Project 2: Bayesian Networks

Deadline: 2018-11-15

Abstract

The goal of this project is to help you better understand Bayesian networks and learn to how to apply them in practice.

Introduction

Given some data, in order to use a Bayesian network (BN) to answer some inference queries about it, one would need to go through the following steps:

- Learn the structure of a BN from data,
- Learn the parameters of the BN obtained in the previous step,
- Use the resulting fully-specified BN for inference.

In the first part of this project, you will implement some of the algorithms covered in class for those three steps, namely:

- the K2 algorithm with different score functions for the first step,
- the maximum likelihood approach for the second step,
- the variable elimination algorithm for the last step.

In the second part of this project, you will use your implementation of those algorithms to study two datasets that we provide: `wine` and `protein`. For the first dataset, your learned BN will be used to predict the quality of wines and in the second, it will be used to evaluate the risk of a patient for a disease given the presence of some amino acid types in her DNA sequence.

Part 1: Learning a Bayes Net from Data

In this part, we assume that the training data denoted $\boldsymbol{X} = \left(x_{ij}\right)_{i=1\ldots N, j=1\ldots n}$ where N is the number of training points and n is the number of variables is represented in Python as a 2-dimensional list (i.e., list of lists) `data=`$[[x_{11}, x_{12}, \ldots, x_{1n}], [x_{21}, x_{22}, \ldots, x_{2n}], \ldots, [x_{N1}, x_{N2}, \ldots, x_{Nn}]]$.

Structure Learning

Recall that learning the structure of a BN amounts to solving the following optimization problem

$$\max_{G \in \mathcal{G}} f(G, \boldsymbol{X})$$

where $\mathcal{G}$= all DAGs with n nodes and $f(G, \boldsymbol{X})$ is a score function that evaluates how G fits the data $\boldsymbol{X}$. Given the difficulty of this problem, simplification assumptions have to be made (e.g., so that f can be decomposed as a sum of scores over nodes) and heuristic methods are usually employed (e.g., so that the size of the search space is reduced).

For solving this problem, you will implement the K2 algorithm, which has been presented in class. Recall this algorithm is an heuristic method for solving the following simpler optimization problem:

$$\max_{G \in \mathcal{G}'} f(G, \boldsymbol{X})$$

where $\mathcal{G}'$=DAGs satisfying a fixed topological order, which we assume is the order of the variables in which they appear in $\boldsymbol{X}$. This is without loss of generality because you can always reorder the columns of $\boldsymbol{X}$ before running the K2 algorithm. You will make sure that your implementation is generic such that it can be used with different score functions. For this reason, your K2 algorithm should respect the following signature:

```
[graph, totalScore] = K2Algorithm(K, data, scoreFunction)
```

where the parameters are defined as follows:

- `K` is the maximum number of allowed parents for a variable,
- `data` is a list of lists, containing the training data.,
- `scoreFunction` is a score function whose signature is:

  ```
  score = scoreFunction(variable, parents, data)
  ```

  where `variable` is the index of a variable (i.e., a column in $\boldsymbol{X}$), `parents` is a list of indices that correspond to potential parents for `variable`, `data` is the training data and `score` is the score obtained if `parents` were connected to `variable` in the BN. See the explanation below for `totalScore` to understand the relation between score function f and `scoreFunction`.

- `graph` represents the resulting BN structure. For simplicity, we assume that it is encoded as an adjacency matrix $\boldsymbol{G} = \left(g_{ij}\right)_{i=1,\ldots,n,j=1,\ldots,n} \in \{0,1\}^{n\times n}$ where $g_{ij} = 1$ indicates the existence of a directed edge from node i to node j and $g_{ij} = 0$ indicates the non-existence of such an edge. In Python, `graph` is a 2-dimensional list. See the example below:

  ```
  graph = [[0, 1, 1, 0]
           [0, 0, 0, 0]
           [0, 1, 0, 1]
           [0, 0, 0, 0]]
  ```

  It means that variable 1 is the parent of variables 2 and 3, variable 3 is the parent of variables 2 and 4.

- `totalScore` represents $f(G, \boldsymbol{X})$ that is the sum of the scores given by `scoreFunction` for each variable if the BN structure G is given by `graph`. In other words, it corresponds to the value of the objective function that has been maximized by the K2 algorithm and `graph` corresponds to the graph structure that attains that value.

