Object Oriented Programming 2014/15

Learning dynamic Bayesian networks

MEEC – IST

1 Problem

Longitudinal data, also known as **multivariate time series** (MTS) in the machine learning community, is obtained by conducting repeated measurements over time. They frequently arise from biomedical and clinical studies, as well as from other scientific areas, such as economy (e.g. stock market), engineering (e.g. hydrologic cycles) and meteorology (e.g. weather), to name a few. This project is concerned with learning from MTS.

A MTS can be modelled as a set of n-dimensional observations of a stochastic process over T sequential instants of time. The problem of learning from MTS consists in estimating the relations among the variables of the observed process, as well as their joint evolution throughout time. The relations to learn can be divided in two groups: the **intra-temporal** relations, expressing the dependences between the variables at a particular time instant t, and the **inter-temporal** relations, specifying how the variables observed at times $0, \ldots, t$ affect the variables at $t + 1$.

In this project we will use a dynamic Bayesian network (DBN) to model a MTS. DBNs are defined upon Bayesian networks (BN). BN are used to model **time-invariant** processes, whereas DBN model **time-variant** ones.

2 Bayesian networks

For convenience, consider the following notation. Let:

- $\vec{X} = (X_1, \ldots, X_n)$ be a random vector composed by random variables X_i, each having a domain $\mathcal{X}_i \subset \mathbb{R}$.
- The elements of $\mathcal{X}_i$ are denoted by $x_{i1}, \ldots, x_{ir_i}$, where r_i is the number of values X_i can take, i.e., the size of $\mathcal{X}_i$.
- In a directed graph, a node u is a parent of node v if there is an arc going from u to v. The set of parents of a node X is denoted by $\vec{pa}(X)$.

A **Bayesian network** (BN) is a graphical representation of a joint probability distribution over a set of random variables and it is defined as a triple $B = (\vec{X}, G, \boldsymbol{\theta})$, where:

- $\vec{X} = (X_1, \ldots, X_n)$ is a random vector;
- $G = (\vec{X}, E)$ is a **directed acyclic graph** (DAG) whose nodes correspond to the elements of $\vec{X}$ and edges E specify conditional dependences between the variables.
- $\boldsymbol{\theta} = \{\theta_{ijk}\}$ is a set of parameters, specifying the local probability distributions of the network via
$$\theta_{ijk} = P_B(X_i = x_{ik} \mid \vec{pa}(X_i) = w_{ij}), \tag{1}$$
where $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, q_i\}$ and $k \in \{1, \ldots, r_i\}$. The set of possible configurations of $\vec{pa}(X_i)$, i.e., the set of different combinations of values that the parents of X_i can take, is denoted by $\{w_{i1}, \ldots, w_{iq_i}\}$, where $q_i = \prod_{X_j \in \vec{pa}(X_i)} r_j$ is the number of all possible configurations.

A Bayesian network B defines a joint probability distribution over $\vec{X}$:
$$P_B(X_1, \ldots, X_n) = \prod_{i=1}^{n} P_B(X_i \mid \vec{pa}(X_i)). \tag{2}$$

2.1 An algorithm to learn Bayesian networks

The problem of learning a BN given data D consists in finding the BN that best fits the data D. In this project we focus our attention on **score-based learning** approaches, where a **scoring criterion** ϕ is considered in order to quantify the fitting of a BN. In this context, the problem of learning a BN given a data D and a scoring criterion ϕ, recasts to finding a BN that maximizes the value $\phi(B, D)$.

Score-based learning algorithms can be extremely efficient if the scoring criterion employed is decomposable, named in that case **local score-based learning** algorithms. A scoring criterion ϕ is **decomposable** if the score assigned to each network decomposes over the network structure in such a way that it can be expressed as a sum of local scores ϕ_i that depends only on each node X_i and its parents, that is, scores of the following form:
$$\phi(B, D) = \sum_{i=1}^{n} \phi_i(\vec{pa}(X_i), D).$$

Learning unrestricted BNs from data under typical scoring criteria is NP-hard. Indeed, exact polynomial-time bounded approaches for learning BNs are only possible for a handful of network structures. Consequently, the standard methodology for addressing the problem of learning BNs became heuristic search, based on scoring metrics optimization, conducted over some search space. In this project we will focus in one of this heuristic methods.

