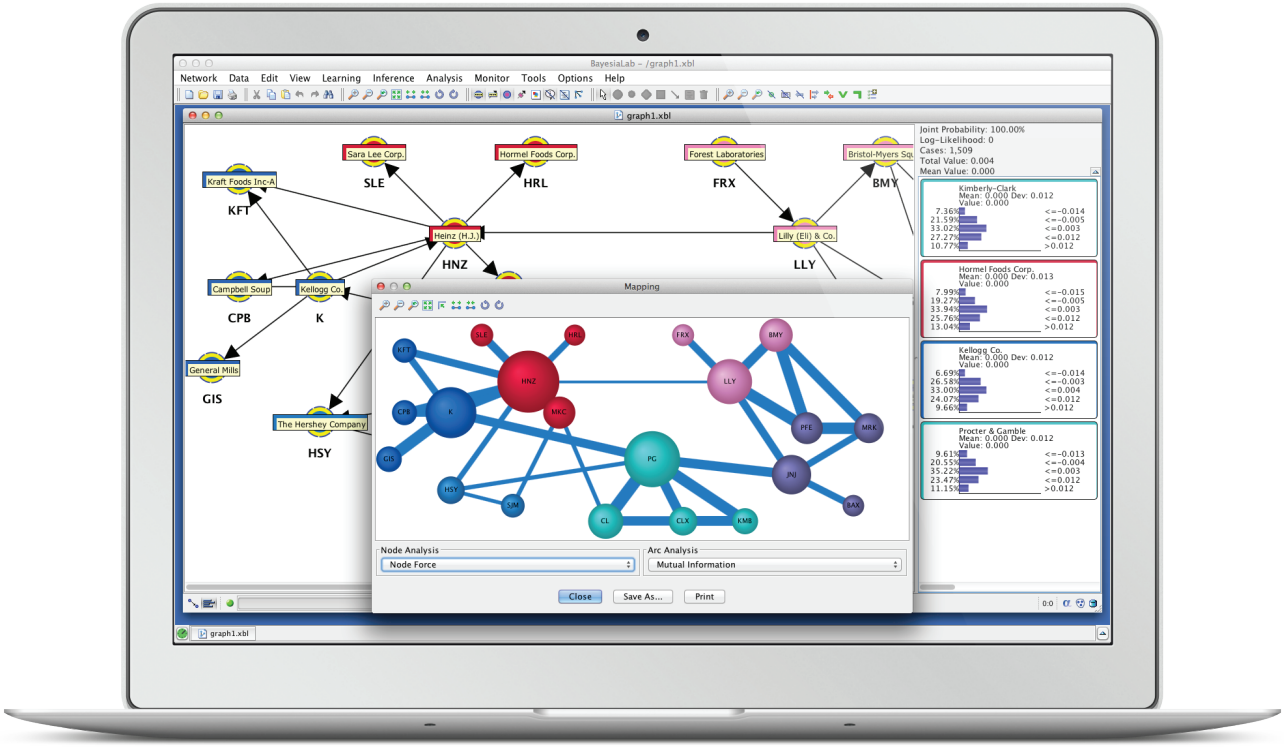
BAYESIA

Bayesian Networks & BayesiaLab

STEFAN CONRADY | LIONEL JOUFFE



A Practical Introduction for Researchers

bayesia.us • bayesia.com • bayesia.sg

Bayesian Networks & BayesiaLab
A Practical Introduction for Researchers

Lionel Jouffe Stefan Conrady

bayesia.us bayesia.com bayesia.sg

Contents

Overview of Book

Preface

While Bayesian networks have flourished in academia over the past three decades, their adoption in applied research has progressed more slowly. A key reason has been the difficulty of constructing Bayesian networks for practical use. For many years, researchers had to develop their own software to work with Bayesian networks, rendering the methodology inaccessible to most scientists.

The release of BayesiaLab 1.0 in 2002 marked a turning point. Developed by a newly founded French company, BayesiaLab was created specifically to address this challenge. Led by Dr. Lionel Jouffe and Dr. Paul Munteanu, the development team designed BayesiaLab with research practitioners in mind–not just computer scientists. This practitioner orientation is reflected most clearly in BayesiaLab's graphical user interface, which enables users to interact directly with Bayesian networks as graphs, rather than through computer code.

At the time of writing, BayesiaLab is approaching its sixth major release and has evolved into a comprehensive software platform–a full "laboratory" for exploring a wide range of research questions.