For the score functions, you will implement the K2 original score and the Bayesian Information Criterion (BIC) score (as described in class) according to the signature of `scoreFunction`. For convenience, we provide you the formula of the i-th variable, denoted X_i, for any $i = 1, \ldots, n$. The Bayesian K2 score function is given by:

$$g(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!$$

where π_i is the list of candidate parents of X_i, q_i is the number of possible assignments for those parents, r_i is the number of possible values for X_i, N_{ijk} is the number of simultaneous occurrences of the k-th value for X_i and the j-th assignment of its parents, and $N_{ij} = \sum_k N_{ijk}$. In practice, to avoid overflow and underflow issues, you will use and compute the log of this score.

Typically, a higher score means the structure fits better the training data. However, if the dataset is large, the BN structure learned with the K2 score may be too complex (because adding edges usually increases the likelihood). The Bayesian Information Criterion (BIC) score is applied to avoid this problem. It can be computed as follows for variable X_i:

$$BIC(i, \pi_i) = 2 \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \left(\frac{N_{ijk}}{N_{ij}} \right) - q_i (r_i - 1) \log N$$

Parameter learning

Assuming that a BN structure is given as an adjacency matrix, its (conditional) probabilities are estimated for each node (i.e., variable) by maximum likelihood. The main function realizing this task has the following signature:

```
cpt = MLEstimationVariable(variable, parents, data)
```

where the arguments have the same semantics as previously and `cpt` is a $(d+1)$-dimensional list with d the number of parents. More specifically, `cpt`$[i_0][i_1]\dots[i_d]$ corresponds to the conditional probability that `variable` takes its i_0-th value given that for $j = 1, \dots, d$, its j-th parent takes its i_j-th value.

To learn the whole BN model, you will call `MLEstimation` on each variable given the parents indicated by the structure `graph` obtained in the previous step. This operation should be realized in the following function:

```
cptList = MLEstimationVariable(graph, data)
```

where `graph` is an adjacency matrix and `cptList` is the list of CPTs, each represented as a list for each variable in the order they appear in `data`.

Inference

With a fully-specified Bayes net, it is possible to perform inference tasks. The answer of inference queries can be computed using the variable elimination algorithm that you will implement. It should have the following signature:

```
prob = variableElimination(index, observations, model)
```

where

- `index` is the index of the hidden variable of interest,
- `observations` is a list of pairs (i, o) where i is the index of an observed variable and o is the index of the corresponding observed value,
- `model` is a pair whose first component is an adjacency matrix and whose second component is a list of CPTs (each represented as a list as described previously),
- `prob` is a list containing the resulting probabilities.

For instance, the call of `variableElimination(4, [(1, 1), (3, 2)], M)` computes $\mathbb{P}(X_4 \mid X_1 = x_{11}, X_3 = x_{32})$ assuming that the BN is represented by model `M`, the variables are named X_i's and their possible values are denoted x_{ij}'s.

Coding Tasks

To summarize, for Part 1, you need to code the following functions:

- `K2Algorithm`
- `K2Score` and `BICScore` according to the specification of `scoreFunction`
- `MLEstimationVariable` and `MLEstimation`
- `variableElimination`

The functions related to structure learning and parameter learning should be provided in a single file named `structurelearning.py`. The function `variableElimination` should be provided in a file named `variableelimination.py`.

For testing, you can run/debug your code on a small artificial dataset you create and/or the example given on Slide 15 on BN learning.

Bonus points will be given if you implement some of the other algorithms seen in class (i.e., the heuristic method based on the maximum weight spanning tree[1] or the belief propagation algorithm) and experimentally compare them to the previous methods in your report. In that case, please name your source code as `algorithm.py` where you replace `algorithm` by the name of the algorithm you implemented.

Part 2: Applications

To help you evaluate your implementations, we provide you some real datasets that have already been cleaned and pre-processed.

Data Format

All data in this project are provided as `csv` files. An example of a dataset with n variables and N cases is in the following format:

var_1	var_2	...	var_n
d_{11}	d_{12}	...	d_{1n}
d_{21}	d_{22}	...	d_{1n}
	...		
d_{m1}	d_{m2}	...	d_{mn}

[1] You can use any libraries and modules in Python to solve the maximum weight spanning tree problem, such as: networkx.algorithms.tree.mst.maximum_spanning_tree.

The first row of the `csv` file corresponds to the names of the n variables of the dataset. Each subsequent row contains one data item for the n variables respectively. To read the data from a file named `data.csv`, you may use the `csv` module of Python3. For instance:

```
import csv

with open('data.csv', 'r', encoding="utf-8") as csvfile:
    reader = csv.reader(csvfile)
    print(list(reader))
```

The output should be:

```
[[var1, var2,...,varn], [d11, d12,...,d1n],...,
 [dm1, dm2,..., dmn]]
```

Before passing the data to your functions, you need to skip the first line of the data file and only store the values of the variables in a list `data`.