Learning BNs is typically divided in **structure learning** and **parameter learning**. Structure learning is guided by a scoring criterion that performs a task of **model selection**.

2.1.1 Structure learning

Suppose we have some network structure, which is a DAG. We can define its **neighbourhood** in **DAG-space** as all networks we can reach by applying one of the **operators**:

- **add** an edge;
- **delete** an edge;

- **reverse** an edge.

At each search step, we find the best neighbour and move to it; only one operator is applied at each time. Note, however, that this best neighbour must be in DAG-space. When removing an edge from a DAG the resulting network will always be a DAG, but when adding or flipping an edge from a DAG we need to guarantee that the resulting network is still a DAG.

This procedure is called **Greedy hill-Climbing** (GHC) and it can get stuck in local optima. Two simple strategies can be effective to escape from these local optima:

- **TABU list**: do not revisit recently seen structures;
- **Random restarts**: apply some operators at random when at a local optimum.

The canonical GHC procedure is presented in Algorithm 1. The stopping criterion $\mathcal{C}$ may vary; it can be the score of the best network exceeds a threshold, a local maximum is attained, etc. In this project we will use as a stopping criterion attaining a local maximum, that is, $\phi(\mathcal{N}_{res}) \geq \phi(\mathcal{N}')$, where $\mathcal{N}_{res}$ is the best structure found so far and $\mathcal{N}'$ is the best neighbour structure found in the previous step. Typically the number of parents a node can have is also upper bounded. In this project we consider that a node has at most 3 parents.

Algorithm 1 GHC algorithm for learning BNs

```
Input: initial structure $\mathcal{N}_{init}$, dataset $D$, a scoring function $\phi$, stopping criteria $\mathcal{C}$
Output: final structure $\mathcal{N}_{res}$

1. $\mathcal{N}_{res} = \mathcal{N}_{init}$ and $\mathcal{N}' = \mathcal{N}_{res}$
2. while $\mathcal{C}$ is not satisfied
3.     $\mathcal{N}'' = \arg\max_{\mathcal{N} \in \text{neighbourhood}(\mathcal{N}')} \phi(\mathcal{N})$
4.     if $\phi(\mathcal{N}'') > \phi(\mathcal{N}_{res})$ then $\mathcal{N}_{res} = \mathcal{N}''$
5.     $\mathcal{N}' = \mathcal{N}''$
6. end while
7. return $\mathcal{N}_{res}$
```

Algorithm 2 elicits the usage of TABU list and random restarts within the GHC algorithm. In this case the stopping criterion $\mathcal{C}$ can be a maximum number of random restarts. In this project consider it to be a user parameter.

2.1.2 Parameter learning

When the structure of the network is fixed in advance maximizing the likelihood of the data T reduces to estimating the parameters θ_{ijk}. In this case, the *maximum likelihood* (ML) parameters are simply the **observed frequency estimates** (OFE) given by

$$\hat{\theta}_{ijk} = \hat{P}_T(X_i = x_{ik} \mid \vec{pa}(X_i) = w_{ij}) = \frac{N_{ijk} + N'}{N_{ij} + r_i \times N'}$$

where:

- N_{ijk} is the number of instances in the data D where the variable X_i takes its k-th value x_{ik} and the variables in $\vec{pa}(X_i)$ take their j-th configuration w_{ij};

