Predicting videogames sales through Bayesian reasoning

Diego Chinellato diego.chinellato@studio.unibo.it
Alma Mater Studiorum - University of Bologna
Fundamentals of AI and KR course
Module 3 project report

Abstract

Financial data, such as data representing sales of a certain asset in a given economic sector, is intrinsically imbued with uncertainty, due to both the uncertainty of the financial market as well as the noise present in the data itself. In this report, probabilistic reasoning tools are applied to financial data concerning the sales in the videogames market. First, four different Bayesian Networks models are constructed either by explicitly defining the structure or by inferring it from the data. Then, for each model, parameters are learned from the available data. Finally, inference techniques are applied to perform probabilistic reasoning on the resulting model.

1. Introduction

In recent days, probabilistic inference has become a core technology within AI, mostly due to the design and development of powerful graph-based methods and data structures for representing and manipulating complex probability distributions (Pearl, 1988; Koller and Friedman, 2009; Russell and Norvig, 2021), called *probabilistic graphical models*. One important data structure belonging to this family of models is called Bayesian Network (BN). BNs are a compact representation of a set of a random variables (RVs) and the probabilistic influence (i.e., *conditional independence* relationships) they exert one one the other; in other words, BNs are a great tool when it comes to dealing with uncertainty and probabilistic environments. In this report, an application of BNs to a particular sector of the economic market (namely, the videogames sector) is described.

2. Mathematical preliminaries

The main mathematical tool underlying BNs is *probability theory*; in particular, the subjective (or Bayesian) interpretation of probability is adopted, i.e. probability is interpreted as reasonable expectation representing a state of knowledge or as quantification of a personal belief. To introduce BNs, first consider a probability model of a certain domain and its associated full joint probability distribution. In principle, the full joint distribution can be used to answer all possible questions about the domain; using the full joint distribution directly to answer probability queries, however, is most often unfeasible, as the the procedure is intractable in the worst case (Russell and Norvig, 2021). For a full joint distribution with n RVs and maximum arity d, the time as well as space complexity of inference by enumeration is $O(d^n)$, hence intractable (not considering the fact that, even if complexity allowed, it would still be very difficult most of the times to find all $O(d^n)$ entries required by the joint distribution table).

To overcome this issue, we can make use of two powerful relationships among RVs. One is called *absolute (marginal) independence*. Formally, two random variables X and Y are independent, denoted $\mathbf{P} \models (X \perp Y)$, if and only if:

$$\mathbf{P}(X|Y) = \mathbf{P}(X) \; or \; \mathbf{P}(Y|X) = \mathbf{P}(Y) \; or \; \mathbf{P}(X,Y) = \mathbf{P}(X)\mathbf{P}(Y) \tag{1}$$

All these forms are equivalent. Intuitively, if X and Y are (absolutely) independent then it means that the values that X takes are not influenced by the values taken by Y (and viceversa). Independence assertions can greatly reduce the size of the domain and the complexity of the inference problem; unfortunately, this type of "total" independence is very rare to have. A second, more common, type of independence among variables is called *conditional independence*. The general definition of conditional independence of two variables X and Y, given a third variable Z, denoted $\mathbf{P} \models (X \perp Y|Z)$ is

$$\mathbf{P}(X,Y|Z) = \mathbf{P}(X|Z)\mathbf{P}(Y|Z) \; or \; \mathbf{P}(X|Y,Z) = \mathbf{P}(X|Z) \; or \; \mathbf{P}(Y|X,Z) = \mathbf{P}(Y|Z) \tag{2}$$

All three forms are again equivalent. In most cases, the use of conditional independence reduces the size of the representation of the joint distribution from exponential in n to linear in n. Conditional independence is also the basis for Bayesian networks.

2.1 Bayesian Networks

Formally, a Bayesian network[1] is a directed acyclic graph (DAG) in which each node is annotated with quantitative probability information:

1. Each node corresponds to a random variable, either discrete or continuous.
2. A set of directed edges connects pairs of nodes. If there is an edge from node X to node Y, X is said to be a *parent* of Y.
3. Each node X_i has a *Conditional Probability Distribution* (CPD) $\mathbf{P}(X_i|Parents(X_i))$ that quantifies the effect of the parents on the node. For simplicity, we will consider BNs where the CDPs can be represented as tables, i.e. *Conditional Probability Tables* (CPT).

