TCSS 143A Winter 2019, Programming Assignment 1

Given the problem statement below, complete the following:

- ✓ Procedural program that is organized into multiple methods and runs correctly (70 pts). In order to receive credit, your code MUST compile and run, and complete at least some of the steps; comment out the code that shows your attempt but does not work
- ✓ Coding style and comments
  - Coding style (meaningful identifiers, indentation, etc.) and the header with your name and course info at the top of the file (10 pts)
  - Comments
    - Javadocs for every method and class (10 points)
    - Inline comments explaining code logic (10 points)

Programs that are not procedural and/or do not use multiple methods or use global variables will not receive a grade higher than 40 pts all together, even if a program runs perfectly, and the documentation is correct and thorough.

- ✓ Extra credit:
  - Detailed algorithm written in English / pseudocode (5 pts)
  - Test plan (5 pts)

You are NOT allowed to help one another with this program or use somebody else's code – check the rules listed on the syllabus. However, you may seek help from CSS mentors or a TA or an instructor.

Problem statement

For this program, you are to implement a simple machine-learning algorithm that uses a rule-based classifier to predict whether or not a particular patient has diabetes. In order to do so, you will need to first train your program, using a provided data set, to recognize a disease. Once a program is capable of doing it, you will run it on new data sets and predict the existence or absence of a disease. While solving this problem, you are to use a procedural approach in Java. Decompose your program into methods and use small lists where appropriate, i.e. you are only to use data types covered in chapters 1-6 and chapter 10.

Training

In order to train your program to recognize diabetes, you need to use a data set that contains both data and diagnoses (this is the learning phase). Your program is to ask the user to enter the name of the training file. You may assume that the file will be a txt file. Each line on that file constitutes one patient record. Each patient's record consists of 6 attributes and a numerical diagnosis. From our perspective, it does not matter much what these attributes signify. The only important observation here is that the attributes are floating point numbers delimited by tabs. If an attribute is missing, a question mark replaces a number. In your program, treat question marks as zeros. As far as diagnosis is concerned (the 7^{th} value listed on a line), any value >= 0.65 indicates a presence of a disease, while a value < 0.65 indicates its absence. You may assume that all data in the file will be valid, i.e. there is no need to handle user error or file errors. Sample file `train.txt` is provided

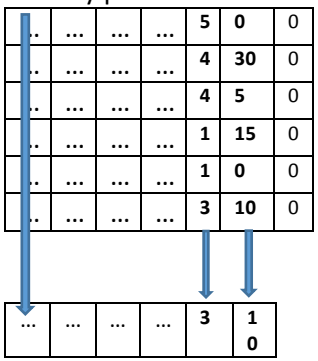
Here is a sample line from the input file:

```
25     0      175.6        101.14       159.4        3      0
```

Since the last element is a zero, this patient has no diabetes. In case you are wondering, the first 6 attributes are: age, sex, random blood sugar level, fasting blood sugar level, oral glucose tolerance level, hemoglobin A1c.

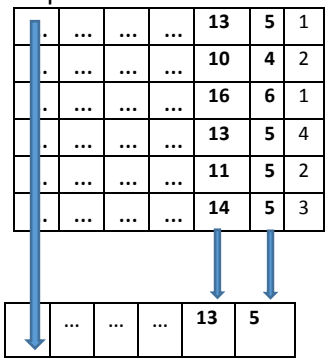
Based on the input data that follows this format, the program is to create a classifier that will be used later on to process new patients and make diagnostic predictions for these new patients. The classifier is created in the following fashion. For all patients with no diabetes, for each attribute, we calculate the average value of that attribute. For all patients with diabetes, for each attribute, we calculate the average value of that attribute. After processing all the patients, we end up with two sets of averages – one set for healthy patients (use a list) and one set for ill patients (use a list). In short, as you process a line, you need to first determine whether you are dealing with a healthy or ill patient and update your averages accordingly. The picture below illustrates the idea (one row in either table is one line in a file).

Healthy patients



Average of healthy patients

Ill patients

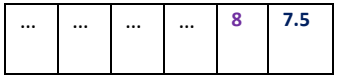


Average of ill patients

Once you have these two averages calculated, you need to find a midpoint (mean) between them – these are called the class separation values.

Average of healthy patients

Average of ill patients



Class separation values

Your program needs to print the averages of healthy patients, ill patients, and class separation values to the screen.
This is the sample run of the beginning of the program (blue highlighting indicates user input):

```
Enter the name of the training set file: train.txt
Healthy patients' averages:
[37.50,0.00,182.80,107.79,164.47,3.65]
Ill patients' averages:
[56.50,1.00,255.72,134.95,277.06,8.15]
Separator values:
[47.00,0.50,219.26,121.37,220.76,5.90]
```

Run Classifier

Once you know class separation values, you can run your program on new sets and make predictions for new patients that haven't been diagnosed yet. This is where the second input file comes in. The file will be in a csv format and each line in that file will model one patient: his/her id and 6 attributes described above. The attributes will be in the same exact order as in the training file and will be separated by commas. You may assume that all data in the file will be valid, i.e. there is no need to handle user error or file errors. Sample file `set1.csv` is provided.
For each of these patients, you need to compare his attributes to the separator values. If a patient's attribute has a higher value than the corresponding separator value, then the attribute needs to be labeled as "at risk". Overall, if a patient has more than half "at risk" attributes, the patient is classified as "ill". In the picture below, we are dealing with an "ill" patient since this patient is above the norm on more than half of the attributes.

47	0.5	219.26	121.37	220.76	5.9

Class separator values

63	**1**	**230.45**	**127.9**	203	4.2

Patient values

Once you determine if a patient should be diagnosed as having diabetes, you need to print that patient's id and diagnosis to `results.csv` (sample file provided):

- use 1, if 4 attributes are above average
- use 2, if 5 attributes are above average
- use 3, if all attributes are above average

Finally, print to the screen how many patients you processed, how many received diagnosis codes 1, 2, and 3.

This is the sample run of the next phase of the program (blue highlighting indicates user input):

```
Enter the name of file containing patient data: set1.csv
Number of patients:        303
Diagnosis code 1:           79
Diagnosis code 2:            6
Diagnosis code 3:            0
Check results.csv for specific patients and their codes
```

Program Submission

The source code is to be called `Project1.java`

On or before the due date, use the link posted in Canvas next to Project 1 to submit your code and all associated documentation. Make sure you know how to do it before the due date since late assignments will not be accepted. Valid documentation file formats are: doc, docx, rtf, pdf.

TEST PLAN

Reason for test case	Input values	Expected output
class separator values calculation	Healthy patients' averages: [37.50,0.00,182.80,107.79,164.47,3.65] Ill patients' averages: [56.50,1.00,255.72,134.95,277.06,8.15]	[47.00,0.50,219.26,121.37,220.76,5.90]