     - **read_txtfile() function** 10
     - **create() function** 10
     - **remove() function** 10
     - **list_patients() function** 11
     - **prob_calc() function** 12
     - **probability() function** 13
     - **recommendation() function** 13
     - **output_function(out) function** 13
  2. *Time Spent On This Assignment* 14
  3. *Reusability of The Code* 14
- *User Catalogue* *15*
  1. *User manual* 15
  2. *Restrictions on the program* 15

- # Analysis

Although we are more prepared for cancer through prevention, early detection, and treatment, we still have limitations that may cause cancer to win against us such as overdiagnosis and overtreatment. These two misconceptions may cause serious negative effects in every aspect of a patient's life such as psychology, finance, and time.

To prevent them we have to write a utility that can help doctors with cancer diagnosis and treatment.

This utility should get information about patients and commands from a .txt file. It will make calculations and execute commands in the .txt input file and provide its outputs to an output .txt file.

To add a new patient to the system "create" command will be used, it will print a message if the patient already exists.

To delete a patient from the system "remove" command will be used, it will print a message if the patient does not exist on the system.

To list all the patients in the system "list" command will be used, it will print a left-aligned list with all the information of the patients. It will print an empty list if no patient exists.

To calculate the actual probability of the patient having the disease "probability" command will be used, it should use confusion matrix to calculate the percentage. It will print a message with the patient's name, probability, and disease name. If the patient doesn't exist on the system it will print a message.

To recommend a treatment to a patient "recommendation" command will be used, it will compare the actual probability and treatment risk of the patient and print a suggestion. If the patient doesn't exist on the system, it will print a message.

- # Design

The program should execute 5 types of commands with the inputs taken from the txt file.

1. Getting the input from the .txt file

In the problem, we are asked to get inputs from a .txt file. To do that, first input file should be opened then read line by line and append those lines to a list. We are going to use this list to separate commands and information. For every line in the file, lines should be split between commands and data using the space between them. Then, data should be split using ", " and put in a list "split_list". After that, commands in the input file should be checked. If it matches with a certain function, that function should be executed using data in the same line

.

2. *Create a new patient*

This command will add a new patient to the "patient_data_list" but first, it should be checked if the patient already exists on the system. We should search for the patient's name in "patient_data_list". If it exists, the program should print "Patient … cannot be recorded due to duplication.". Other case is adding the patient to the system. We should add the patient's data from the list we separated from the input file "split_list" to "patient_data_list" using list.append() and print "Patient … is recorded.". These steps will be executed by a function named "create()"

```
[['Hayriye', '0.999', 'Breast Cancer', '50/100000', 'Surgery', '0.40'], ['Deniz', '0.9999', 'Lung Cancer',
 '40/100000', 'Radiotherapy', '0.50'], ['Ateş', '0.99', 'Thyroid Cancer', '16/100000', 'Chemotherapy', '0.02'],
 ['Toprak', '0.98', 'Prostate Cancer', '21/100000', 'Hormonotherapy', '0.20'], ['Hypatia', '0.9975', 'Stomach
 Cancer', '15/100000', 'Immunotherapy', '0.04'], ['Pakiz', '0.9997', 'Colon Cancer', '14/100000', 'Targeted
 Therapy', '0.30'], ['Su', '0.98', 'Breast Cancer', '50/100000', 'Chemotherapy', '0.20']]
```

```
Patient Hayriye is recorded.
Patient Deniz is recorded.
Patient Ateş is recorded.
Patient Ateş cannot be recorded due to duplication.
```

3. *Delete an existing patient*

This command will remove the patient from the system. First, it should be checked if the patient exists on the system by searching their name in "patient_data_list". If there is no such patient, the program should print "Patient … cannot be removed due to absence.". On the other case, we should find the index of the patient's data and remove them from "patient_data_list" using list.pop(index) and print "Patient … is removed." These steps will be executed by a function named "remove()"

```
Patient Ateş is removed.
Patient Ateş cannot be removed due to absence.
```

4. List all patients with their information

This command will print a list of all the patients in the system. When printing the list, tabs should be used instead of spaces. First headings should be printed with the right amount of tabs between them. Then headings and patient data should be separated with a line of "-". This line should end at the end of the last heading word. Then in a loop, patient data should be printed with the right amount of tabs that make the list properly left-aligned.

The number of tabs to use should be controlled with conditions that check the length of the strings at the indexes 0,2,4 whether they're less or more than a certain length.

Some ratios in "patient_data_list" should be converted to percentages when printing to the list. To do that, their indexes should be found in the "patient_data_list" then these string variables should be converted to float type and they should be multiplied by 100.