Bayesian networks can represent essentially any full joint distribution and in many cases can do so very concisely. The topology of the network - the set of nodes and links - specifies the conditional independence relationships that hold in the domain; the intuitive meaning of an edge from X to Y is usually that X has a *direct influence* on Y, i.e. causes should be parents of effects.

The process of building a BN, then, can be seen as a two-steps process: first, the network structure is defined, i.e. the set of nodes and the set of edges making up the DAG is determined; second, a set of parameters is determined for each node, i.e. the CPD associated to each node. Indeed, both steps can be "hard-coded" by explicitly providing the DAG components and/or the CDPs; another option would be to use a learning algorithm to partly or completely automatize the building process. As the experimental part involve both "hard-coded" as well as learned networks, a brief description of learning methods for BNs is given in the following two subsections.

1. Also called: belief network, probabilistic network, causal network, knowledge map, graphical model (Koller and Friedman, 2009).

2.1.1 Learning the structure

The problem of learning the structure of a BN, given a dataset of n points $\mathcal{D} = \{X_1, ..., X_n\}$, corresponds to the problem of determining the set of directed arcs E which make up the DAG $G = (V, E)$ (the set of nodes is of course already known), using a certain criterion to measure its quality. Indeed, this task is computationally very expensive due to the huge size of the search space of possible graphs, growing super-exponentially in the number of nodes n (Scanagatta et al., 2019).

One of the earliest approaches to this problem was presented in Chow and Liu (1968), in which the authors describe an algorithm for learning a tree-structured network for approximating a full joint distribution by decomposing it in a product of second-order conditional distributions (i.e., each variables has only one parent). Indeed, this early approach is quite simple, but several improvements and generalizations of the Chow-Liu tree have been proposed in recent years (Kovács and Szántai, 2010).

Nowadays, modern approaches for structure learning usually fall in one of two approaches, namely *score-based* and *constraint-based* approaches. With score-based structure learning, we are given a scoring function $score(G, \mathcal{D})$ which measures the fitness of the current graph G to the dataset $\mathcal{D}$, and then we need to solve the following optimization problem:

$$\arg\max_{G} score(G, \mathcal{D})$$

One of the most commonly used scoring functions is the Bayesian Dirichlet equivalent uniform (BDeu) (Heckerman et al., 1995), which computes the the posterior probability of a certain graph given the data, assuming a uniform prior probability over all possible DAGs. Another important metric is the Bayesian Information Criterion (BIC), which can be seen as an approximation of the BDeu (Scanagatta et al., 2019).

After defining the scoring function, a number of different methods can be used to perform the search according to the scoring function ad select the (local) optimal DAG; for example, we can employ a local-search based algorithm such as Hill Climb Search, which iteratively improves a local solution until it reaches a local optima. It's worth noting that most algorithms to perform structure learning are metaheuristics algorithms with no guarantee on the quality of the resulting network; an exact algorithm has been proposed in Korhonen and Parviainen (2013), running in time $O(3^n n^k)$ for n nodes (with k hyperparameter), i.e. exponential in the number of nodes.

Constraint-based approaches are based on statistical tests: they perform several conditional hypothesis tests to learn the independence relations among the variables, and then use this relations to build the graph. The SOTA of constraint-based approaches is the PC algorithm (Scanagatta et al., 2019).

2.1.2 Learning the parameters

Once we have the graph structure, we need to learn the parameters of the network, i.e. the CDP associated to each node. Algorithms for parameter learning for BNs usually fall in one of two categories, based on whether they are suitable for complete data or incomplete data. Concerning the first category, one common approach is to compute the probabilities as the relative frequencies of the possible values via a Maximum Likelihood Estimation

(MLE) procedure. MLE in this context works as follows: we are given a BN parametrized by parameters θ (where each θ_i represents the CPT associated with variable X_i in the network) and a *(global) likelihood function* $\mathcal{L}(\theta|\mathcal{D})$, defined as

$$\mathcal{L}(\theta|\mathcal{D}) = \prod_i \ell_i(\theta_i|\mathcal{D})$$

which measures the "quality" of parameters θ according to data $\mathcal{D}$; each $\ell_i(\theta_i|\mathcal{D})$ is a *conditional (local) likelihood* defined as:

$$\ell_i(\theta_i|\mathcal{D}) = \prod_m \mathbf{P}(x_i = m|Parents(X_i) = m, \theta_i)$$

