Project 3: Diagnostics

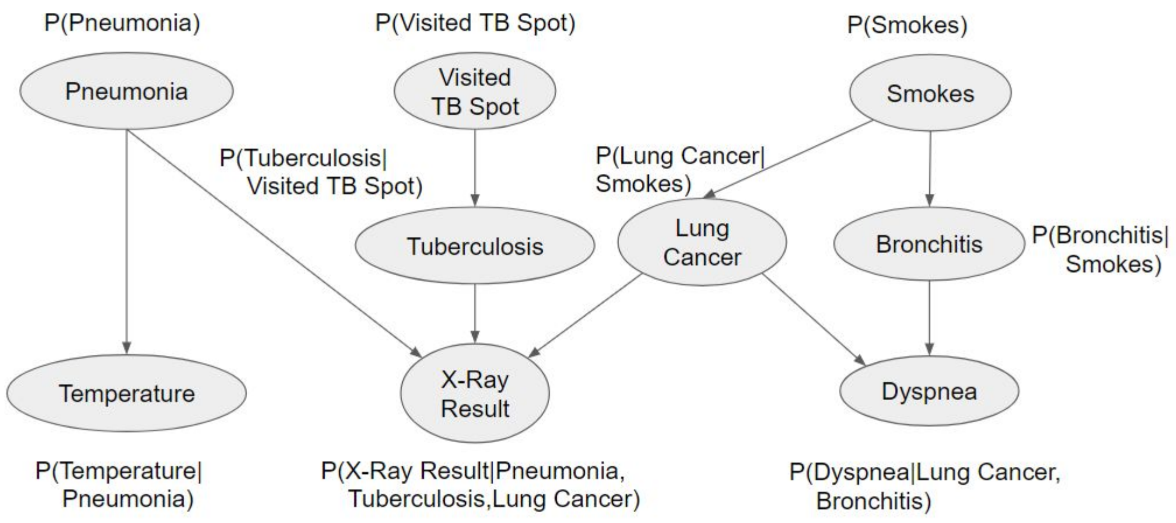
Artificial Intelligence (IDL340)

Basics

- This project should be performed alone or in pairs.
- You will use the R programming language.
- The use of third party code libraries/R packages is not permitted.
- The required code is available in the Diagnostics_1.0.0.zip file on student portal.
- See the R FAQ document on student portal for questions about using R.

Project Overview

You will implement a Bayesian network that will provide estimations of the probability that a person has Pneumonia, Tuberculosis, Lung Cancer and Bronchitis given a set of observed variables. See the diagram below for other variables, and also for the causal structure that your network should have.



All variables except for Temperature are binary (0=False/1=True), and Temperature is normally distributed.

Your task is to set up this Bayesian network, train it from a set of 10000 historical cases and be able to use it to estimate the probability of the four illness variables when all other variables are observed using Metropolis in Gibbs MCMC sampling. This will mean creating one function that trains the network from the historical cases, and one function that is used to predict the probabilities of illness variables given observed values of the other variables.

You will provide

An R script with two function that can be passed to a function of the same form as the runDiagnostics function.

Pass Requirements

Your function needs to:

- Equal or surpass par performance
- Meet the execution time requirement

See documentation on the runDiagnostics function in the Diagnostics package for details.

Important Functions

The important functions in the WheresCroc package are:

Function Name	**Description**
runDiagnostics	When you pass a ‘learn’ and ‘diagnose’ function, this function calls the passed learn function with the 10000 training cases and expects a network to be returned. It then calls the passed diagnose function to get estimates of the probabilities of the illness variables for the 10 sample test cases. It will also show you error comparisons with other cases (ideal, the par performance model, etc). The help documentation contains important information required for completing this project. The output of your diagnose function should be in the form of a 10x4 matrix giving the probabilities of the four illness variables being true in each of the 10 cases. The columns are ordered in the same order as the illness variables appear in the datasets. The documentation states that your learn function should return ‘a network object’. This object is never used by any code except your own: It is immediately passed to your diagnose function. Accordingly, the form of this object is entirely up to you, and it is only called a network object because it should of course encode the information in

	the Bayesian network you train from the historic data.

Datasets

There are two datasets included in the project package: 10000 historic data cases for training and 10 test cases for testing. The training cases included values for all variables, the test cases have unknown (NA) values for the four illness variables. They should be automatically lazily loaded, so when you load the Diagnostics package (using library) they will be available (just type their name in the console). If they are not lazily loaded on your system, you can load them using the data function (after loading the package).

hist data(hist)	The 10000 training cases.
cases data(cases)	The 10 sample test cases.

Dangers

Make sure you include a burn period in your implementation of the Metropolis in Gibbs MCMC sampling algorithm. The par function uses a burn of 100 for a sample size of 1000.

Advice

You might also find it speeds things up to pre-generate lots of random numbers and store them in a vector. You can then access them by an index (which you increase each time you fetch a random number. This may be considerably faster than many calls to runif, each generating a single random number.

Competition Hints (how to improve on the par function)

Basically, if you implement the algorithm correctly, a lot is going to come down to chance based on the samples generated when we execute your code. But since everyone has the same historic data, same test cases and same random seed, even this randomness is likely to disappear.

The only obvious way to improve the performance of your network is to increase the number of samples taken, but as the comparisons shown in the runDiagnostics function indicate, this does not help a great deal. Moreover, since the sampling is very time consuming it will not be possible to increase them by much without hitting the time limit (and you want a nice amount of leeway there). So be very careful about choosing to increase the number of samples taken!

The use of ML fitting techniques appeared to do better than Bayesian approaches (again, see the comparisons shown in the runDiagnostics function). But with only 10 test cases you should not read much into this.

Remember: Implement the basic algorithm and ensure you have a passing implementation before trying to be ambitious!