Still, the point-and-click convenience of BayesiaLab does not exempt researchers from understanding the fundamentals of Bayesian networks. In fact, as BayesiaLab has made these models accessible to a much wider audience, the demand for effective training has grown significantly. We recognized the need for a book that enables self-guided learning–a resource that introduces both the theory of Bayesian networks and the practical use of BayesiaLab.

This book reflects the inherently visual nature of Bayesian networks. Hundreds of illustrations and screenshots provide tutorial-style guidance on BayesiaLab's core features. Key steps are shown repeatedly in different contexts to reinforce understanding. Our goal is to offer the reader a clear, step-by-step path from theoretical principles to practical implementation in BayesiaLab.

The foundations of the Bayesian network formalism span multiple disciplines, including computer science, probability theory, information theory, logic, machine learning, and statistics. Likewise, their applications extend across nearly all fields. As a result, the examples in this book draw from a variety of domains, demonstrating how each connects to the Bayesian network paradigm.

Ultimately, our aim is twofold: to reveal the theoretical power of Bayesian networks and to teach BayesiaLab as the platform that enables their practical application.

Structure of the Book

Part 1

The three short chapters in Part 1 are designed to provide foundational familiarity with Bayesian networks and BayesiaLab. After completing this section, readers should feel equipped to explore any of the subsequent chapters. Part 1 may also serve as an executive summary for those seeking a high-level introduction to the field.

- **Chapter 1**
- **Chapter 2**
- **Chapter 3**

Part 2

The chapters in Part 2 are mostly self-contained tutorials and may be studied in any order. However, beginning with Chapter 8, a working knowledge of BayesiaLab's core functions is assumed.

- **Chapter 4**
- **Chapter 5**
- **Chapter 6**
- **Chapter 7**
- **Chapter 8**
- **Chapter 9**
- **Chapter 10**
- **Chapter 11**

Notation

- Product features and special terms are written in **bold title/sentence case**, e.g., **Data Import Wizard**, **Parameter Estimation**, **Entropy**, and **Mutual Information**.
- Variable symbols are written in math notation. For multi-character variable or state names, use *...* with escaped spaces when needed, e.g., X, $Factor_0$, *Bicep Girth*, and $X = True$.
- Hyperlinks use descriptive link text, e.g., BayesiaLab User Guide.
- Menu paths are shown with breadcrumb notation and code formatting, e.g., `Main Menu > Learning > Supervised Learning > Markov Blanket`.
- Inline code formatting (e.g., `Ctrl+S`) is used for exact menu items, commands, interface labels, and keyboard shortcuts.

Chapter 1: Introduction

With Professor Judea Pearl receiving the prestigious 2011 A.M. Turing Award, Bayesian networks have presumably received more public recognition than ever before. Judea Pearl's achievement of establishing Bayesian networks as a new paradigm is fittingly summarized by Stuart Russell (2011):

> "[Judea Pearl] is credited with the invention of Bayesian networks, a mathematical formalism for defining complex probability models, as well as the principal algorithms used for inference in these models. This work not only revolutionized the field of artificial intelligence but also became an important tool for many other branches of engineering and the natural sciences. He later created a mathematical framework for causal inference that has had significant impact in the social sciences."

All Roads Lead to Bayesian Networks

There are numerous ways we could provide motivation for using Bayesian networks. A selection of quotes illustrates that we could approach Bayesian networks from many different perspectives, such as machine learning, probability theory, or knowledge management.

> "Bayesian networks are as important to AI and machine learning as Boolean circuits are to computer science." (Stuart Russell in Darwiche, 2009)

> "Bayesian networks are to probability calculus what spreadsheets are for arithmetic." (Conrady and Jouffe, 2015)

> "Currently, Bayesian Networks have become one of the most complete, self-sustained and coherent formalisms used for knowledge acquisition, representation and application through computer systems." (Bouhamed et al., 2015)

