TCSS 142 Programming Assignment 2
Due: see Canvas for due date and time

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.

The entire solution needs to be decomposed into several functions where data that needs to be passed between main and other functions should be passed as arguments to the function parameters. Keep in mind, primitive data such as integers, floats, characters, etc. are pass by value where any changes to the parameters are not returned to the calling function. Only data stored in lists can be changed inside a function with those changes being returned to the calling function through the parameters. Of course, functions may, at times, pass information back to main through a return statement.

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 a coronary heart disease. 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.

<u>You may not use Python constructs that have not been discussed in class so far.</u>

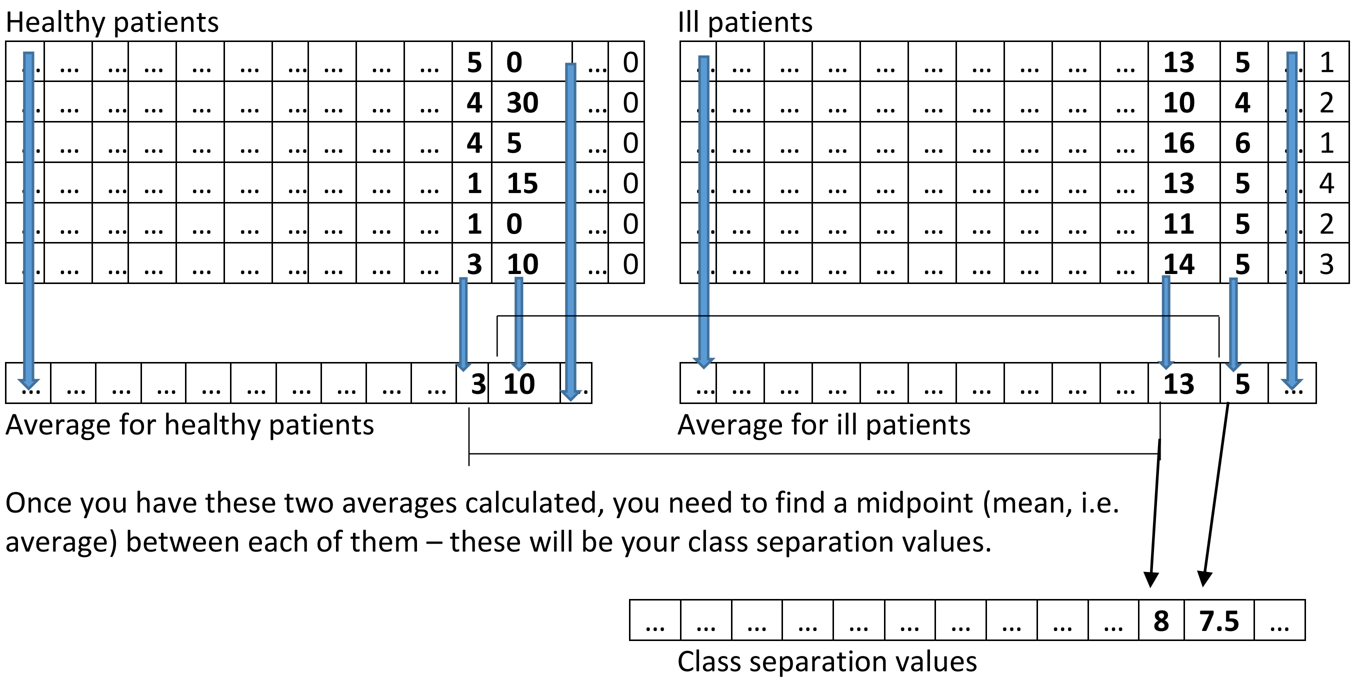
Training

In order to train your program to recognize heart disease, you need to use a data set that contains both data and diagnoses (this is the learning phase). Your program is to use (train.csv) for the training file (use train.csv for testing and create your own test files based on that file). Each line on that file constitutes one patient's record. Each patient's record consists of 13 attributes **and** a numerical diagnosis (14 values in all). From our perspective, what these attributes represent does not matter. The only important observation here is that the attributes are floating point numbers delimited by commas. If an attribute is missing, a question mark replaces a number. Since we do not know a value for a question mark, that attribute will not be included into the running total. More simply, a question mark has a value of 0. As far as diagnosis (the 14th value on a line) is concerned, any value > 0 indicates a presence of a disease, while 0 indicates its absence.