Datasets

In the related files of project 2, we provide 2 data sets for you to evaluate your functions: `wine.csv` and `protein.csv`.

The file `wine.csv` contains 12 variables. The first 11 variables describe some features of a wine, which can be experimentally measured:
`fixed acidity`, `volatile acidity`, `citric acidity`, `residual sugar`, `chlorides`, `free sulfur dioxide`, `total sulfur dioxide`, `density`, `ph`, `sulphets`, `alcohol`. The values for those variables have been normalized and discretized. They can take a discrete value from 1 to 5. The last variable `quality` is the quality of the wine, also expressed on a discrete scale from 1 to 5. This dataset will be used to understand the important factors that determine the quality of a wine.

The file `protein.csv` contains 6 variables. The first variable `nuc` represents the risk for a certain disease. It can take three possible values: -1 means high risk and 0 means normal and 1 means immune. The next 5 variables correspond to amino acid types in a given protein DNA sequence. A protein DNA sequence is shown as Figure 1. There are 4 bases in DNA: adenine (A), cytosine (C), guanine (G), thymine (T). Different sequences of those bases generate different types of amino acids. For instance, the first 3 bases "TTG" generates a L-Threonine whose abbreviation is "L".

Figure 1: A protein sequence.

The risk for the disease (`nuc`) can be determined by the ratio of each amino acid. The ratios of each amino acid are provided in the 5 last columns (A, R, N, D, and Q) of the dataset. The values have been again normalized and discretized. Those variables can take discrete values from 0 to respectively 5, 6, 6, 1 and 1. This dataset will be used to study which sequence of DNA bases affects most the risk for this disease.

Coding Tasks

For learning the BN model, you will only use a part of a dataset (say $p\%$) and do the inference on the other part, denoted $\mathcal{D}$. To determine p, you can choose the number of cases such that the learning process takes over 30 minutes. In the report, you should explain your settings (for the upper bound K of parent nodes, number of cases you use, etc). Please also provide the adjacency matrix `Graph` and `totalScore` as the result of your structure learning for each dataset in the report.

To evaluate a BN model, you can compute the answers of some inference queries and compare the results with $\mathcal{D}$. Notably, you will use the following accuracy score for a query $\mathbb{P}(X_1 \,|\, X_i = x_i, \forall i = 2, \ldots, n)$.

$$acc = \frac{\sum_{\boldsymbol{x} \in \mathcal{D}} [\rho(\boldsymbol{x}) = x_1]}{|\mathcal{D}|}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\rho(\boldsymbol{x}) = arg\max_y \mathbb{P}(X_1 = y \,|\, X_i = x_i, \forall i = 2, \ldots, n)$, which is the most probable value for X_1 given the values $x_2, \ldots x_n$.

For the `wine` dataset, the variable of interest, X_1, is taken as `quality` and for the `protein` dataset, it is `nuc`. The score *acc* will allow you to decide which learning method (i.e., score functions and MSWT if you implement it) is better for this prediction task.

To further investigate the datasets, you can use your inference algorithm to answer other similar queries $\mathbb{P}(X_1 \,|\, X_I = x_I)$ to find which subset of variables (whose indices are in $I \subseteq \{2, 3 \ldots, n\}$) is most useful to predict the

variable of interest (i.e., `quality` or `nuc`). This is to answer the following question: if we could only observe $k < n - 1$ variables in $X_2, \ldots, X_n$, which ones are the most related to the variable of interest? You can explain your findings in your report (notably for different k).

The code you write for Part 2 should be saved into two script files `wine.py` and `protein.py`.

Submission and Due Date

You need to submit a zipped file named `P2-Firstname-Lastname.zip` where you replace `Firstname` and `Lastname` by your first name and last name respectively. This compressed file should contain:

- all the source code files mentioned above,
- a `README.txt` file explaining how to run your code in order to reproduce your experiments, and
- a **short** report in pdf format.

The submission will be on Canvas Assignments. The due date is 11:59 pm on Nov 15th, 2018. There will be a penalty of 20% per day for late submission. Note that submissions will be checked for plagiarism.

Grading

Your program will be graded along three criteria:

1. Functional Correctness
2. Implementation Constraints
3. Report

An example of Functional Correctness is whether or not your algorithms produce the correct output. Implementation Constraints checks whether you stick to the implementation requirements. The length of the report has not much importance: we prefer clear and concise comments/explanations.