Algorithm 2 GHC algorithm for learning BNs, with TABU list and random restarts

```
Input: initial structure $\mathcal{N}_{init}$, dataset $D$, a scoring function $\phi$, stopping criteria $\mathcal{C}$
Output: final structure $\mathcal{N}_{res}$

1. $\mathcal{N}_{res} = \mathcal{N}_{init}$, $\mathcal{N}' = \mathcal{N}_{res}$ and TABU = $\{\mathcal{N}_{res}\}$
2. while $\mathcal{C}$ is not satisfied
3.     $\mathcal{N}'' = \arg\max_{\mathcal{N} \in \text{neighbourhood}(\mathcal{N}') \text{ and } \mathcal{N} \notin \text{TABU}} \phi(\mathcal{N})$
4.     if $\phi(\mathcal{N}') > \phi(\mathcal{N}'')$ then $\mathcal{N}''$ = random($\mathcal{N}'$)
5.     if $\phi(\mathcal{N}'') > \phi(\mathcal{N}_{res})$ then $\mathcal{N}_{res} = \mathcal{N}''$
6.     TABU = TABU $\cup\, \mathcal{N}'$
7.     $\mathcal{N}' = \mathcal{N}''$
8. end while
9. return $\mathcal{N}_{res}$
```

- N_{ij} is the number of instances in the data D where the variables in $\vec{pa}(X_i)$ take their j-th configuration w_{ij}.
- N is the total number of instances in data D; and
- N' are *pseudo-counts* that avoid the common mistake of assigning probability zero to an event that is extremely unlikely, but not impossible. In this project we always take N' to be 0.5.

2.1.3 Model selection

In this project we consider only two different scoring criteria, the *log-likelihood* (LL) score and the *minimum description length* (MDL) score. The LL score is defined as

$$\mathrm{LL}(B|D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log\left(\frac{N_{ijk}}{N_{ij}}\right). \tag{3}$$

The LL score tends to favour complete network structures and it does not provide an useful representation of the independence assumptions of the learned network. This phenomenon that leads to *overfitting* is usually avoided by limiting the number of parents per network variable and/or using some penalization factor over the LL score. The MDL scoring criterion appeared in this context and it is defined as

$$\mathrm{MDL}(B|D) = \mathrm{LL}(B|D) - \frac{1}{2}\log(N)|B|, \tag{4}$$

where $|B|$ denotes the network complexity, that is, the number of parameters in Θ for the network B, and it is given by:

$$|B| = \sum_{i=1}^{n}(r_i - 1)q_i.$$

2.2 An example of computing the parameters of a Bayesian network

The example provided in this section considers that the BN structure is already fixed and learned from data D. From this fixed network structure, the counts needed to compute the BN parameters and score are computed as follows.

As an example consider $\vec{X} = (X_1, X_2, X_3)$ where X_1 and X_3, are binary random variables, and X_2 is a ternary random variable. In this case, the number of values each variable can take is given by:

- $r_1 = 2$, with $x_{11} = 0$ and $x_{12} = 1$;
- $r_2 = 3$, with $x_{21} = 0$, $x_{22} = 1$, $x_{23} = 2$;
- $r_3 = 2$, with $x_{31} = 0$ and $x_{32} = 1$.

Consider the BN B with the following network structure

$$B \equiv X_1 \to X_2 \to X_3.$$

The number of parent configurations each variable can take is given by:

- $q_1 = 1$, with $w_{11} = \epsilon$, where ϵ is the empty configuration;
- $q_2 = 2$, with $w_{21} = x_{11} = 0$ and $w_{22} = x_{12} = 1$, as $\vec{pa}(X_2) = \{X_1\}$ and X_1 is binary;
- $q_3 = 3$, with $w_{31} = x_{21} = 0$, $w_{32} = x_{22} = 1$ and $w_{33} = x_{23} = 2$, as $\vec{pa}(X_3) = \{X_2\}$ and X_2 is ternary.

In addition, consider the dataset D to be given by

X_1	X_2	X_3
0	0	0
0	1	1
1	2	0
1	1	1
0	1	0
1	2	0
1	0	1

and, so, $n = 3$ and $N = 7$.