Then, goal of MLE is to determine the set of parameters θ by solving

$$\hat{\theta} = \arg\max_\theta \mathcal{L}(\theta|\mathcal{D})$$

In practice, for CPTs this is done by computing the state counts and dividing by the conditional sample size: given a variable X_i, then we have one parameter $\theta_{x|u}$ for each combination of values from X_i and $Parents(X_i)$, i.e. $\theta_i = \{\theta_{x|u}\}\ \forall x \in Values(X_i)$, $\forall u \in Values(Parents(X_i))$. Then, the ML estimator can be derived directly as $\theta_{x|u} = \frac{M(u,x)}{\sum_x M(u,x)}$, where $M(u,x)$ is the number of times variables X_i takes value x with parent value u (Koller and Friedman, 2009).

2.2 Inference

The basic task for a probabilistic inference system is to compute the posterior probability distribution for a set of query variables $\mathbf{X}$ given a set of observed variables $\mathbf{E}$ - i.e. *conditional probability queries* of the form $\mathbf{P}(X|\mathbf{E} = \mathbf{e})$. Note that such queries can in principle be answered by only summing out terms from the full joint distribution, i.e. $\mathbf{P}(X|\mathbf{e}) = \alpha\mathbf{P}(X, \mathbf{e}) = \alpha \sum_y \mathbf{P}(X, \mathbf{e}, \mathbf{y})$, where $\mathbf{Y}$ are the hidden (nonevidence, nonquery) variables.

The above approach (called inference by enumeration) is not efficient as its time complexity is $O(d^n)$ for a network of n variables and maximum arity d. Still, this algorithm can be improved by eliminating repeated calculations, using a very simple idea: do the calculation once and save the results for later use (dynamic programming). This approach is called variable elimination algorithm: it works by evaluating expressions in right-to-left order, intermediate results are stored, and summations over each variable are done only for those portions of the expression that depend on the variable. However, the efficiency of the variable elimination algorithm is strongly dependant on the network structure: for polytrees (also called singly-connected networks)[2], time as well space complexity of variable elimination is $O(d^k n)$, i.e. linear in the number of nodes. In the general case of multiply connected networks, variable elimination has again an exponential complexity, both in time and in space. Note that there are also approximate inference techniques which allow for efficient inference independently on the network structure.

2. A polytree is a DAG in which there is at most one undirected path between any two nodes in the graph, or equivalently a DAG whose underlying undirected graph is a tree.

Feature	Description	Notes
Platform	Platform of the game release	String, 31 unique values
Year	Year of the game release	String, 40 unique values
Genre	Genre of the game	String, 13 unique values
Developer	Developer of the game	String, 1697 unique values
Publisher	Publisher of the game	String, 582 unique values
Critic Score	Aggregated score compiled by Metacritic staff	Float, range: [0,100]
User Score	Score by Metacritic's subscribers	Float, range: [0, 10]
NA Sales	Sales in North America (millions)	Float, range: [0, 42]
EU Sales	Sales in Europe (millions)	Float, range: [0, 29]
JP Sales	Sales in Japan (millions)	Float, range: [0, 10]
Other Sales	Sales in the rest of the world (millions)	Float, range: [0, 11]
Global Sales	Total worldwide sales (millions)	Float, range: [0.1, 83]

Table 1: Description of the main features available in the dataset. Other features were present but not used in the definition of the networks.

3. Experiments

3.1 Dataset

The dataset considered for the experimental part is available at this link. It consists on data about videogames who sold at least 100000 copies (approx. 16000 records); there are game-specific features such as the genre or the platform on which it was release, ratings from both users and critics and sales data, divided by region. The main features are described in Table 1. To define and query the network, the `pgmpy` Python library was used.

Before defining (or learning) the networks, data has to be processed for several reasons. First, some features have been removed as they're not really helpful, such as the user and critic count which pretty much depend on the website from which the data was scraped from, or the name of the game. Second, `pgmpy` unfortunately does not support learning of continuous variables, so all the non-discrete features in the dataset need to be discretized as well. Third, missing values have also been removed (MLE only works with complete data). Finally, some data points representing for instance publishers or developers with a low number of games in the dataset were removed. Since there is already a huge number of unique values in these variables (which will, in turn, result in very big CDTs), this operation helps in reducing the number of unique values and thus in having slightly smaller CDTs and to ease the general computational cost of the subsequent algorithms, while only slightly impacting the performances of the resulting networks. After preprocessing, the data consists of about 6000 records of 12 features each.

