Project 1

Bayesian Structure Learning

Due Date: by 5 pm on Friday, 17 October. Penalty-free grace period until 5 pm on Monday, 21 October. See "Late Policy" for details.

This project is a competition to find Bayesian network structures that best fit some given data. The fitness of the structures will be measured by the Bayesian score (described in the course textbook *Algorithms for Decision Making* section 5.1, or the older textbook DMU section 2.4.1).

Three CSV-formatted datasets have been provided in `AA228-CS238-Student/project1/data/`. The first row indicates variable names. These datasets are taken from titanic, wine and a secret black box, respectively. We have discretized the data so that you only have to deal with discrete variables in this assignment.

1. small.csv *8 variables*
2. medium.csv *13 variables*
3. large.csv *50 variables*

The files can be accessed from the `AA228-CS238-Student` repository, which also includes starter code: https://github.com/sisl/AA228-CS238-Student/

You will try to find the structure for each dataset yielding the highest Bayesian score. The student receiving the highest score will win the competition. The competition results will be posted on the course website after the due date.

Rules

- Your program should output a file containing the network structure. The output filename should be the same as the input filename, but with a `.gph` extension, e.g., `small.gph`.
- A generic example `example.gph` is provided to you in `AA228-CS238-Student/project1/example/`.
- A specific example of a graph for Titanic dataset with only 3 edges (numsiblings → numparentschildren, numsiblings → passengerclass, numparentschildren → sex) will look like `titanicexample.gph` provided in `AA228-CS238-Student/project1/example/`.
- You can use any programming language but you cannot use any package directly related to structure learning. You can use general optimization packages so long as you discuss what you use in your writeup and make it clear how it is used in your code. Recommended packages:
  - `Graphs.jl` for Julia
  - `NetworkX` for Python
  - For reading in the CSV files, you can use `DataFrames.jl` for Julia and `Pandas` for Python
- Discussions are encouraged, and we expect similar approaches to the project from multiple people, but you must write your own code. Otherwise, it violates the Stanford Honor Code.
- Submit a `README.pdf` describing your strategy. This should not be more than 1 or 2 pages (excluding your code) with a description of your algorithm, the time taken for each graph, and the graph plots (with plots *not* counting towards page limit). Only brief explanations are necessary. Also, please typeset your code and include it in the PDF (note, code does *not* count towards your page limit).
- **Grading Rubric:**
  - Small Graph (`small.gph`) – 10%
  - Medium Graph (`medium.gph`) – 20%
  - Large Graph (`large.gph`) – 30%
  - README.pdf – 40%
    - Description of algorithm – 10%
    - Running time for each problem – 10%
    - Visualization of each graph – 10%
    - Code (included in PDF) – 10%

Supplementary Bayesian Score Tutorial – A previous TA put together a more detailed walkthrough for computing the Bayesian score that may be useful. The video is available here.

LaTeX Template

We provide an *optional* LaTeX template on Overleaf for your README.pdf write-up. Note you're free to use your own template (and you're not even required to use LaTeX).

- Click the template link, click "Menu", and "Copy Project" (make sure you're signed into Overleaf)

Submission

- Submit your `.gph` files via Gradescope under the **Project 1 (.gph files)** assignment.
- Submit your `README.pdf` via Gradescope under the **Project 1 (README.pdf)** assignment.

Submission Video Tutorial – A previous TA put together a quick video tutorial explaining the repository and how to submit to Gradescope.

FAQs

This list continuously grows to reflect common queries made on Ed. You may find your query answered here without even needing to wait on Ed!

- **What programming languages can I use?**
  - You may use any language you like! We are only looking for your source code and the output `.gph` files

- You may use any language you like! We are only looking for your source code and the output `.gph` files.