For the given dataset D and BN B, we have that the N_{ijk} counts, for all i, j, k, are:

	1st parent config.			2nd parent config.			3rd parent config.	
X_1	$N_{111} = 3$	$N_{112} = 4$						
X_2	$N_{211} = 1$	$N_{212} = 2$	$N_{213} = 0$	$N_{221} = 1$	$N_{222} = 1$	$N_{223} = 2$		
X_3	$N_{311} = 1$	$N_{312} = 1$		$N_{321} = 1$	$N_{322} = 2$		$N_{331} = 2$	$N_{332} = 0$

In addition, the $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ counts, for all i, j, are given by:

	1st parent config.	2nd parent config.	3rd parent config.
X_1	$N_{11} = 7$		
X_2	$N_{21} = 3$	$N_{22} = 4$	
X_3	$N_{31} = 2$	$N_{32} = 3$	$N_{33} = 2$

The parameters stored in the BN B at node X_1 are those given by:

	$x_{11} = 0$	$x_{12} = 1$
$w_{11} = \epsilon$	$\hat{\theta}_{111} = \frac{N_{111}+N'}{N_{11}+r_1 \times N'} = \frac{3+0.5}{7+2\times 0.5} = \frac{3.5}{8}$	$\hat{\theta}_{112} = \frac{N_{112}+N'}{N_{11}+r_1 \times N'} = \frac{4+0.5}{7+2\times 0.5} = \frac{4.5}{8}$

In addition, the parameters stored in the BN B at node X_2 are those given by:

	$x_{21} = 0$	$x_{22} = 1$	$x_{23} = 2$
$w_{21} = 0$	$\hat{\theta}_{211} = \frac{N_{211}+N'}{N_{21}+r_2 \times N'} = \frac{1+0.5}{3+3\times 0.5} = \frac{1.5}{4.5}$	$\hat{\theta}_{212} = \frac{N_{212}+N'}{N_{21}+r_2 \times N'} = \frac{2+0.5}{3+3\times 0.5} = \frac{2.5}{4.5}$	$\hat{\theta}_{213} = \frac{N_{213}+N'}{N_{21}+r_2 \times N'} = \frac{0+0.5}{3+3\times 0.5} = \frac{0.5}{4.5}$
$w_{22} = 1$	$\hat{\theta}_{221} = \frac{N_{221}+N'}{N_{22}+r_2 \times N'} = \frac{1+0.5}{4+3\times 0.5} = \frac{1.5}{5.5}$	$\hat{\theta}_{222} = \frac{N_{222}+N'}{N_{22}+r_2 \times N'} = \frac{1+0.5}{4+3\times 0.5} = \frac{1.5}{5.5}$	$\hat{\theta}_{223} = \frac{N_{223}+N'}{N_{22}+r_2 \times N'} = \frac{2+0.5}{4+3\times 0.5} = \frac{2.5}{5.5}$

Moreover, the parameters stored in the BN B at node X_3 are those given by:

	$x_{31} = 0$	$x_{32} = 1$
$w_{31} = 0$	$\hat{\theta}_{311} = \frac{N_{311}+N'}{N_{31}+r_3 \times N'} = \frac{1+0.5}{2+2\times 0.5} = \frac{1.5}{3}$	$\hat{\theta}_{312} = \frac{N_{312}+N'}{N_{31}+r_3 \times N'} = \frac{1+0.5}{2+2\times 0.5} = \frac{1.5}{3}$
$w_{32} = 1$	$\hat{\theta}_{321} = \frac{N_{321}+N'}{N_{32}+r_3 \times N'} = \frac{1+0.5}{3+2\times 0.5} = \frac{1.5}{4}$	$\hat{\theta}_{322} = \frac{N_{322}+N'}{N_{32}+r_3 \times N'} = \frac{2+0.5}{3+2\times 0.5} = \frac{2.5}{4}$
$w_{33} = 2$	$\hat{\theta}_{331} = \frac{N_{331}+N'}{N_{33}+r_3 \times N'} = \frac{2+0.5}{2+2\times 0.5} = \frac{2.5}{3}$	$\hat{\theta}_{332} = \frac{N_{332}+N'}{N_{33}+r_3 \times N'} = \frac{0+0.5}{2+2\times 0.5} = \frac{0.5}{3}$

3 Dynamic Bayesian networks

For convenience, consider the following notation. Let:

