Product availability Bayesian network

When a new product is about to be introduced to the market, it is often convenient to create a sale forecast and to estimate the quantity of the good that can be sold over a defined period of time. This could be a very useful piece of information for the manufacturer, or the seller in general, in order to come up with advantageous insights and to make correct decision about the quantity to be produced/put on the market or the right price to assign to the product. Knowing if a product is going to be sold out everywhere immediately after its release or if instead very few people would want to buy it, can be beneficial also to the customer, who can decide to wait and make the purchase at a lower price or simply avoid long lines at the stores.

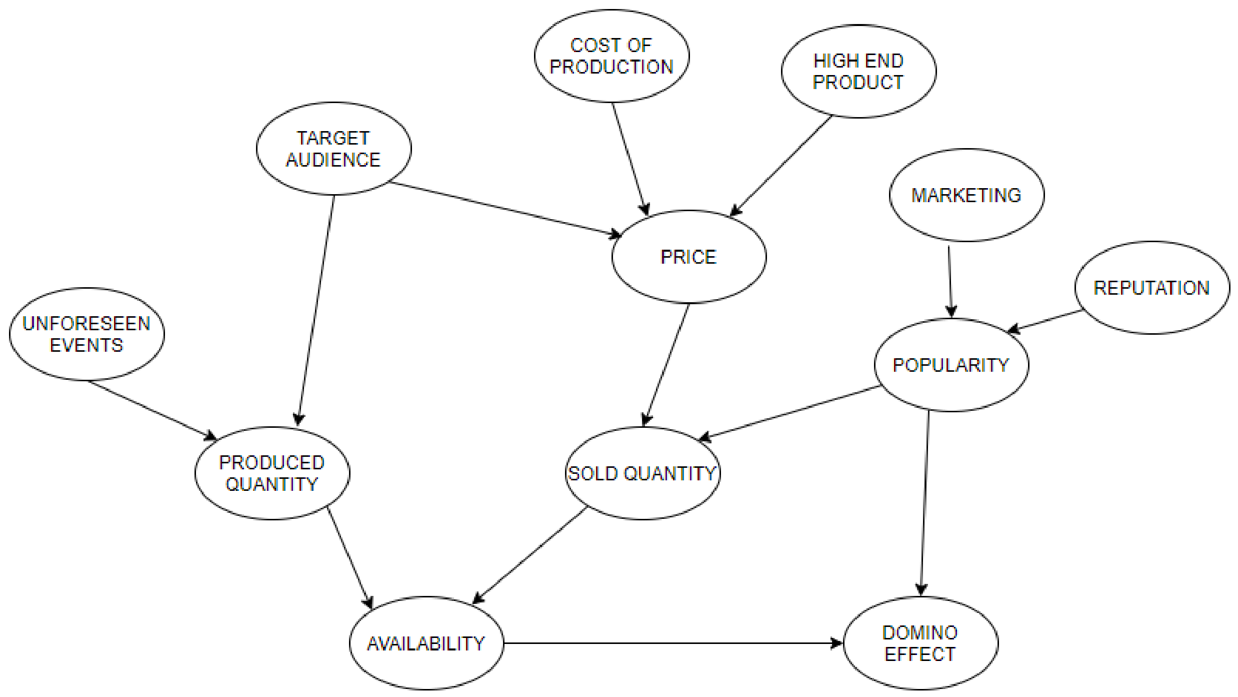
With this project I tried to model with a Bayesian Network the behavior of the market with respect to the launch of a new product and in particular to predict its availability at the stores after its release. The use of a Bayesian network allowed me to deal with uncertainty and partial information, and to make probabilistic predictions by defining probabilities of different causes that could affect my target in a very simplified model of my domain of interest.

The model

I used 12 discrete variables to represent the domain :

- ***Unforeseen Events:*** [*Yes / No*] It accounts for the possibility that something unexpected happens that affects the quantity of product that can be produced, like a supply chain problem or the shortage of some component/material. It has 2 possible states stating if some unplanned event happened or not.
- ***Target Audience:*** [*Mass /* Niche] It expresses whether the product is directed towards a mass or a niche market.
- ***Cost of Production:*** [*Low / High*] How much the product is going to cost to be produced or bought from the producer if the seller does not manufacture it.
- ***High End Product:*** [*Yes / No*] Boolean variable which indicates if the product is a luxury good or not.
- ***Price:*** [*Low / High*] Selling price of the product.
- ***Marketing:*** [*Null / Low / High*] Amount of budget that the vendor decide to invest on promoting its product.
- ***Reputation:*** [*Normal / High*] Brand reputation of the vendor.
- ***Popularity:*** [*Null / High / Very High*] It indicates how much a product is popular among the consumers. Null here does not mean unpopular but neutral (non notable opinion).
- ***Sold Quantity:*** [*Low / Medium / High*] Volumes of product that have been sold after the release. It’s the evidence of the sales performance.
- ***Produced Quantity:*** [*Low / Medium / High*] Volumes of product that have been produced or put on the market by the seller.
- ***Availability:*** [*Out of stock / Keep up with demand / Surplus*] Availability of the product at the stores (online also).
- ***Domino Effect:*** [*Yes / No*] It addresses the possibility that a scarce availability of the product together with its high popularity makes the product even more difficult to find.

