Bike sharing Forecasting through Bayesian Networks

Fundamentals of Artificial Intelligence Module 3

Pietro Epis, Michele Milesi, Anna Valanzano

April 2022

Abstract

Basing on a dataset about bike sharing in London we designed a Bayesian Network to make some simple predictions through exact and approximate inference.

1 London bike sharing dataset

In Table 1 we report a short description about the dataset of bike sharing in London (or further references see London bike sharing dataset)

Variables	Type	Domain
cnt	int	[0, 7860]
temperature	float	[-1.5, 34]
temperature1_feels	float	[-6, 34]
wind	float	[0, 56.5]
hum	float	[20.5, 100]
weather_code	int	{1, 2 , 3, 4, 7, 10, 26, 94}
timestamp	string	dates and hours in 2015
is_holiday	integer	{0, 1}
is_weekend	integer	{0, 1}
season	integer	{0, 1, 2, 3}

Table 1: Description of the dataset **before the preprocessing**

The variable *cnt* represents the count of new bike sharing, while all the other variables refer to several factors that might influence the bike sharing, regarding for instance the weather (as all the variables in Table 1 between *temperature* and *weather_code*) and the date (as the remaining variables as *timestamp*).
The main problem was related to continuos variables as *cnt*, which for instance can assume any value in the range [0, 7860], or as *timestamp*), which can represent any date and hour in 2015.
The continuos domains are intractable trough Bayesian Network. That's way we converted all the continuos domains in discrete ones. For example, the count of new bike sharing has been bucketized in seven bins, the temperature in four bins, and so on.
Futhermore, we changed some domain only to make it more intuitive. For instance, *is_holiday*, an integer with domain {0, 1}, became a string, that is "yes" if it is holiday, "no" otherwise.
We report in In Table 2 the description of the dataset after all the preprocessing actions.

Variables	Type	Cardinality	Domain
cnt	integer	8	{0, 1, 2, 3, 4, 5, 6, 7}
temperature	integer	4	{0, 1, 2, 3}
temperature_feels	integer	4	{0, 1, 2, 3}
wind	string	2	{"yes", "no"}
hum	integer	4	{0, 1, 2, 3}
weather	string	8	see Table 3
time	string	2	{"morning", "afternoon", "evening", "night"}
is_holiday	string	2	{"yes", "no"}
is_weekend	string	2	{"yes", "no"}
season	string	4	{"spring", "summer", "fall", "winter"}

Table 2: Description of the dataset of the dataset **after the preprocessing**

Weather_code	Values
1	Clear
2	Scattered clouds
3	Broken clouds
4	Cloudy
7	Rain
10	Thunderstorm
26	Snowfall
94	Freezing Fog

Table 3: The new domain of the variable *weather*

2 Construction of the Bayesian Network

To define the Bayesian Network, that is a data structure representing the dependencies among the variables, we used the Python library `pgmpy` and in particular the class `BayesianNetwork`. Then, we used the methods `add_nodes_from` and `add_edge` to add respectively nodes and edges and, basing on our intuitions about the dataset. For instance, we deemed it appropriate to represent the dependency of the variable `temperature_feels` from the variables `temperature`, `hum` and `wind`, indeed the perceived temperature is likely to increase in absence of wind and whether humidity is high. Furthermore, the edge from `time` to `cnt` formalizes the idea that the number of rented bikes is more likely to be high during the day rather than during night. The other edges have been introduced according to similar reasonings.
We defined the Conditional Probability tables trough the method `TabularCPD`. We reported the resulting Network structure in Figure 1.