- $\vec{X} = (X_1, \ldots, X_n)$ be a random vector composed by the attributes that the are changed by some process.
- $\vec{X}[t] = (X_1[t], \ldots, X_n[t])$ be a random vector that denotes the instantiation of the attributes at time t.
- $\vec{X}[t_1 : t_2]$ be the set of random vectors $\vec{X}[t]$ for $t_1 \leq t \leq t_2$; for instance,

$$\vec{X}[0:T] = \vec{X}[0] \cup \cdots \cup \vec{X}[T], \text{for} T \geq 0.$$

- $P(\vec{X}[0], \ldots, \vec{X}[T])$, abbreviated by $P(\vec{X}[0:T])$, be the joint probability distribution over the temporal trajectory of the process from $\vec{X}[0]$ to $\vec{X}[T]$.

A **Dynamic Bayesian Network** (DBN) is a representation of the joint probability distributions over all possible trajectories of a process. It is composed by:

- a **prior network** B^0, which is a BN that specifies a distribution over the initial states $\vec{X}[0]$;
- a set of **transition networks** B_0^t over the variables $\vec{X}[0:t]$, specifying the state transition probabilities, for $0 < t \leq T$.

In the previous definition, the timeline is discretized into several **time-slices**, each comprising measurements of the process at a specific time instant. Slices are equally spaced in time, which means that the sampling interval is constant. As the temporal trajectories can be extremely complex, further simplifying assumptions on the process are usually made.

A common premise is to consider **first-order Markov** processes, in which future values only depend on present ones but not on the past trajectory, such that

$$P(\vec{X}[t+1] \mid \vec{X}[0:t]) = P(\vec{X}[t+1] \mid \vec{X}[t]).$$

Another hypothesis is to assume that transition probabilities do not depend on t, i.e., the process is **stationary**.

In this project we focus solely first-order Markov and stationary DBNs.

3.1 An algorithm to learn dynamic Bayesian networks

As aforementioned, this project focus solely in first-order Markov and stationary DBNs. Therefore, only two networks need to be learned: the initial network B_0 and, for all $0 \leq t < T$, $\underline{\text{only one}}$ transition network B_t^{t+1}.

In this project a GHC procedure will also be used to learn a DBN. The initial network B_0 is learned as explained in Section 2.1. To learn the transition network B_t^{t+1} the GHC procedure needs to be adapted to consider that edges can only appear from time-slice t to time-slice $t+1$, or from time-slice $t+1$ to $t+1$; in this case the DAG restriction need only to be verified when considering edges in the same time-slice. Note that the local contribution of node $X_i[t+1]$ to the global score of B_t^{t+1} is given by the parents from the same time-slice $t+1$ as well as the parents from the previous time-slice t.

3.2 Computing the probability from a transition network

After learning a DBN, we typically want to know the value of some variable $X_i[t+1]$ given that we already know the values of $\vec{X}[t]$, inferring in this way future values from past ones. To this end we need to compute for every possible value x_{ik} of $X_i[t+1]$ the one that yields the highest probability given by:

$$P\left(X_i[t+1] = x_{ik} \mid \vec{X}[t] = (v_1, \dots, v_n)\right) = \sum_{d_1=1}^{r_1} \cdots \sum_{d_{i-1}=1}^{r_{i-1}} \sum_{d_{i+1}=1}^{r_{i+1}} \sum_{d_n=1}^{r_n}$$
$$P\left(\vec{X}[t+1] = (d_1, \dots, d_{i-1}, x_{ik}, d_{i+1}, \dots, d_n) \mid \vec{X}[t] = (v_1, \dots, v_n)\right).$$

4 Implementation details

The following sections provide further details about the project implementation, namely, program parameters, input files format, running in the command line and results presentation.

4.1 Program parameters

The program should receive the following parameters

train	filename of a dataset from which a GHC network is going to be learned
test	filename of a dataset from which we want to perform inference
score	the score to be used to build the GHC network
randrest	maximum number of random restarts for the GHC procedure
var	index of $\vec{X}[t+1]$ from which we want to know the most probable value

and it should return the results of inferring from the instances in *test* (with the values of $\vec{X}[t]$) the most probable value of $X_{var}[t+1]$, according to the DBN built from the *train* dataset. The score used to build the DBN is determined in the parameter *score* and it could be one of the following strings: LL or MDL (case sensitive). The number of random restarts needed by the GHC procedure is given in the parameter *randrest*; if *randrest* is not used in the implementation, it should be ignored (but it is anyway passed to the program). If *var* is not given, all indexes from 1 to n should be considered.