Network Topology



A Bayesian network has to be constructed as a directed acyclic graph in which the nodes represent variables of interest and the links (edges) represent causal influences among the variables.

As we can see, in this simplified world the availability of a product is directly influenced by the *Produced Quantity* and the *Sold Quantity*. The produced quantity is in turn considered to be dependent on the number of clients we want to reach (as well as on our capacity to accomplish that, if nothing unexpected happens, here modeled with the node '*Unforeseen Events*'), while the quantity that can be sold depends on the price and the popularity of the product. I also introduced the node '*Domino Effect'* since a Bayesian network should be an acyclic graph while I wanted to express the fact that not only the popularity indirectly influences the availability of the product, but also the opposite can be considered true. Indeed this concept can be leveraged by companies to make their product more appealing with very limited release (e.g. drop strategy).

Conditional Probability Tables

In order to make predictions with our model, we need to define the conditional probability distribution (CPD) of each variable and use them to fill the conditional probability tables (CPT). To define the CPTs, I gathered some data from the internet in order to come up with the conditional probabilities of each variable given its parents or with prior probabilities for nodes with no parents. Each node has a CPD associated with it, hence we I needed to define 12 CPDs in this case.

Here's an example of the CPT for the variable '*Produced Quantity*'

Unforeseen Events	Yes	Yes	No	No
Target Audience	Mass	Niche	Mass	Niche
Prod. Qty (Low)	0.4	0.9	0.01	0.7
Prod. Qty (Medium)	0.55	0.09	0.08	0.27
Prod. Qty (High)	0.05	0.01	0.91	0.03

Reasoning on Independence

We would like a method for probabilistic inference, that is the computation of posterior probabilities of some variables given observed evidence about others. Identified some random variables we could use the full joint distribution to answer to all possible queries. Since the full joint probability distribution specifies the probability of each complete assignment of values to random variables, it does not scale well: for a domain described by n Boolean variables, it requires an input table of size $O(2^n)$. For this reason the full joint distribution is just not a practical tool for building reasoning systems.

The notion of independence is a key in the power of Bayesian networks. They can in fact represent essentially any full joint probability distribution very concisely if each node is conditionally independent of its other predecessors given its parents.

Independence assertions are usually based on knowledge of the domain. They can dramatically reduce the amount of information necessary to specify the full joint distribution and so the complexity of the inference problem. Unfortunately, a clean separation of entire sets of variables by independence is quite rare. Bayesian networks leverage conditional independence between variables, given other variables. Conditional independence assertions are brought by direct causal relationships in the domain and can allow probabilistic systems to scale up; moreover, they are much more commonly available than absolute independence assertions.

Another important independence property is implied by the topology of the network semantics: a node is conditionally independent of all other nodes in the network, given its parents, children, and children's parents (given its Markov blanket).

Here we can see the Markov blanket of the node '*Sold Quantity*'

```
model.get_markov_blanket('Sold Qty')
```

```
['Produced Qty', 'Price', 'Availability', 'Popularity']
```

The d-separation principle allows us to determine whether a set X of variables is independent of another set Y, given a third set Z. With the library pgmpy we can check if two variables are d-separated by looking for an active trail between the two, along the edges of the network. If there is no active trail between the two variables, they are d-separated.

```
print(model.is_active_trail('Marketing','Availability'))
print(model.is_active_trail('Marketing','Availability',observed=['Popularity']))
```

```
True
False
```

In the first case we can see that the availability of a product in the stores is not independent from the marketing budget the seller spends to promote it, but it becomes independent from it knowing the popularity among the public of the product itself since marketing only influences the available quantity through it.

Considering instead the *V-structure* determined by the variables 'Cost of Production' and 'High End Product', we can see that they cease to be d-separated as soon as we gather evidence about the price of the product.

```
print(model.is_active_trail('Cost of Production','High End Product'))
print(model.is_active_trail('Cost of Production','High End Product',observed=['Price']))
```

```
False
True
```

Inference

There are two possible inference methods: *exact inference* and *approximate inference*. The former computes analytically the posterior probability distribution for a set of query variables, given some observed event, the latter approximates the posterior probability distribution in some way, often with sampling methods.

To perform exact inference, I used the Variable Elimination algorithm that pgmpy features, which is more efficient with respect to Inference by Enumeration.

Bayesian networks are very useful for making predictions, following the relationship between variables in the causal direction, as well as for diagnostic reasons, when we perceive as evidence some effect that we want to explain by determining its probable cause.