Generation of CPDs Since it wouldn't have been feasible to manually define the CPD Tables for variables with large domain or lots of dependencies, we developed a function devoted to generate the table of conditional probabilities associated to a specific variable, according to its parents (that represent its dependencies) and their cardinality.

cnt	p(cnt | E)
[47, 473]	0.2583
(473, 753]	0.1790
(753, 992]	0.1720
(992, 1256]	0.0925
(1256, 1595]	0.0866
(1595, 2066]	0.0881
(2066, 2762]	0.0629
(2762, 7860]	0.0605

Table 4: Result of exact inference query

3 Exact Inference

The basic task for any probabilistic inference system is to compute the posterior probability distribution for a set of query variables, given some observed event. Therefore, given a set of evidence variables E and a query variable X a typical query asks for the posterior probability distribution

$$P(X|e) = \frac{P(X,e)}{P(e)} = \alpha P(X|e) \quad (1)$$

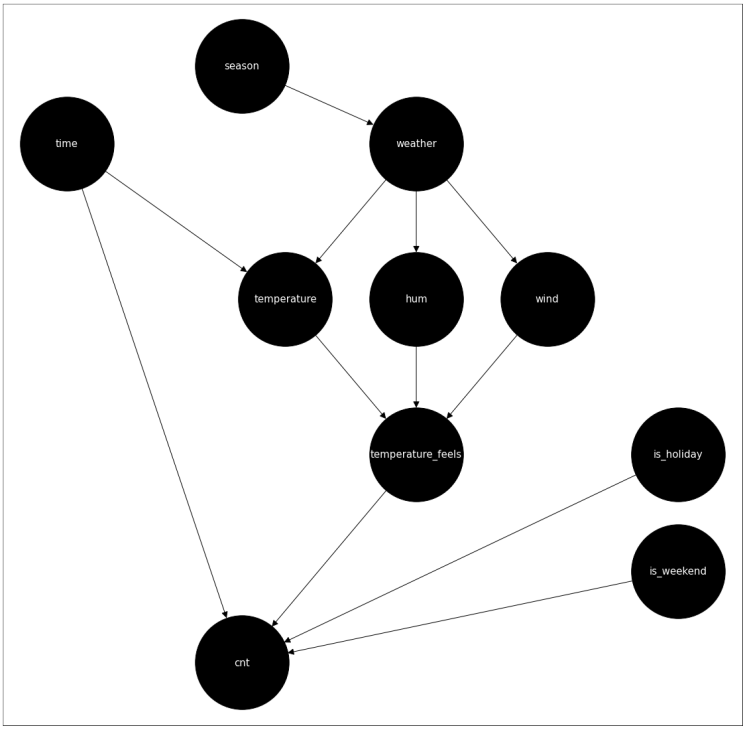

Figure 1: The Network structure

Unfortunately, inference is a very challenging task: for many probabilities of interest, answering to any of these queries exactly is a NP-hard problem. However, the Variable Elimination Algorithm allows a more efficient computation, basing on the simple idea of performing calculations once and saving results for later use. To exploit this efficient strategy we use the `pgmpy` class `VariableElimination` and we performed some queries on the corresponding object. For example, a query was about the number of bikes that would be rented if the season is winter, the temperature is low, it is windy and rainy. Therefore, the evidence E is $E = \{temperature = [-1, 7.75], wind = yes, weather = rain, season = winter\}$. In Table 4 there are the results of this query.

4 Approximate Inference

Given the intractability of exact inference in large and multiple connected networks, it turns out essential to consider approximate methods. In this report we examined two Direct Sampling meth-

ods: the Rejection Sampling and the Likelihood weighting algorithm. To implement approximate inference we used the `pgmpy` class `BayesianModelSampling`.

4.1 Rejection Sampling

To compute the approximate probability $\widehat{P}(X|e)$ the Rejection Sampling technique generate samples and reject those that are not consistent with the evidence e. After generating all the samples that match the evidence, the algorithm compute the approximate probability $\widehat{P}(X|e)$ as the ratio between the number of times in which the X occurs in the samples and the total number of samples:

$$\widehat{P}(X|e) = \frac{N_{PS}(X,e)}{N_{PS}(e)} \approx \frac{P(X,e)}{P(e)} = P(X|e)$$

If the probability of the evidence $P(e)$ is small, Rejection Sampling can be very expensive, because it generates a lot of samples and rejects most of them.