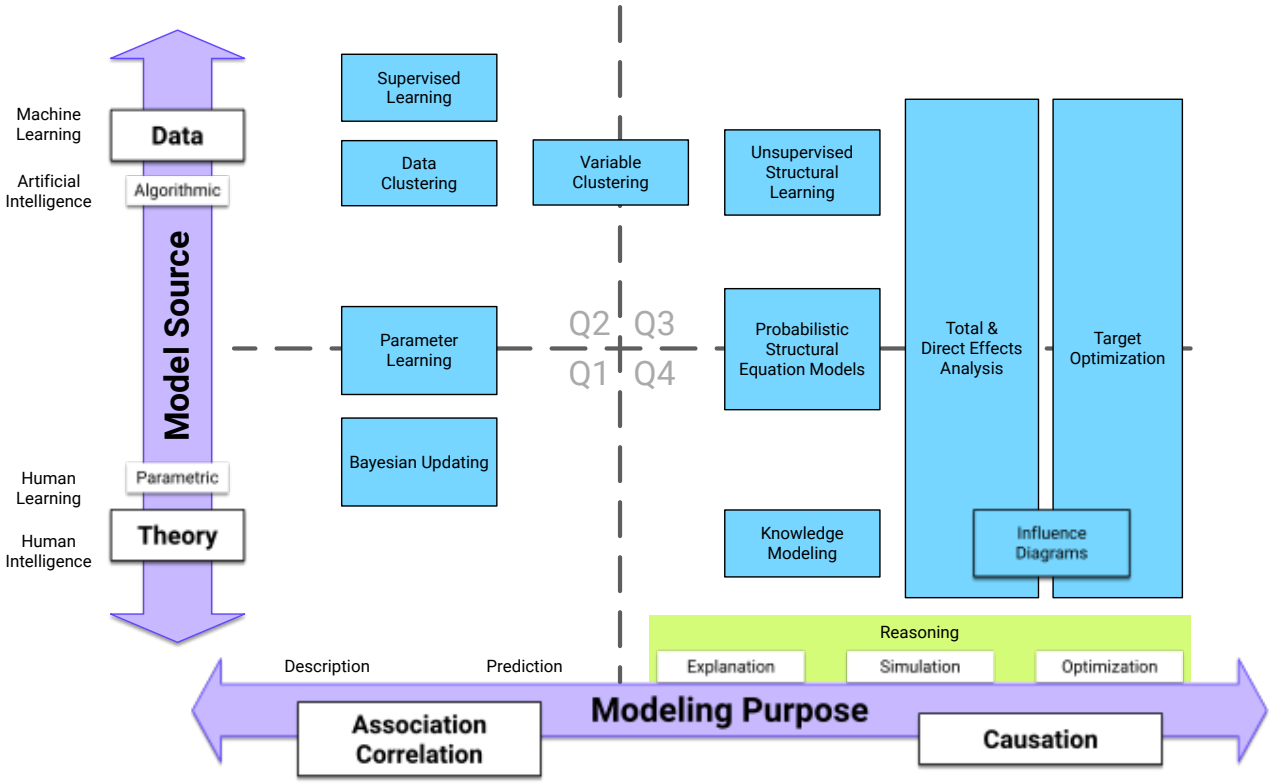
In this first chapter, however, we approach Bayesian networks from the perspective of analytical modeling. Given the current prominence of analytics, our aim is to position Bayesian networks in relation to traditional statistical methods and, in addition, to compare them with more recent developments in data mining. This context is especially relevant in light of the attention that Big Data and related technologies now receive. Their dominance in public discourse may overshadow other valuable methods of scientific inquiry, methods whose relevance becomes clear through the application of Bayesian networks.

Once we have established how Bayesian networks fit into the broader "world of analytics," Chapter 2 introduces the mathematical formalism that underpins the Bayesian network paradigm. This chapter draws heavily on a technical report by Judea Pearl, providing an authoritative foundation. While BayesiaLab has made it remarkably easy to apply Bayesian networks in research, it is essential to underscore the role of theory. A solid grasp of the underlying principles is key to using Bayesian networks correctly and effectively.

Chapter 3 concludes the first part of the book with an overview of the BayesiaLab software platform. Here, we illustrate how the theoretical strengths of Bayesian networks are operationalized in a versatile research tool, applicable across a wide range of domains, from bioinformatics to marketing science and beyond.

A Map of Analytic Modeling

Following the ideas of Breiman (2001) and Shmueli (2010), we create a "map of analytic modeling" that is defined by two axes:



- **Modeling Purpose.** The x-axis ranges from **Association/Correlation** to **Causation**. Labels on the x-axis indicate a conceptual progression, including Description, Prediction, Explanation, Simulation, and Optimization.
- **Model Source.** The y-axis represents the source of the model specification. It ranges from **Theory** (bottom) to **Data** (top). Theory is also tagged with **Parametric** as the predominant modeling approach. Additionally, it is tagged with **Human Intelligence**, hinting at the origin of Theory. On the opposite end of the y-axis, Data is associated with **Machine Learning** and **Artificial Intelligence**. It is also tagged with **Algorithmic** in contrast to parametric modeling.