Some interesting results for both cases are shown below

```
query1 = exact_inference.query(['Availability'],{'Produced Qty':'Low','Marketing':'High','Price':'Low'})
print(query1)
```

```
+---------------------------------+---------------------+
| Availability                    |   phi(Availability) |
+=================================+=====================+
| Availability(Out of stock )     |              0.7884 |
+---------------------------------+---------------------+
| Availability(Keep up with demand) |            0.1962 |
+---------------------------------+---------------------+
| Availability(Surplus)           |              0.0153 |
+---------------------------------+---------------------+
```

[query1] Here we see that if the vendor decides to produce a small amount of product and to sell it at a low price while promoting it, chances are very good (~ 80%) that the product is going to be sold out.

```
query2 = exact_inference.query(['Sold Qty'],{'Marketing':'Low','Reputation':'Normal'})
query3 = exact_inference.query(['Sold Qty'],{'Marketing':'High','Reputation':'Normal'})
print(query2)
print(query3)
```

```
+------------------+-----------------+ +------------------+-----------------+
| Sold Qty         |   phi(Sold Qty) | | Sold Qty         |   phi(Sold Qty) |
+==================+=================+ +==================+=================+
| Sold Qty(Low)    |          0.4347 | | Sold Qty(Low)    |          0.1559 |
+------------------+-----------------+ +------------------+-----------------+
| Sold Qty(Medium) |          0.4293 | | Sold Qty(Medium) |          0.3628 |
+------------------+-----------------+ +------------------+-----------------+
| Sold Qty(High)   |          0.1361 | | Sold Qty(High)   |          0.4813 |
+------------------+-----------------+ +------------------+-----------------+
```

[query2/3] From the above example we can see a very interesting result that shows the importance of marketing for a product when the vendor is not well known among the customers, while a very popular company can spend less on promoting it and still be able to sell an high quantity with a good probability.

```
query4 = exact_inference.query(['Price'],{'Availability':'Surplus','Target Audience':'Niche'})
print(query4)
```

```
+-------------+--------------+
| Price       |   phi(Price) |
+=============+==============+
| Price(Low)  |       0.1865 |
+-------------+--------------+
| Price(High) |       0.8135 |
+-------------+--------------+
```

[query4] This query shows how the probable cause (81%) for a surplus of products in the stores, that is targeted to a niche market, is a too high price.

```
query5 = exact_inference.map_query(['Reputation'],{'Price':'High','Availability':'Out of stock','Produced Qty':'High'})
print(query5)
```

```
{'Reputation': 'High'}
```

[query5] With the method *map_query* of pgmpy we can ask the most likely explanation for some evidence. In this case, if we know that an expensive product is out of stock even if it was mass produced, we can conclude that probably the brand reputation of the vendor is high (this is for example what happens with Apple products)

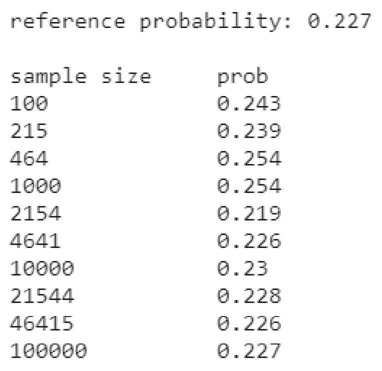
```
query6 = exact_inference.query(['Availability'],{'Target Audience':'Mass'})
query7 = exact_inference.query(['Availability'],{'Target Audience':'Mass','Unforeseen Events':'Yes'})
print(query6)
print(query7)
```

```
+---------------------------------+---------------------+ +---------------------------------+---------------------+
| Availability                    |   phi(Availability) | | Availability                    |   phi(Availability) |
+=================================+=====================+ +=================================+=====================+
| Availability(Out of stock)      |              0.1328 | | Availability(Out of stock)      |              0.4308 |
+---------------------------------+---------------------+ +---------------------------------+---------------------+
| Availability(Keep up with demand) |            0.3245 | | Availability(Keep up with demand) |            0.4545 |
+---------------------------------+---------------------+ +---------------------------------+---------------------+
| Availability(Surplus)           |              0.5428 | | Availability(Surplus)           |              0.1147 |
+---------------------------------+---------------------+ +---------------------------------+---------------------+
```

[query6/7] In these two last examples we can clearly see how the probability of a product going out of stock increases drastically (from 13% to nearly 45%) if we add the evidence that an unpredictable event actually happened, changing the amount of product that can be produced.

Sometimes performing exact inference is not really possible due, for instance, to very large network with many nodes. Approximate inference addresses this issue by approximating the posterior probability distribution with a sampling method. I will now show an example of an approximate inference with the Weighted Likelihood sampling method and I will compare it to the result of the exact inference using Variable Elimination on the same query.

I queried the network to get P(Domino Effect = No | Marketing = High, Produced Qty = Low) and run 10 experiments with an increasing number of samples.



```
reference probability: 0.227

sample size     prob
100             0.243
215             0.239
464             0.254
1000            0.254
2154            0.219
4641            0.226
10000           0.23
21544           0.228
46415           0.226
100000          0.227
```

As we can see, the approximate probability progressively converges to the reference value and get to the same result with 10000 samples.