4.2 Likelihood Weighting

The Likelihood Weighting technique avoids the inefficiency of Rejection Sampling by generating only events that are consistent with the evidence e. Each event is weighted by the likelihood of the event according to the evidence: an event in which the evidence appears unlikely is given less weight[1]. Therefore, to compute $\widehat{P}(X|e)$, the algorithm sums the weights of samples in which X occurs and divides the result by the sum of all the weights.

4.3 Comparing the different methods

We generated several sets of samples of different dimension according to the two strategies using the `pgmpy` methods `rejection_sample` and `likelihood_weighted_sample`. Afterwards, we computed the approximate probabilities as reported in Sections 4.1 and 4.2.

We consider two queries:

- Query 1: which is the probability that $time = evening$ if $cnt = 3$ and $temperature = 1$?
- Query 2: which is the probability that $cnt = 3$ if $temperature = 1$?

In Figure 2 we report how the approximate probabilities change varying the size of the set of samples for the two queries using Exact and Approximate Inference.

5 Conclusions

We can observe that increasing the number of samples the approximate probabilities become closer and closer to the exact ones. In fact, as we can see in Figure 3, increasing the number of samples the absolute and relative error become closer and closer to zero.

[1] Artificial Intelligence. A Modern Approach, by Stuart Russel and Peter Norvig, 3rd Ed.

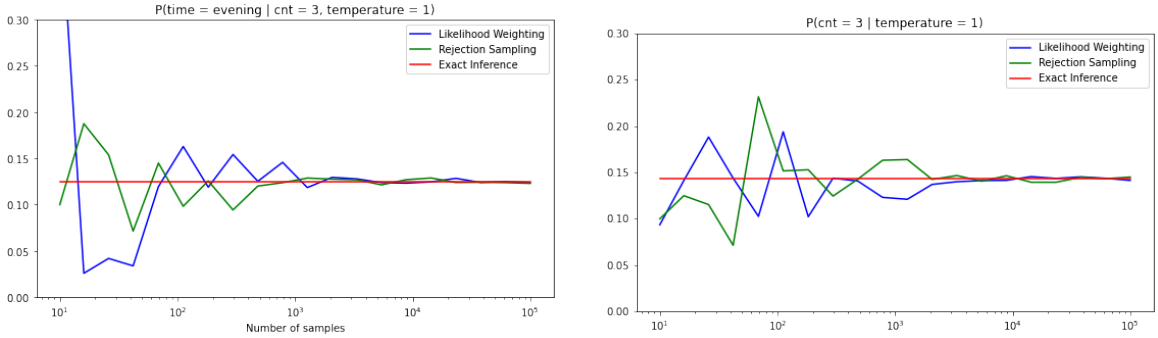

Figure 2: Probabilities using exact and approximate inference

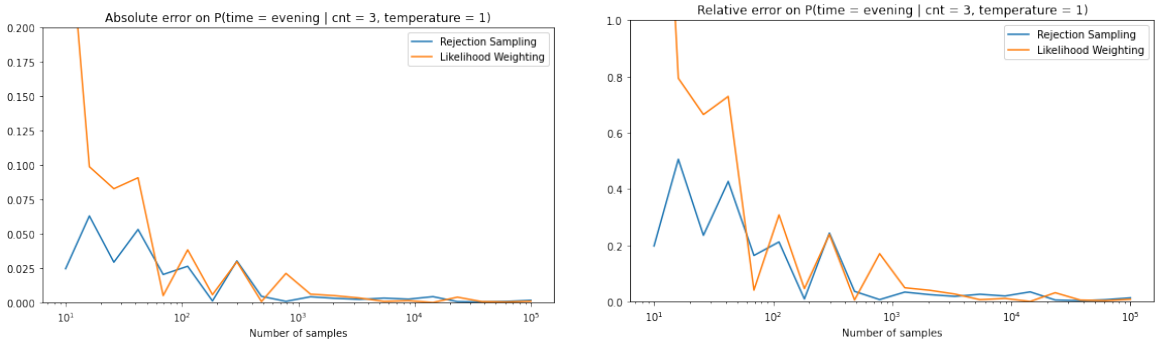

Figure 3: Absolute and Relative errors for the first query