Here is a sample line:
63.0,1.0,1.0,145.0,233.0,1.0,2.0,150.0,0.0,2.3,3.0,0.0,6.0,0

Since the last element is a zero, this patient has no heart disease. In case you are wondering, the first 13 attributes are: age, gender, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise induced angina, segment depression, slope of peak segment depression, number of major vessels, and defect type.

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 heart disease, for each attribute, we calculate the average value of each attribute. For all patients with heart disease, for each attribute, we calculate the average value of each attribute. After processing all the patients, we end up with two sets of averages – one set for healthy patients and one set for ill patients. 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).



Once you have these two averages calculated, you need to find a midpoint (mean, i.e. average) between each of them – these will be your class separation values.

<u>I strongly suggest you have the above working by the end of week 1 of the two weeks you have to complete the project.   E.G, at the end of week 1 the supplied train.csv file should produce the following output:</u>

```
Total Lines Processed: 303
Total Healthy Count: 164
Total Ill Count: 139
Averages of Healthy Patients:
52.59, 0.56, 2.79, 129.25, 242.64, 0.14, 0.84, 158.38, 0.14, 0.59, 1.41, 0.27, 3.77
Averages of Ill Patients:
56.63, 0.82, 3.59, 134.57, 251.47, 0.16, 1.17, 139.26, 0.55, 1.57, 1.83, 1.13, 5.80
Seperation Values are:
54.61, 0.69, 3.19, 131.91, 247.06, 0.15, 1.00, 148.82, 0.34, 1.08, 1.62, 0.70, 4.79
```

Get the above output by week 1's end and then complete the following by the end of week 2:

Run Classifier

Once you know class separation values, you can run your program on new sets and make the predictions. One such set is provided – it is called cleveland.csv. In this data set, a patient id and 13 attributes are provided for each patient. For each of these patients, you need to compare the 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 counted as "at risk", i.e. only increment the counter when the patient is "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.

Class separator values:

1	2	121	3	14	3.5	1	2	4	5	8	7.5	121

Patient's attributes:

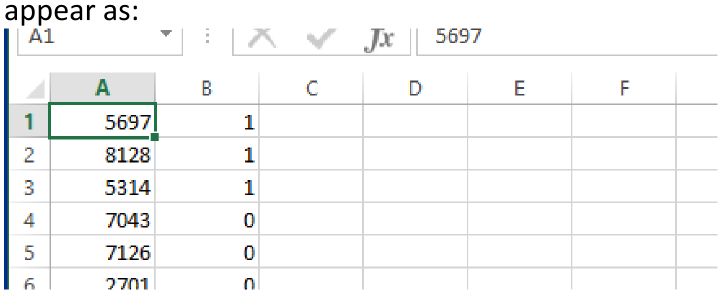
0	**3**	**143**	**5**	**15**	3	1	**3**	**5**	5	8	**8.7**	100

Model Accuracy
One interesting question to ask regarding this model is the accuracy of its predictions. In order to determine accuracy of the model, you need to train the program as described above, but before you run it on a new file, you run it on the training file again. You treat the training file as if the diagnosis column were missing and have your program predict whether a patient is healthy or not. Then, you compare your program's diagnosis to the actual diagnosis appearing in the file. If all program predictions are accurate, then your accuracy is 100%, if a file contained 10 patients and 8 of your diagnosis are correct, then accuracy is 80%, etc. To accomplish this, you will need to count the number of correct diagnosis and when finished, divide that count by the total number of patients. Remember that your program is only concerned with a presence or absence of a disease, and not with the degree of a disease (**values of 1-4 are the same to us, as they indicate the presence of a disease**)

Output
Your program is to generate statistics to a screen and to an output file. The following information needs to be printed to the screen, based on the training set file:

- Total Line
- Healthy count
- Ill count
- healthy patients averages
- ill patients averages
- separator values
- accuracy of the model

In addition, your program is to produce the results of using the classifier on the new data set (cleveland.csv) and to write them to the output csv file named (clevelanddiag.csv – **Do NOT alter this file name. Also, DO NOT import csv and use csv.writer for your output**). The file should contain one column listing the patient's id and the second column listing the diagnosis (**use 0 for healthy and 1 for ill**). **Make sure both the ID and diagnosis are integers**, **NOT** strings or Boolean (True/False). For example, a few lines in this output file will appear as:



A1 | 5697

	A	B	C	D	E	F
1	5697	1				
2	8128	1				
3	5314	1				
4	7043	0				
5	7126	0				
6	2701	0				

Sample output file is provided on Canvas as well.