Needless to say, this map displays a highly simplified view of the world of analytics. Despite this caveat, we will use this map and its coordinate system to position different modeling approaches.

Quadrant 2: Predictive Modeling

Many of today's predictive modeling techniques are algorithmic and would fall mostly into **Quadrant 2**. In **Quadrant 2**, a researcher would be primarily interested in the predictive performance of a model, i.e., Y is of interest.

$$\underset{\text{of interest}}{Y} = f(X)$$

Neural networks are a typical example of implementing machine learning techniques in this context. Such models often lack theory. However, they can be excellent "statistical devices" for producing predictions.

Quadrant 4: Explanatory Modeling

In **Quadrant 4**, the researcher is interested in identifying a model structure that best reflects the underlying "true" data-generating process, i.e., we are looking for an explanatory model. Thus, the function f is of greater interest than Y:

$$\underset{\text{of interest}}{f} (X)$$

Traditional statistical techniques that have an explanatory purpose and are used in epidemiology and the social sciences would mostly belong in **Quadrant 4**. Regressions are the best-known models in this context. Extending further into the causal direction, we would progress into the field of operations research, including simulation and optimization.

Despite the diverging objectives of predictive modeling versus explanatory modeling, i.e., predicting Y versus understanding f, the respective methods are not necessarily incompatible. In our map, this is suggested by the blue boxes that gradually fade out as they cross the boundaries and extend beyond their "home" quadrant. However, the best-performing modeling approaches do rarely serve predictive and explanatory purposes equally well. In many situations, the optimal fit-for-purpose models remain very distinct from each other. In fact, Shmueli (2010) has shown that a structurally "less true" model can yield better predictive performance than the "true" explanatory model.

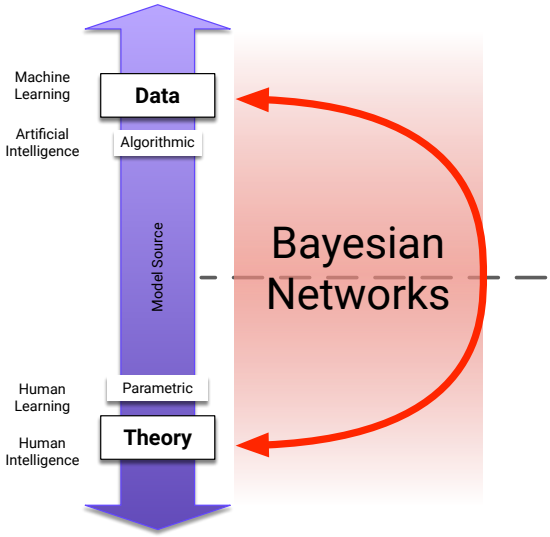
We should also point out that recent advances in machine learning and data mining have mostly occurred in **Quadrant 2** and disproportionately benefited predictive modeling. Unfortunately, most machine-learned models are remarkably difficult to interpret in terms of their structural meaning, so new theories are rarely generated this way. For instance, the well-known Netflix Prize competition produced well-performing predictive models but yielded little explanatory insight into the structural drivers of choice behavior.

Conversely, in **Quadrant 4**, developing explanatory models through machine learning remains challenging. Unlike in **Quadrant 2**, having an ever-increasing amount of data does not necessarily aid in discovering theory via machine learning.

