732A96/TDDE15 ADVANCED MACHINE LEARNING

LAB 1: GRAPHICAL MODELS

JOSE M. PEÑA
IDA, LINKÖPING UNIVERSITY, SWEDEN

1. INSTRUCTIONS

- **Deadline for individual and group reports**
  See LISAM.
- **What and how to hand in**
  Each student must send a report with his/her solutions to the lab. Submission is done by using the functionality 'Submit' of the respective lab in LISAM/Submissions. See above for the submission deadline. The file should be named Name_LastName.pdf. The report must be concise but complete. It should include (i) the code implemented or the calls made to existing functions, (ii) the results of such code or calls, and (iii) explanations for (i) and (ii).
  In addition, students must discuss their lab solutions in a group. Check LISAM for the groups. Each group must compile a collaborative report that will be used for presentation at the seminar. The report should clearly state the names of the students that participated in its compilation and a short description of how each student contributed to the report. This report should be submitted to LISAM by using the functionality 'Submit' of the respective group lab submission in LISAM/Submissions. See above for the submission deadline. The file should be named Group_X.pdf where X is the group number. Please, upload a copy of the group report to the folder LISAM/Collaborative workspace. The collaborative reports are corrected and graded. The individual reports are also checked, but feedback on them will not be given. A student passes the lab if the group report passes the seminar and the individual report has reasonable quality, otherwise the student must complete his/her individual report by correcting the mistakes in it.
  Attendance to the seminar is obligatory. In the seminar, some groups will be responsible for presenting their group reports. Each student in these groups must be prepared to individually present an arbitrary part of the report. The selection of the speakers is done randomly during the seminar. In the seminar, some groups will act as opponents to the reports provided by the presenters. The opponent group should examine the group report of the presenter group before the seminar (available at LISAM/Collaborative workspace), and prepare a minimum of three questions, comments and/or improvements. The opponent group will ask these questions during the seminar. Check LISAM for the list of presenter and opponent groups.
- **Resources**
  The lab is designed to be solved with the R packages bnlearn and gRain. Students may also want to use the RStudio development environment.
  - Literature:
    - Package documentation.
    - Højsgaard, S. Graphical Independence Networks with the gRain Package for R. *Journal of Statistical Software* 46, 2012.
    - Scutari, S. Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software* 35, 2010.
  - Hint: Spend the first hour exploring the site `www.bnlearn.com`. Try the code in `www.bnlearn.com/examples`.

2. Questions

The purpose of the lab is to put in practice some of the concepts covered in the lectures. You can use any data set you like, e.g. you own data, data from public repositories, or data included in the R packages bnlearn and gRain. Check for instance `www.bnlearn.com/documentation`. The learning.test, asia or alarm data sets should suffice. Some questions may be easier to solve with one data set than with the others and, thus, you may need to try with different data sets.

(1) Show that multiple runs of the hill-climbing algorithm can return non-equivalent DAGs. Explain why this happens.
    Hint: Check the function hc in the bnlearn package. Note that you can specify the initial structure, the number of random restarts, the score, and the equivalent sample size (a.k.a imaginary sample size) in the BDeu score. Use these options to answer the question. You may also want to use the functions plot, arcs, vstructs, cpdag and all.equal.

(2) Show that increasing the equivalent sample size (a.k.a imaginary sample size) in the BDeu score decreases regularization. Explain why this happens.
    Hint: Run some structure learning algorithm (e.g. check the function hc in the bnlearn package) multiple times and show that it tends to end in more densely connected DAGs when large imaginary sample sizes are used. Or produce a histogram of the scores of a random sample of DAGs with different imaginary sample sizes and see if they come closer or not one to another (e.g. check the functions hist, random.graph, sapply and score in the bnlearn and core packages).

(3) You already know the LS algorithm for inference in BNs, which is an exact algorithm. There are also approximate algorithms for when the exact ones are too demanding computationally. Compare the answers given to some queries by exact and approximate inference algorithms. When running the approximate algorithm several times, why may we obtain different answers ? Is the approximate algorithm equally accurate when the query includes no observed nodes and when it includes some observed nodes ?
    Hint: For exact inference, you may need the functions bn.fit and as.grain from the bnlearn package, and the functions compile, setFinding and querygrain from the package gRain. For approximate inference, you may need the functions prop.table, table and cpdist from the bnlearn package.
    When you try to load the package gRain, you will get an error as the package RBGL cannot be found. You have to install this package by running the following two commands (answer no to any offer to update packages):
    source("https://bioconductor.org/biocLite.R")
    biocLite("RBGL")

(4) There are 29281 DAGs with five nodes. Compute approximately the fraction of the 29281 DAGs that represent different independence models. In the light of the result obtained, would you say that it is preferable to perform structure learning in the space of DAGs or in the space of essential graphs ?
    Hint: You do not need to produce the 29281 DAGs. Instead you can sample DAGs uniformly, convert each DAG in the sample to its essential graph (a.k.a completed partial DAG or CPDAG), and then count how many different essential graphs you have left. You can do all this with the functions random.graph, cpdag, lapply, unique and length in the bnlearn and core packages. Try different values for the parameters every and burn.in in the function random.graph.