3.2 Networks definition

Four different Bayesian Networks were defined. In particular, the first model (named `custom` model) was defined by explicitly providing the set of edges that make up the network. A

graphical representation of this network can be seen in Figure 1. A rationale of all the edges considered would be too lengthy, so only the intuition for some edges is given:

- $Year \rightarrow Platform$: as time passes, some platforms become outdated, and are progressively less likely to be chosen to produce a game (up to the point when they are dismissed); moreover, certain platforms are more popular in certain years, e.g. the platform release year or when highly awaited games are released.
- $Platform \rightarrow Genre$: certain platforms are simply better suited for certain genres than others; e.g. there are very few shooter games for platforms such as DS and 3DS.
- $Publisher \rightarrow (EU/NA/JP/Other)\_Sales$: the publisher is the entity that deals with marketing, promoting, distributing and ultimately selling the game, hence it is quite natural to think that it has a huge influence on the overall sales.

Also, note that this network was developed with the intent of having an polytree to be used for efficient (exact) inference. Unfortunately, the resulting network is not a polytree due to the position of the "Genre" node, but removing that node would result in a too big information loss; moreover, at runtime the network appears to be quite efficient for inference as well.

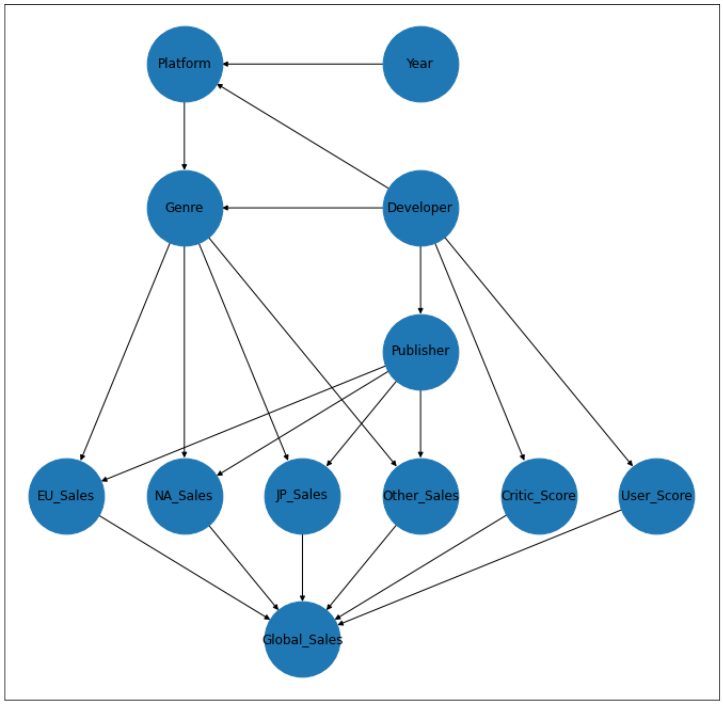

Figure 1: The `custom` network.

Three more networks have been defined by means of structure learning. In particular, one network (named `tree` model) was learned through a TreeSearch procedure based on the Chow-Liu Tree algorithm (Chow and Liu, 1968) with root node "Year". The resulting network can be seen in Figure 2.