- **I like the competition and leaderboard aspect. Can we use late days for the competition?**
  - No. You can only use late days for the general project grading. Any submissions after the deadline will not be considered for the leaderboard.
- **Can we use the `bayesian_score` function in `BayesNets.jl`?**
  - No. You can't use any structure learning related packages, so you'll have to implement your own score function.
- **What's the higher Bayesian score value: -2345.6 or -3456.7?**
  - -2345.6
- **What priors are we using?**
  - We are using a Uniform Dirichlet Prior (all pseudo-counts $\alpha_{ijk} = 1$).
- **Can you please explain what's in the CSV file?**
  - The header line in the CSV file gives you the names of all the nodes of the graph. You'll use them for creating your `.gph` file. Each row of the CSV file represents a sample from the graph, i.e. the value for each discrete variable. Different variables might have a different number of discrete outcomes. That number is determined by the maximum value for that variable found in the dataset, and the minimum value is 1 for all variables. More explicitly, if the variable takes on values 1, 2, and 5 in the dataset, then the variable has 5 different discrete outcomes.
- **Can we make multiple submissions?**
  - YES! But remember, your last submission will be scored and show up on the leaderboard.
- **Can you point us to a survey of structure learning algorithms?**
  - YES! See Structure Learning of Probabilistic Graphical Models: A Comprehensive Survey.
- **Do you have some general advice for the competition?**
  - This competition boils down to combining various algorithms and strategies, including algorithms outside of the textbook, while making your code efficient and tuning any hyper-parameters for optimal performance.
- **Are there any runtime constraints on the code submitted for project 1?**
  - No, there are no runtime constraints. As long as there is a reasonable attempt for the solution, we expect to give you full credit. Also, you are welcome to use whatever resources you have access to. You should submit code that we could run (but we won't necessarily run it). If you want to run a long time and get an extra good graph, that's fine. You will need to report how long you ran your code in your write-up.
- **Do you check for cyclic graphs?**
  - Yes, our tester script checks for that.
- **What is the grading criteria, a.k.a. what do I need to do to get full credit?**
  - You have to implement your own scoring function.
  - You need to provide a graph that performs better than a baseline (comparing the Bayesian scores). Any correctly implemented structure learning algorithm should easily beat the baseline.
  - If you implement some structure learning algorithm or a variant, then you'll get full credit — so long as you fulfill the other requirements on the write-up.
- **Can we submit multiple code files with different algorithm implementations?**
  - Yes. Please mention how the graphs compare to each other in your README, and to ensure the best chances in the competition, please make sure that the better performing graphs are most recently submitted through `submit`.
- **Do we need a specific name for code files, like we do for README and solution files?**
  - No specific file name needs to be used. However, making title names clear and mentioning them in your write-up is super helpful for the grader!
- **Does the Bayesian score computed through `submit` include the $\ln P(G)$ term?**
  - No, it does not.
- **What does `idx2names` mean in the `write_gph` method?**
  - `idx2names` is the ordering of the node names that you use. Basically, a dictionary that can map the node index to the node name.
- **How do I convert linear indices to subscripts and vice versa?**
  - For Julia, use **`CartesianIndices`** and **`LinearIndices`**. For Python, use **`numpy.unravel_index`** and **`numpy.ravel_multi_index`**. For MATLAB, use **`ind2sub`** and **`sub2ind`**.
  - One of our previous TAs (Robert Moss) put together a detailed notebook illustrating how to use each of these functions: Notebook on Subscript and Linear Indexing
- **How do I plot the graphs?**
  - For Python, `NetworkX` has draw functions. NetworkX also has a function `write_dot`, which would allow you to use GraphViz to generate the plots: `dot -Tpng input.dot > output.png`.
  - For Julia, you can use `TikzGraphs.jl`, `GraphPlot.jl`, or `GraphRecipes`. Examples are provided in the AA228-CS238-Student repository.
- **How do I typeset my code?**
  - If you're using $\LaTeX$, you can use the `verbatim` environment as a simple approach or the `listings` environment for syntax highlighting (or `pythontex` if you're feeling fancy).
- **Can I consult outside sources for help?**
  - For projects, you may consult any material, including books, the textbook, online resources, classmates, and AI tools (e.g., ChatGPT), for **guidance and advice only**. All code must be **written and typed up by you**; do not copy and paste code from others or external sources