Sample program run (training set: train.csv):

```
Total Lines Processed: 303
Total Healthy Count: 164
Total Ill Count: 139
Averages of Healthy Patients:
52.59, 0.56, 2.79, 129.25, 242.64, 0.14, 0.84, 158.38, 0.14, 0.59, 1.41, 0.27, 3.77
Averages of Ill Patients:
56.63, 0.82, 3.59, 134.57, 251.47, 0.16, 1.17, 139.26, 0.55, 1.57, 1.83, 1.13, 5.80
Seperation Values are:
54.61, 0.69, 3.19, 131.91, 247.06, 0.15, 1.00, 148.82, 0.34, 1.08, 1.62, 0.70, 4.79
```

test set: cleveland.csv should produce the output file “clevelanddiag.csv.”

Program Submission
If you want your assignment to be graded, it has to be compatible with our platform, namely Python 3.4.1 or higher. The source code is to be called yourNetid_project2.py
On or before the due date, use the link posted in Canvas next to Programming Assignment 2 to submit your code. Make sure you know how to do that before the due date since late assignments will not be accepted.
To help get you started, I strongly suggest you use pseudocode and the top down/refinement techniques presented in class. Consider the basic operations that main needs to perform to generate the final results. Once determined, refine each step into specific details continuing to use pseudocode. Each general step should result as a function call from main. The refinement of each step would be the detailed steps within each function. Generally, at the top level, what will become steps in main, you will want to do the following:

open input file - train.csv
setup counters (healthy and ill counters) and accumulators (healthy and ill totals as lists)

As you will be calling a function to acquire and return the counters and healthy/ill totals lists, the lists can be updated in a function with the updates returning through the parameters. However, the counters individually will not return with their updated values. Python does allow multiple values to be returned in return statements but, most languages do not provide such a mechanism. For this reason, the counters could be implemented in a two element list where the element at index 0 represents healthy counts and the element at index 1 represents ill counts. This two element counter list can then be passed to a function which will update their values and returned through the parameter.

getHealthyIllTotals – pass the two lists (healthy and ill), the counters list, and the opened input file
close input file
get the totalPatients by adding the two counters

displayCounts – pass the counters and the totalPatients count

get healthy and ill averages – use the same function for each. For example:
healthyAverages = getAverages(healthyTotals, healthyIllCounts[0])
illAverages = getAverages(illTotals, healthyIllCounts[1])

get the separation values. For example:
sepValues = getSeparationValues(healthyAverages, illAverages)

Display the 3 lists on the console. Use a single function that receives a list and a message. For example:
displayList(healthyAverages, "Healthy patients' averages:")
displayList(illAverages, "Ill patients' averages:")
displayList(sepValues, "Separation Values:")

Check your accuracy by making diagnosis on the same input file
Open input file - train.csv
Check accuracy- return the total of correct diagnosis. For example:
diagCnt = checkAccuracy(in1, sepValues)
Close input file

Display your accuracy – i.e diagCnt / totalPatients

Make diagnosis on new input file "cleveland.csv" and write findings to output file "clevelanddiag.csv"

For example:

```
in1 = open("cleveland.csv", "r")
out1 = open("clevelanddiag.csv", "w")
writeDiagnosis(in1, out1, sepValues)
```

Close input and output files

Refine the details of each function using a similar process as above.

To avoid complexity issues, I strongly suggest getting your program to run correctly up to (but not including) the check for accuracy, i.e. the display of the three lists (healthy averages, ill averages, separation values).

Once successful with the processing of the three lists, add the accuracy check function and run your program to assure you get an accuracy of 0.80 (80%).

Finally, write the diagnosis function for a new file.