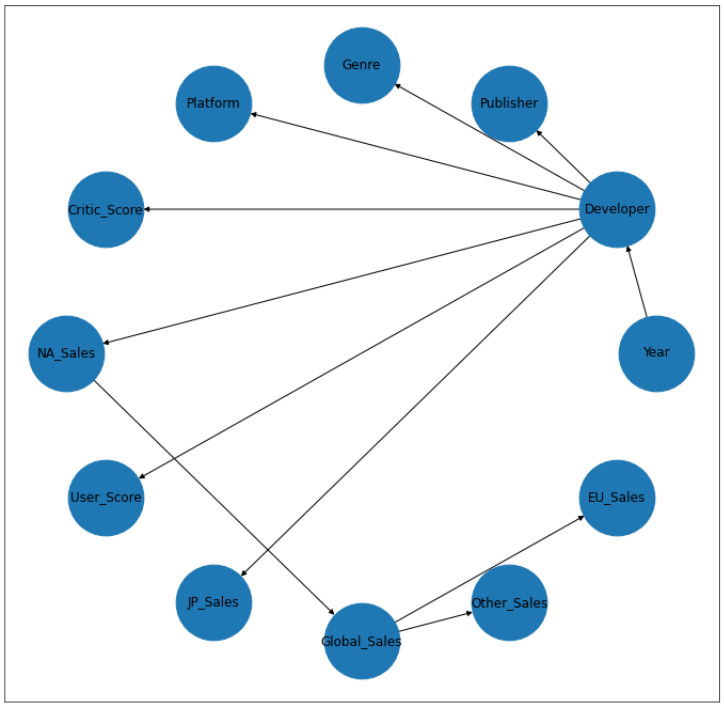

Figure 2: The `tree` network.

Indeed, the `tree` network is a polytree, hence it allows for efficient exact inference. However, we can also notice a number of issues, namely the fact that some edges do not really make sense, e.g. the causal links going from "Global_Sales" to "EU_Sales" and "Other_Sales" (rather, it's the other way around), or the link from "Year" to "Developer". It's also interesting to note that, according to this model, the "Developer" variable is in a sense the most "central" variable in the network.

Finally, the other two networks where learned following a score-based approach. The adopted scoring function is BDeu, while the search strategy used is Hill Climb search, a local search strategy (the simplest one) which involves choosing a solution and then iteratively improve it until it can't be improved any longer (i.e., a local optima). The first learned network, named `hc_base` network, can be seen in Figure 3.

Similarly to the previous network, the `hc_base` network also has a number of issues: not only we have again edges that do not really make sense (e.g. $Global\_Sales \rightarrow NA\_Sales$, $Global\_Sales \rightarrow Critic\_Score$ or $JP\_Sales \rightarrow User\_Score$), on top of that the resulting network is also a multiply connected network.

To partly overcome this issues, we can try to inject some "prior knowledge" into the learning algorithm. With `pgmpy` it is possible to specify several hyperparameters to control the Hill Climb search process, such as a list of fixed edges, i.e. a list of edges that will always be there in the final learned model, or a black list, i.e. a list of edges that are excluded from the search and so will never appear in the final model. We consider as fixed edges the following set:

```
{(NA_Sales, Global_Sales), (EU_Sales, Global_Sales), (JP_Sales, Global_Sales),
(Other_Sales, Global_Sales)}
```

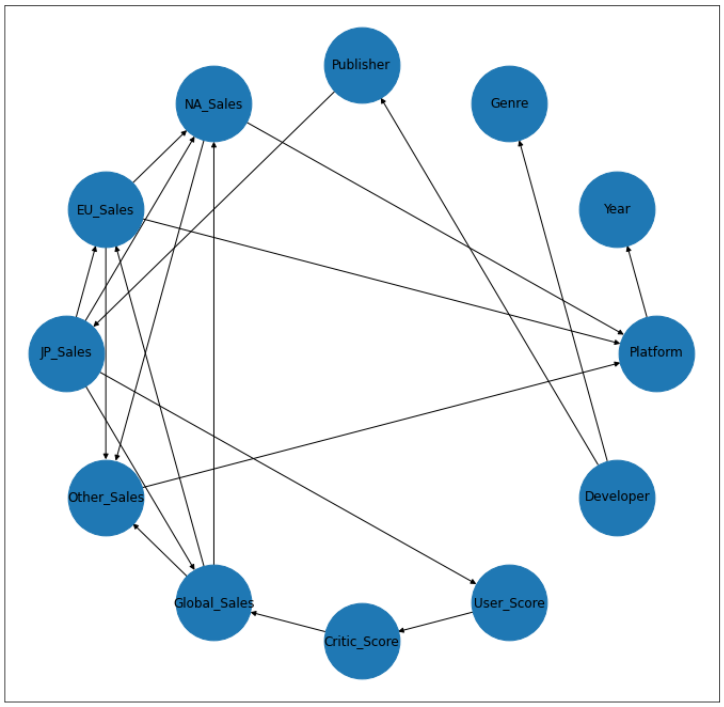

Figure 3: The `hc_base` network.