4.2 Input files format

The *train* dataset must be provided as a .csv file[*] formatted as

$$\begin{array}{cccccccccc}
X_1[0], & \ldots, & X_n[0], & X_1[1], & \ldots, & X_n[1], & \ldots, & X_1[T], & \ldots, & X_n[T] \\
v_{11}[0], & \ldots, & v_{n1}[0], & v_{11}[1], & \ldots, & v_{n1}[1], & \ldots, & v_{11}[T], & \ldots, & v_{n1}[T] \\
\ldots, & \ldots, & \ldots, & \ldots, & \ldots, & \ldots, & \ldots, & \ldots, & \ldots, & \ldots \\
v_{1N}[0], & \ldots, & v_{nN}[0], & v_{1N}[1], & \ldots, & v_{nN}[1], & \ldots, & v_{1N}[T], & \ldots, & v_{nN}[T]
\end{array}$$

where the first line corresponds to the name of the variables in the random vector

$$(\vec{X}[0], \vec{X}[1], \ldots, \vec{X}[T])$$

and the following lines to their values in each instance of the training data. For the sake of simplicity, we assume that any variable $X_i[t]$, with $1 \leq i \leq n$ and $0 \leq t \leq T$, ranges from 0 to $r_i - 1$. Observe, however, that all instances in the data need not to have the same size, and so the data might not be rectangular. Recall notation introduced in previous sections to understand the size of the data.

The *test* set is also a .csv file that provides new instances from which we want to perform inference, based on the DBN learned from the *train* dataset as explained in the previous sections, and complies with the following format:

$$\begin{array}{ccc}
X_1[t], & \ldots, & X_n[t] \\
y_{11}[t], & \ldots, & y_{n1}[t] \\
\ldots, & \ldots, & \ldots \\
y_{1z}[t], & \ldots, & y_{nz}[t]
\end{array}$$

where z is the number of instances in the *test* set and t any instant with $0 \leq t < T$. In the *test* set we also assume that variable $X_i[t]$, with $1 \leq i \leq n$ and $0 \leq t < T$, ranges from 0 to $r_i - 1$.

A few examples of *train* and *test* sets will be provided in the "Project section" of the OOP website. Stay tuned!

4.3 Running in the command line

A .jar file must be created so that the program runs by typing in the terminal

[*]Comma separated value (.csv) files can be opened in excel or in any text editor.

```
java -jar <<YOUR-JAR-NAME>>.jar train test score randrest var
```

where *train* and *test* are the names of the input files, and *score* is the string that identifies the score to be used to build the DBN model. These input files should be found outside of the `.jar` file. An absolute or a relative path may be used. The *randrest* is the number of random restarts used to build the GHC network (if Algorithm 2 is implemented); the *var* is the index of $\vec{X}[t+1]$ from which we want to perform inference (optional parameter, c.f. Section 4.1).

4.4 Results

When running the program in the command line the following output must be printed to the terminal:

```
Building DBN:            TIME time
Initial network:         format to be defined!
Transition network:      format to be defined!
Performing inference:
-> instance 1:           inferred_value_for_1
...                      ...
-> instance z:           inferred_value_for_z
```

where:

- $TIME$ is the time spent to build the DBN model; and
- $inferred\_value\_for\_l$, with $1 \leq l \leq z$, is the result of inferring from the l-th instance of the *test* set, according to the learned DBN, as described in Section 3.2. If inference is to be performed in more that one index of $\vec{X}[t+1]$, present the inferred values separated by commas in increasing order of the indexes.

