732A96 ADVANCED MACHINE LEARNING

LAB 1: PROBABILISTIC GRAPHICAL MODELS

JOSE M. PEÑA
IDA, LINKÖPING UNIVERSITY, SWEDEN

1. Information

- Deadline: 12/09-16 at 9 a.m.
- Grade: Pass/Incomplete.
- What to hand in: A concise but complete report answering the questions below. This should include (i) the code you implemented or the calls you made to existing functions, (ii) the results of such code or calls, and (iii) explanations for (i) and (ii). You can use any programming language, toolbox or GUI software you like, e.g. BNToolbox in Matlab, GUI software such as Hugin, Genie, SamIam, or R packages such as bnlearn and gRain. I recommend to use the R packages though. I also recommend that you use the RStudio development environment.
- How to hand in: By e-mail to jose.m.pena@liu.se with message header 'AML: PGM Lab'.
- Tips: 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, data included in the software of your choice, etc. Check for instance `www.bnlearn.com/documentation`. The learning.test, asia and 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 hill-climbing algorithm can return non-equivalent DAGs. Can we check if two DAGs are equivalent by checking if they receive the same score ?
Tips: Check the function hc in the bnlearn package. Note that you can specify the initial structure as well as the number of random restarts. Use these options to answer the question. You may also want to use the functions plot, arcs, vstructs, cpdag and all.equal.

(2) Consider any DAG obtained in the first question. Modify it by performing a sequence of covered arc reversals. Show that the DAG resulting from each reversal is equivalent to the original one.
Tips: You may want to use the functions plot, set.arc, drop.arc and reverse.arc in the bnlearn package.

(3) Show that increasing the equivalent sample size (a.k.a imaginary sample size) in the BDeu score decreases regularization. Explain why this happens.
Tips: 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. 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). You can also can compare the DAGs learnt with different imaginary sample sizes and the true DAG available at `www.bnlearn.com/documentation`.

(4) 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 many observerd nodes ?

Tips: 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 cpquery from the bnlearn package.

(5) There are 29281 DAGs with five nodes. Compute approximately the fraction of the 29281 DAGs that represent different independence models. What are the advantages and disadvantages of to perform structure learning in the space of DAGs compared to performing it in the space of independence models (e.g. representing the latter with essential graphs) ?

Tips: 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.