while as black list we consider the set:

{(Platform, Year), (Year, Year), (Genre, Year), (Publisher, Year), (NA_Sales, Year), (EU_Sales, Year), (JP_Sales, Year), (Other_Sales, Year), (Global_Sales, Year), (Critic_Score, Year), (User_Score, Year), (Developer, Year), (User_Score, Critic_Score), (Critic_Score, User_Score), (Publisher, Developer), (EU_Sales, User_Score), (EU_Sales, Critic_Score), (NA_Sales, User_Score), (NA_Sales, Critic_Score), (JP_Sales, User_Score), (JP_Sales, Critic_Score), (Other_Sales, User_Score), (Other_Sales, Critic_Score), (Global_Sales, User_Score), (Global_Sales, Critic_Score)}.

The network learned this way, named `hc_constrained` network, can be seen in Figure 4. Again, some of the issues that were highlighted with the previously learned networks are still here (e.g. notice the edge from `Publisher` to `JP_Sales` only), although we can see some improvements over the `hc base` model (mostly thanks to the fixed edges and black list).

After defining the four networks, a MLE procedure was used to estimate the CDTs for all of them[3].

3.3 Probabilistic reasoning

Having defined the networks, we can now use them to perform probabilistic reasoning (i.e. inference). We consider three different questions that we may want to ask; the first one is:

3. CDTs are not reported here due to space constraints.

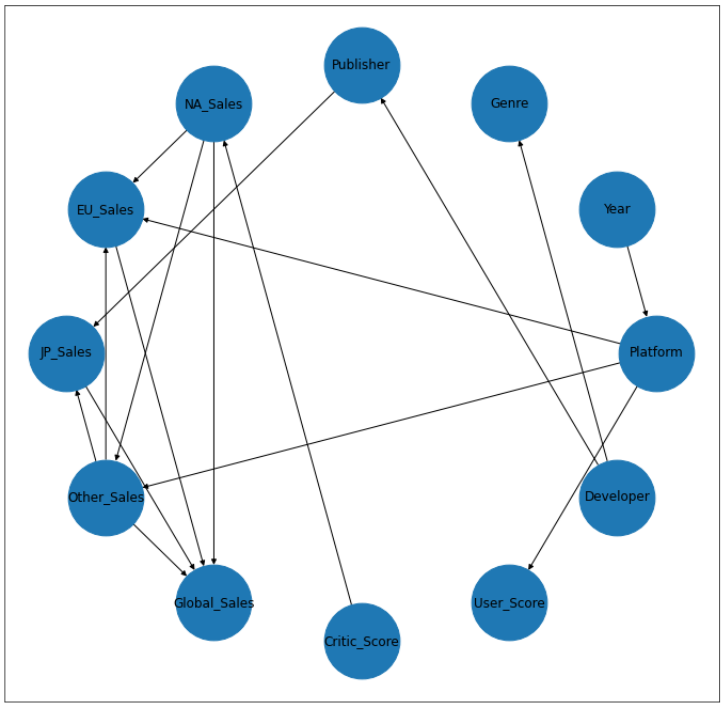

Figure 4: The `hc_constrained` network.

- How do the two different scores (critics and users) affect global sales? Which one is more impactful?

To answer this, we perform a set of queries of the form

$$\mathbf{P}(Global\_Sales \,|\, Critic\_Score = c,\, User\_Score = u)$$

where the values of $(c, u) \in \{(4,0), (3,1), (2,2), (1,3), (0,4)\}$. Results of this set of queries are shown in Figure 5.

The four models produce different results in terms of absolute values of the resulting probability distribution; in particular, the `custom` model produces more "uniform" distributions compared to the more "pointy" distributions of the `HC` models, with the `tree` model somewhere in between. Still, there is a clear pattern emerging from the scenario, namely the fact that an higher critic score seems to be more important in obtaining an higher volume of sales.

A second question one might consider is the following:

- Which platforms should a certain developer (e.g. Ubisoft) choose to develop a game in 2016?

To answer this, again we execute a set of queries of the form

$$\mathbf{P}(Global\_Sales \,|\, Developer = d,\, Year = 2016,\, Publisher = u,\, Platform = p)$$

where $p \in \{\text{“}PS4\text{”}, \text{“}PS3\text{”}, \text{“}PS2\text{”}, \text{“}PC\text{”}, \text{“}XOne\text{”}, \text{“}X360\text{”}, \text{“}3DS\text{”}\}$, and d, u are fixed values (in my experiment, $d = u = \text{“}Ubisoft\text{”}$). Results of this set of queries are shown in Figure 6.