These steps will be executed by a function named "list_patients()"

```
Patient Diagnosis   Disease         Disease     Treatment       Treatment
Name    Accuracy    Name            Incidence   Name            Risk
-------------------------------------------------------------------------
Hayriye 99.90%      Breast Cancer   50/100000   Surgery         40%
Deniz   99.99%      Lung Cancer     40/100000   Radiotherapy    50%
Ateş    99.00%      Thyroid Cancer  16/100000   Chemotherapy    2%
Toprak  98.00%      Prostate Cancer 21/100000   Hormonotherapy  20%
Hypatia 99.75%      Stomach Cancer  15/100000   Immunotherapy   4%
Pakiz   99.97%      Colon Cancer    14/100000   Targeted Therapy30%
```

5. Probability of having the disease

This command should calculate the actual probability of the patient having cancer. First. The patient's existence on the system should be checked. If the patient doesn't exist on the system "Probability for … cannot be calculated due to absence." should be printed. If they exist, the probability should be calculated according to confusion matrix using data which are "Diagnosis Accuracy" and "Disease Incidence". These will be taken from "patient_data_list" and converted to float to make the calculations. For calculations, prob_calc() function will be used. For existence and printing parts probability() function will be used.

Sick people = Numenator of disease incidence
Healthy people =Denominator of disease incidence - Numenator of disease incidence
True positives = Sick people * Diagnosis Accuracy
False Negatives = Healthy People * (1 – Diagnosis Accuracy)
Probability = (True positives / (True Positives + False Negatives)) * 100

"Patient Hayriye has a probability of "Probability" of having … cancer." will be printed if the patient exists on the system.

```
Patient Deniz has a probability of 80% of having lung cancer.
Patient Pakiz has a probability of 31.81% of having colon cancer.
Probability for Ateş cannot be calculated due to absence.
```

6. Recommendation for patient

This command will recommend whether to have the treatment or not depending on the treatment risk and the calculations done in the probability function.

First, we should check if the patient exists on the system. If there is no such patient "Recommendation for … cannot be calculated due to absence." will be printed.

If the patient exists then we have to compare the patient's probability of having the disease and treatment risk. If the treatment risk is higher, "System suggests … NOT to have the treatment." Will be printed. If the probability is higher, "System suggests … to have the treatment." will be printed.

```
System suggests Ateş NOT to have the treatment.
System suggests Hypatia to have the treatment.
Recommendation for Su cannot be calculated due to absence.
System suggests Su NOT to have the treatment.
```

7. Writing outputs to a .txt file

The output of the commands should be written in a .txt file. To do that, the program should create an output file in append mode and write the outputs to that file instead of printing them. For this purpose program has a function named "output_function(args)".

```
Patient Hayriye is recorded.
Patient Deniz is recorded.
Patient Ateş is recorded.
Patient Hayriye has a probability of 33.32% of having breast cancer.
System suggests Ateş NOT to have the treatment.
Patient Toprak is recorded.
Patient Hypatia is recorded.
System suggests Hypatia to have the treatment.
Patient Pakiz is recorded.
Patient Diagnosis   Disease         Disease     Treatment       Treatment
Name    Accuracy    Name            Incidence   Name            Risk
------------------------------------------------------------------------
Hayriye 99.90%      Breast Cancer   50/100000   Surgery         40%
Deniz   99.99%      Lung Cancer     40/100000   Radiotherapy    50%
Ateş    99.00%      Thyroid Cancer  16/100000   Chemotherapy    2%
Toprak  98.00%      Prostate Cancer 21/100000   Hormonotherapy  20%
Hypatia 99.75%      Stomach Cancer  15/100000   Immunotherapy   4%
Pakiz   99.97%      Colon Cancer    14/100000   Targeted Therapy30%
Patient Ateş is removed.
Probability for Ateş cannot be calculated due to absence.
Recommendation for Su cannot be calculated due to absence.
Patient Su is recorded.
System suggests Su NOT to have the treatment.
Patient Diagnosis   Disease         Disease     Treatment       Treatment
Name    Accuracy    Name            Incidence   Name            Risk
------------------------------------------------------------------------
Hayriye 99.90%      Breast Cancer   50/100000   Surgery         40%
Deniz   99.99%      Lung Cancer     40/100000   Radiotherapy    50%
Toprak  98.00%      Prostate Cancer 21/100000   Hormonotherapy  20%
Hypatia 99.75%      Stomach Cancer  15/100000   Immunotherapy   4%
Pakiz   99.97%      Colon Cancer    14/100000   Targeted Therapy30%
Su      98.00%      Breast Cancer   50/100000   Chemotherapy    20%
Patient Deniz has a probability of 80% of having lung cancer.
Patient Pakiz has a probability of 31.81% of having colon cancer.
```