5 Minimum requirements and bonus points

In this project it is required an implementation of GHC as described in Algorithm 1, but adapted for learning DBN as described in Section 3.1. That is, a **GHC procedure for learning DBN without TABU list and random restarts**. Moreover, as the project aims at computing the probability from a transition network (see Section 3.2), **only the transition network B_t^{t+1} needs to be computed**, and so, there is no need to compute the initial network B_0. Recall that the learned DBN needs only to be first-order Markov and stationary, hence, for all $t \in \{0, \ldots, T-1\}$, only one transition network B_t^{t+1} is learned from all data (see support slides for detail). These are the minimum requirements to access the project to a maximum of 6 points (according to Section 6) out of the final grade.

However, if a **GHC version with TABU list and random restarts is implemented** (Algorithm 2) **to learn both the initial network B_0 and the transition network B_t^{t+1}** a bonus point may be added to the final mark of the project. Moreover, if a **Swing GUI is provided**, a bonus of half a point may be also added to the final mark of the project. Therefore, as the project assesses 6 points out of the final grade, with a **correct and efficient**

implementation of Algorithm 2 and a **fully-functional and correctly implemented** GUI, the project may evaluate to 7.5 points (instead of 6).

Notwithstanding, even if a Swing GUI is provided, the program must also run from the command line as described in Section 4.3.

6 Grading

The assessment will be based on the following 6-point scale:

1. **(1 point)**: UML. The UML will be evaluated, in a 1-point scale as: 0–very bad, 0.25–bad, 0.5–average, 0.75–good and 1–excellent.
2. **(5 points)**: A solution that provides an extensible and reusable framework for learning DBNs. The implementation of the requested features in Java are also an important evaluation criteria and the following discounts, on a 5-point scale, are pre-established:
   (a) **(-2 points):** OOP ingredients are not used or they are used incorrectly; this includes polymorphism, open-close principle, etc.
   (b) **(-1 points):** Java features are handled incorrectly; this includes incorrect manipulation of methods from `Object`, `Collection`, etc.
   (c) **(-0.5 points):** Prints outside the format requested in Section 4.4.
   (d) **(-0.5 points):** A non-executable jar file, or a jar file without sources or with sources out of date. Problems in extracting/building a jar file, as well as compiling/running the executable in Java, both from the command line. Problems with Java versions; the Java executable should run properly in the laboratory PCs.

Finally, in a 6-point scale:

1. Files submitted outside of the required format will have a penalty of 5% over the respective grade.
2. Projects submitted after the established date will have the following penalty: for each day of delay there will be a penalty of 2^n points of the grade, where n is the number of days in delay. That is, reports submitted up to 1 day late will be penalized in $2^1 = 2$ points, incurring in a penalty of 0.6 points of the final grade; reports submitted up to 2 days late will be penalized in $2^2 = 4$ points, incurring in a penalty of 1.2 points of the final grade; and so on. Per day of delay we mean cycles of 24h from the day specified for submission.

7 Deadlines and material for submission

The **deadline for submitting the project is May 18th, before 14:00**. The submission is done via fenix, so ensure that you are registered in a project group.

The following files must be submitted:

1. An UML specification including classes and packages (as detailed as possible), in `.pdf` or `.jpg` format. Place the UML files inside a folder named UML.
2. An executable `.jar` (with the respective source files `.java`, compiled classes `.class`, and `MANIFEST.MF` correctly organized into directories).

3. Documentation (generated by the javadoc tool) of the application. Place the documentation inside a folder named JDOC.
4. A final report (up to 10 pages, in .pdf format) containing information that complements the documentation generated by the javadoc tool. Place the final report inside a folder named DOCS.
5. A self assessment form (in `.pdf` format) that will be made available in due time in the course webpage. Place the self assessment form inside the folder named DOCS (the same folder as the final report).

The UML folder, executable (the `.jar` file with the source files, besides the compiled files and `MANIFEST.MF`), the JDOC folder and the DOCS folder, should be **submitted via fenix in a single `.zip` file**.

The final discussion will be held from May 19 to May 30. The distribution of the groups for final discussion will be available in due time. All group members must be present during the discussion. The final grade of the project will depend on this discussion, and it will not necessarily be the same for all group members.