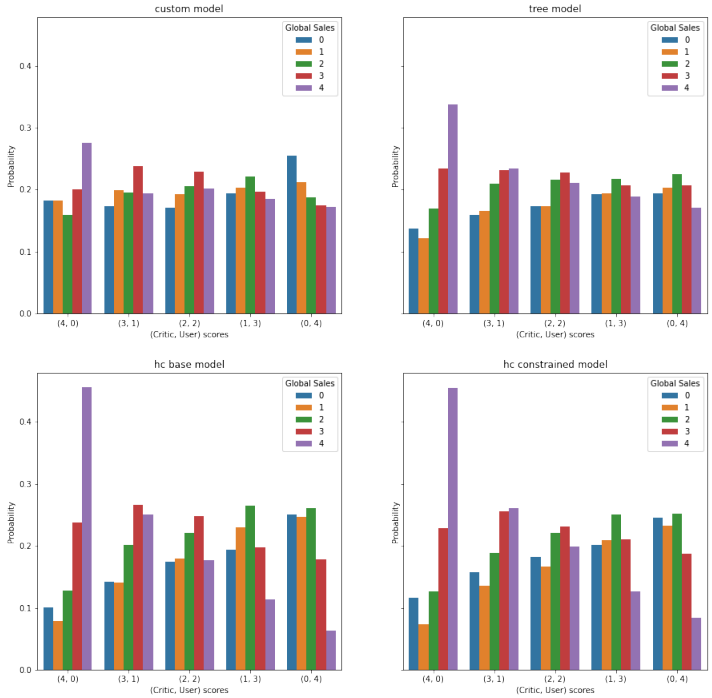

Figure 5: Probability distributions resulting from the first query.

Again, the four models produce different results. The `custom` model suggests that it's a good idea for the developer to choose PS4 and XOne as platforms in 2016, an insight that is confirmed by the `HC base` model and partly (only for PS4) by the `HC constrained` model - and also intuitively it makes sense given that they were the newest platforms available in 2016. Moreover, we notice a lot of disagreement among the different models for the PC platform, with `HC base` suggesting it would be a flop whereas `HC constrained` suggests it would be a moderate success. This disagreement can maybe be explained by the fact that there is a lot uncertainty associated with this particular query - something that is highlighted by the almost-uniform distribution returned by the custom model. Finally, we can notice that the `tree` model, for this set of queries, always returns the same distribution. This is due to the structure of the network, which implies the following conditional independence relationship: $(Global\_Sales \perp Platform | Developer)$ (certainly, an unrealistic assumption).

Finally, a last question considered is the following:

- Which game genre tends to result in higher critic scores for a certain platform?

To answer this, again we execute a set of queries of the form

$$\mathbf{P}(Critic\_Scores \,|\, Genre = g,\, Platform = p)$$

where $g \in \{$"$Sports$", "$Racing$", "$Platform$", "$Misc$", "$Action$", "$Puzzle$", "$Shooter$", "$Simulation$", "$Role-Playing$", "$Adventure$", $\}$, and p is a fixed platform (in my experiment, $p =$ "PC"). Results of this set of queries are shown in Figure 7.