- # Programmer's Catalogue

1. Description of program and functions

```
import os

current_dir_path = os.getcwd()
reading_file_name = "doctors_aid_inputs.txt"
reading_file_path = os.path.join(current_dir_path, reading_file_name)
writing_file_name = "doctors_aid_outputs.txt"
writing_file_path = os.path.join(current_dir_path, writing_file_name)

if os.path.exists(writing_file_path):
    os.remove(writing_file_path)
else:
    pass
```

The program locates the .txt files in this part. If there is an output file located in the directory it removes it because if we don't remove the file and rerun the program it will overwrite the file.

```
patient_data_list = []

for item in read_txtfile():
    function = item.split(" ", 1)[0]
    try:
        first_space = item.index(' ')
        split_list = item[first_space + 1:].split(", ")
    except ValueError:
        split_list = item.split()
    existence = any(split_list[0] in x for x in patient_data_list)
    for j in patient_data_list:
        if split_list[0] in j:
            place = patient_data_list.index(j)
    if function == "create":
        create()
    if function == "remove":
        remove()
    if function == "list":
        list_patients()
    if function == "probability":
        probability()
    if function == "recommendation":
        recommendation()
```

There is an empty list called "patient_data_list" to append new patients in the "create" part.
In for loop, we read the file line by line using "read_txtfile()".
The "function" variable gets a new command in every line by splitting the line in the file before the first space character and goes to the matching function using the if statements below.
Every line in the file except "list" lines has some information after the first space character so when the program reads that line it fails to locate a space character and it gives an error. To avoid that error, I used a try-except block. "existence" is a boolean variable that states if the patient exists on "patient_data_list". For block locates the patient in the "patient_data_list". They will be used in "remove()", "prob_calc()" and "recommendation()".

read_txtfile() function

```
def read_txtfile():
    with open(reading_file_path) as file:
        lines = file.read().splitlines()
        return lines
```

This function reads the input file line by line without the "\n" character and puts them in a list then returns the list.

create() function

```
def create():
    if split_list in patient_data_list
        output_function(f"Patient {split_list[0]} cannot be recorded due to
duplication.")
    else:
        patient_data_list.append(split_list)
        output_function(f"Patient {split_list[0]} is recorded.")
```

This function first checks if the patient exists on the system if the patient is already in the system it just executes "output_function()"
If the patient is not in the system it appends "split_list" to "patient_data_list" and executes "output_function()"
It creates a nested list with patient data in it.

remove() function

```
def remove():
    if existence is True:
        patient_data_list.pop(place)
        output_function(f"Patient {split_list[0]} is removed.")
    else:
        output_function(f"Patient {split_list[0]} cannot be removed due to
absence.")
```

This function checks if the patient is on the system by using the "existence" variable explained before in the report. If it is True, it removes the patient from the system. If it is false it just executes "output_function()"

list_patients() function

```
def list_patients():
    output_function("Patient"+"\t"+"Diagnosis"+"\t"+"Disease"+"\t\t\t"+"Disease"+"\t\t"+"Treatment"+"\t\t"+"Treatment")  # headings
    output_function("Name"+"\t"+"Accuracy"+"\t"+"Name"+"\t\t\t"+"Incidence"+"\t"+"Name"+"\t\t\t"+"Risk" + "\n" + "-"*73)  # headings
    for patient in patient_data_list:
        first_index = float(patient[1])*100  # turns diagnosis accuracy to percentage
        second_index = float(patient[5])*100  # turns treatment risk to percentage
        str_first = f"{first_index:.2f}"+"%"  # turn to a string with 2 decimal places and "%"
        str_second = f"{second_index:.0f}"+"%"  # turn to a string with 0 decimal places and "%"
        if len(patient[0]) < 4:  # sequences of conditions to make the list proper looking and left aligned
            if len(patient[2]) < 12:  # according to lengths of 0,2,4th indexes
                if len(patient[4]) < 8:
                    output_function(patient[0] + "\t\t" + str_first + "\t\t" + patient[2] + "\t\t" + patient[3] + "\t" + patient[4] + "\t\t\t" + str_second)
                elif len(patient[4]) < 12:
                    output_function(patient[0] + "\t\t" + str_first + "\t\t" + patient[2] + "\t\t" + patient[3] + "\t" + patient[4] + "\t\t" + str_second)
                elif len(patient[4]) == 16:
                    output_function(patient[0] + "\t\t" + str_first + "\t\t" + patient[2] + "\t\t" + patient[3] + "\t" + patient[4] + str_second)
                else:
                    output_function(patient[0] + "\t\t" + str_first + "\t\t" + patient[2] + "\t\t" + patient[3] + "\t" + patient[4] + "\t" + str_second)
            elif len(patient[2]) >= 12:
                if len(patient[4]) < 8:
                    output_function(patient[0] + "\t\t" + str_first + "\t\t" + patient[2] + "\t" + patient[3] + "\t" + patient[4] + "\t\t\t" + str_second)
                elif len(patient[4]) < 12:
                    output_function(patient[0] + "\t\t" + str_first + "\t\t" + patient[2] + "\t" + patient[3] + "\t" + patient[4] + "\t\t" + str_second)
                elif len(patient[4]) == 16:
                    output_function(patient[0] + "\t\t" + str_first + "\t\t" + patient[2] + "\t" + patient[3] + "\t" + patient[4] + str_second)
                else:
                    output_function(patient[0] + "\t\t" + str_first + "\t\t" + patient[2] + "\t" + patient[3] + "\t" + patient[4] + "\t" + str_second)
        elif len(patient[0]) >= 4:
            if len(patient[2]) < 12:
                if len(patient[4]) < 8:
                    output_function(patient[0] + "\t" + str_first + "\t\t" + patient[2] + "\t\t" + patient[3] + "\t" + patient[4] + "\t\t\t" + str_second)
                elif len(patient[4]) < 12:
                    output_function(patient[0] + "\t" + str_first + "\t\t" + patient[2] + "\t\t" + patient[3] + "\t" + patient[4] + "\t\t" + str_second)
                elif len(patient[4]) == 16:
                    output_function(patient[0] + "\t" + str_first + "\t\t" + patient[2] + "\t\t" + patient[3] + "\t" + patient[4] + str_second)
                else:
                    output_function(patient[0] + "\t" + str_first + "\t\t" + patient[2] + "\t\t" + patient[3] + "\t" + patient[4] + "\t" + str_second)
            elif len(patient[2]) >= 12:
                if len(patient[4]) < 8:
                    output_function(patient[0] + "\t" + str_first + "\t\t" + patient[2] + "\t" + patient[3] + "\t" + patient[4] + "\t\t\t" + str_second)
                elif len(patient[4]) < 12:
                    output_function(patient[0] + "\t" + str_first + "\t\t" + patient[2] + "\t" + patient[3] + "\t" + patient[4] + "\t\t" + str_second)
                elif len(patient[4]) == 16:
                    output_function(patient[0] + "\t" + str_first + "\t\t" + patient[2] + "\t" + patient[3] + "\t" + patient[4] + str_second)
                else:
                    output_function(patient[0] + "\t" + str_first + "\t\t" + patient[2] + "\t" + patient[3] + "\t" + patient[4] + "\t" + str_second)
```

This function first prints out the headings of the list and separates the headings from the patient data below.
In for loop, first, it turns the ratios of "Diagnosis Accuracy" and "Treatment Risk" into percentages by making them float numbers. Then string formats the variables to print them with desired decimal parts.
In nested if statements, lengths of 0,2,4th indexes are checked and the list is printed with a certain number of "\t" to make the list left aligned and proper.

prob_calc() function

```
def prob_calc():
    if existence is True:
        num_den = patient_data_list[place][3].split("/")
        fn = (float(num_den[1]) - float(num_den[0])) * (1 -
float(patient_data_list[place][1]))
        tp = float(num_den[0]) * float(patient_data_list[place][1])
        prob = tp/(fn+tp)*100
        return prob
    else:
        return None
```

This function calculates the probability of the patient having the disease.
If the patient exists on the system, numerator and denominator of "Disease Incidence" are separated with "/". When making calculations, we have to return them to floats from strings.
num_den[0] = numerator = people diagnosed with cancer
num_den[1] = denominator
num_den[1] - num_den[0] = healthy people
patient_data_list[place][1] = diagnosis accuracy
The "place" variable is the index of patient data in the "patient_data_list" mentioned before in the report.
people diagnosed with cancer * diagnosis accuracy = people who really have cancer among people diagnosed with cancer = tp
healthy people * (1 – diagnosis accuracy) = people who have cancer among cancer-free diagnosed people = fn
(tp/(fn+tp)) * 100 = probability of having the disease
returns the probability
If there is no such patient, returns None
This function will be used in "probability()" and "recommendation()" functions

probability() function

```
def probability():
    if prob_calc() is not None:
        if int(prob_calc()) == float(f"{prob_calc():.2f}"):
            output_function(f"Patient {patient_data_list[place][0]} has a probability of {prob_calc():.0f}% of having {(patient_data_list[place][2]).lower()}.")
        else:
            output_function(f"Patient {patient_data_list[place][0]} has a probability of {prob_calc():.2f}% of having {(patient_data_list[place][2]).lower()}.")
    else:
        output_function(f"Probability for {split_list[0]} cannot be calculated due to absence.")
```