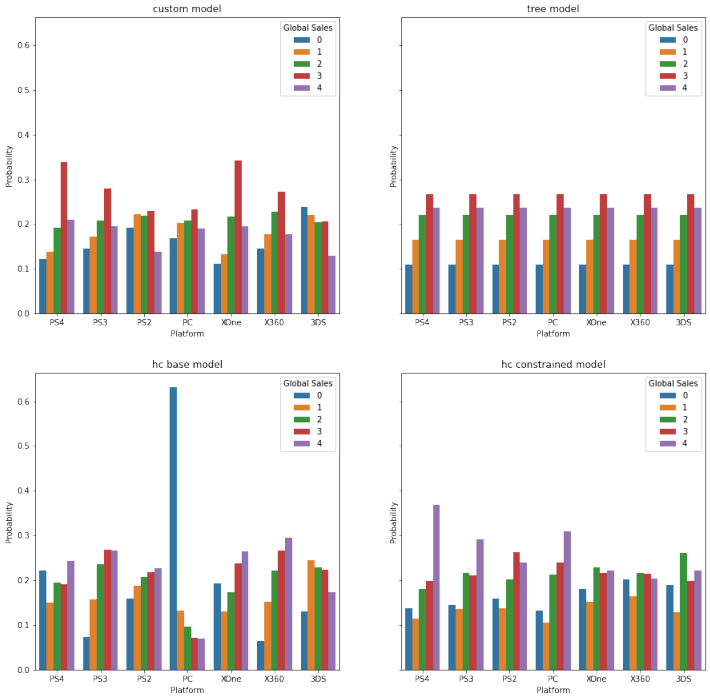

Figure 6: Probability distributions resulting from the second query.

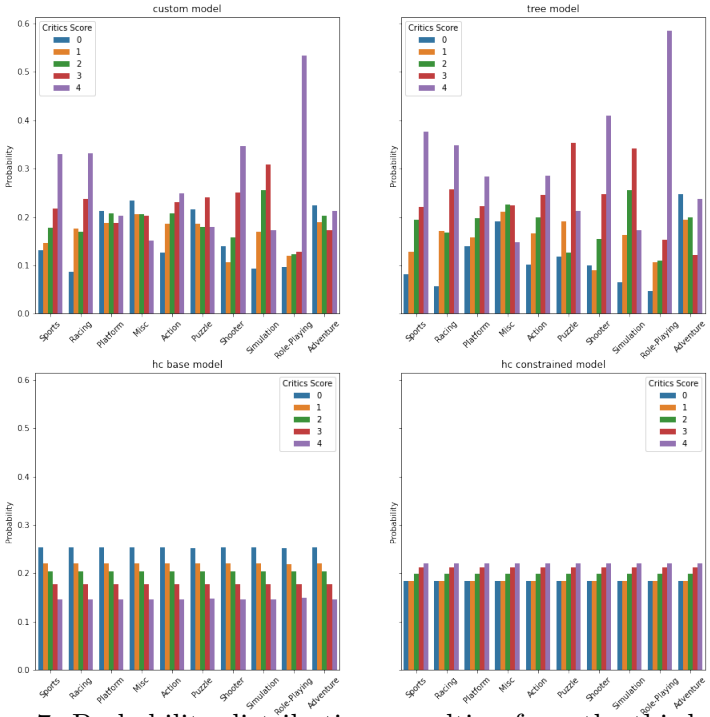

Figure 7: Probability distributions resulting from the third query.

The `custom` model and the `tree` model produce very similar results, with Role-Playing games being the most likely to receive high critic scores for PC platform, followed by Shooter, Sports and Racing games. Moreover, similarly to what happened with previous queries to

the tree model, the `HC` models return always the same distribution independently on the value of Genre, so we expect that these two models contain a independence relationship ($Critic\_Scores \perp Genre|Platform$). Strangely, we can see that this independence holds only for the `HC constrained` model, and not for the `HC base`, so the result is most likely due to the poor quality of that model.

4. Conclusion

In this report, we have seen that Bayesian Networks are indeed a very useful tool for concisely representing complex probabilistic domains, such as financial domains. BNs can be either custom-defined or learned by available data, although we have seen that current (open-source) structure learning methods are still lacking in terms of quality of the learned graph. Parameter learning, on the other hand, appears to be much more reliable and better performing. After defining structure and parameters, the BN can be used to perform inference - i.e. answer probabilistic queries about the domain - both exactly or approximately.

References

CKCN Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

David Heckerman, Dan Geiger, and David M Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243, 1995.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Janne Korhonen and Pekka Parviainen. Exact learning of bounded tree-width bayesian networks. In *Artificial Intelligence and Statistics*, pages 370–378. PMLR, 2013.

Edith Kovács and Tamás Szántai. On the approximation of a discrete multivariate probability distribution using the new concept of t-cherry junction tree. In *Coping with Uncertainty*, pages 39–56. Springer, 2010.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman Publishers, San Mateo, CA, 1988.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 4rd edition, 2021.

Mauro Scanagatta, Antonio Salmerón, and Fabio Stella. A survey on bayesian network structure learning from data. *Progress in Artificial Intelligence*, 8(4):425–439, 2019.