If the "prob_calc()" function returned a value and If the value is xx.00, string format cuts the decimal parts. If it's not xx.00, string format prints the value with 2 decimal parts. Also, string format makes the treatment name's characters lowercase in the sentence.
If "prob_calc()" returned "None", the function prints a warning message.

recommendation() function

```
def recommendation():
    if existence is True:
        if prob_calc() > float(patient_data_list[place][5])*100:
            output_function(f"System suggests {patient_data_list[place][0]}
to have the treatment.")
        elif prob_calc() < float(patient_data_list[place][5])*100:
            output_function(f"System suggests {patient_data_list[place][0]}
NOT to have the treatment.")
    else:
        output_function(f"Recommendation for {split_list[0]} cannot be
calculated due to absence.")
```

patient_data_list[place][5] * 100 = Treatment Risk of the patient
If the patient exists, function compares treatment risk and probability of the patient having the disease.
Treatment risk > Probability = Suggest NOT to have the treatment
Treatment risk < Probability = Suggest having the treatment
If there is no such patient in the system, the function prints a warning message.

output_function(out) function

```
def output_function(out):
    with open(writing_file_path, "a") as outfile:
        outfile.write(args)
        outfile.write("\n")
```

This function writes the output of the other functions line by line to a file named "doctors_aid_outputs". I used output_function(args) for print() in the code.

2. Time Spent On This Assignment

Time spent on analysis	The analysis includes reading the Assignment2.pdf and attachments on Piazza platform carefully and understanding the data structures and functions I have to use in the assignment. Also, I spent a moderate time understanding confusion matrix to be able to calculate the probability of the patient having the disease. Overall, I can say understanding the problem took 3 hours.
Time spent on design and implementation	I spent a relatively long time on the design part of the assignment since I have little programming experience. Most of my time went to trying to figure out how to read the input file and splitting commands and data inside it. After I figured them out, other parts became easier since I had data and commands but aligning the list with tabs took a long time too since there are many possible scenarios. I can say this part took 15 hours for me.
Time spent on testing and reporting	In this part, I spent most of my time testing calculations I made in the prob_calc() function and list alignment with different scenarios. Also, I spent time formatting and testing decimal parts of numbers to make them look proper. Writing the report took a long time too, I tried to explain myself and the code as clearly as possible. This part approximately took 5 hours.

3. Reusability of The Code

The code of this program can be useful in many other scenarios.

Input, create and remove functions can be used to read files and create data lists. Slight modification might be required since the order of the file may be different.

The list function can be used with the desired number of tabs to list information in a good looking way.

The recommendation function might be helpful if there is a case of comparison.

- # User Catalogue

1. User manual

The user has to write all commands and data in a .txt file line by line and put the program and input file in the same directory. Then you can run the program in the terminal by entering the directory and writing:
python3 Assignment2.py
After that, you will get an output file in the same directory with your outputs in it.

2. Restrictions on the program

The user has to write all commands in lowercase characters.

Data placement for the "create" command is fixed
1. Name
2. Diagnosis Accuracy(float)
3. Disease Name
4. Disease incidence(ratio)
5. Treatment Name
6. Treatment Risk(float)

Input should look like this:

```
create Hayriye, 0.999, Breast Cancer, 50/100000, Surgery, 0.40
create Deniz, 0.9999, Lung Cancer, 40/100000, Radiotherapy, 0.50
create Ateş, 0.99, Thyroid Cancer, 16/100000, Chemotherapy, 0.02
probability Hayriye
recommendation Ateş
create Toprak, 0.98, Prostate Cancer, 21/100000, Hormonotherapy, 0.20
create Hypatia, 0.9975, Stomach Cancer, 15/100000, Immunotherapy, 0.04
recommendation Hypatia
create Pakiz, 0.9997, Colon Cancer, 14/100000, Targeted Therapy, 0.30
list
remove Ateş
probability Ateş
recommendation Su
create Su, 0.98, Breast Cancer, 50/100000, Chemotherapy, 0.20
recommendation Su
list
probability Deniz
probability Pakiz
```