Bayesian Networks: Theory and Data

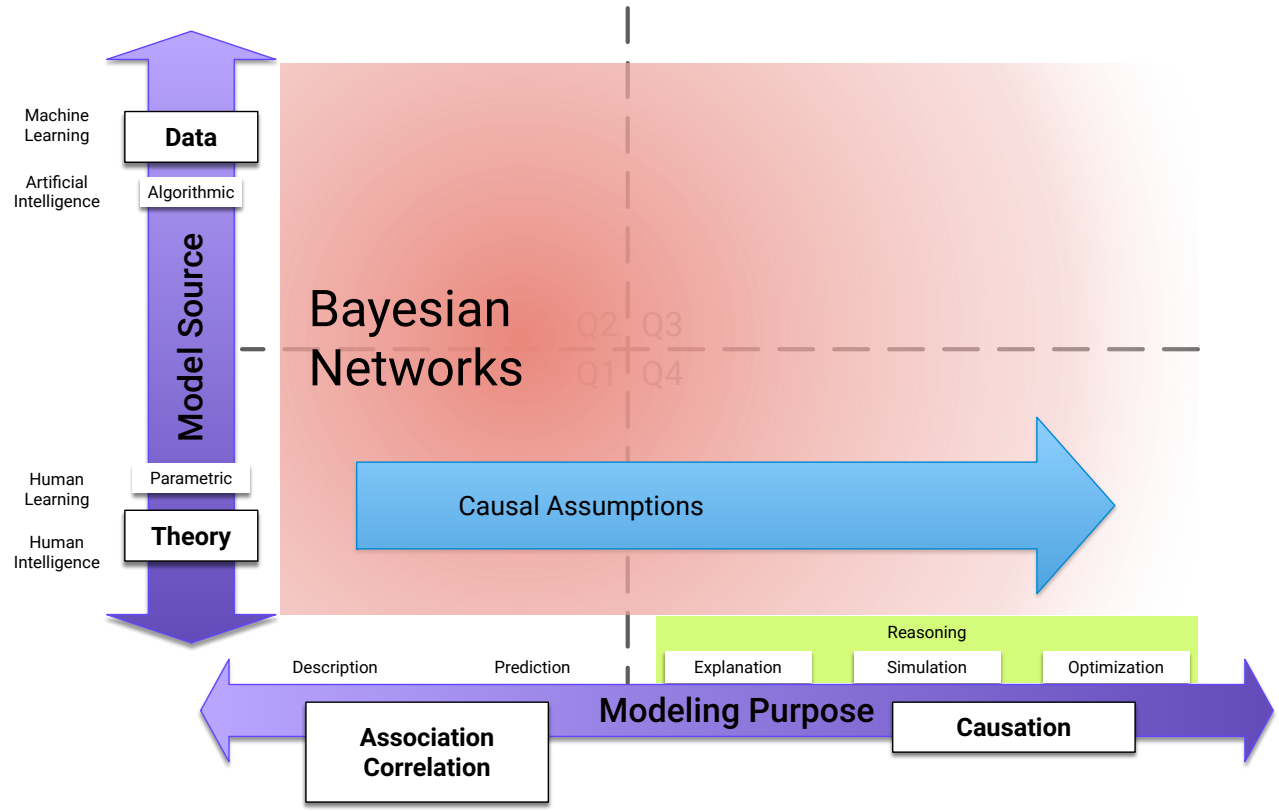
Concerning the horizontal division between **Theory** and **Data** on the **Model Source** axis, Bayesian networks have a special characteristic. Bayesian networks can be built from human knowledge, i.e., from **Theory**, or they can be machine-learned from **Data**. Thus, they can use the entire spectrum as a **Model Source**.

Due to their graphical structure, machine-learned Bayesian networks are visually interpretable, which promotes human learning and theory building. As shown by the bidirectional arc in the following diagram, Bayesian networks enable human and machine learning to work together. In other words, Bayesian networks can be developed through a combination of human and artificial intelligence.



Bayesian Networks: Association and Causation

Beyond crossing the boundaries between **Theory** and **Data**, Bayesian networks also have special qualities concerning causality. Under certain conditions and with specific theory-driven assumptions, Bayesian networks facilitate causal inference. In fact, Bayesian network models can cover the entire range from **Association/Correlation** to **Causation**, spanning the entire x-axis of the map below. In practice, this means that we can add causal assumptions to an existing non-causal network and, thus, create a causal Bayesian network. This is particularly important when we try to simulate an intervention in a domain, such as estimating the effects of a treatment. Working with a causal model is imperative in this context, and Bayesian networks help us make that transition.



As a result, Bayesian networks are a versatile modeling framework suitable for many problem domains. The mathematical formalism underpinning the Bayesian network paradigm will be presented in the next chapter.

Chapter 2: Bayesian Network Theory

This chapter is based mainly on Pearl and Russell (2000) and was adapted with permission.

History

Probabilistic Graphical Models based on **Directed Acyclic Graphs (DAG)** have a long and rich tradition, beginning with the work of geneticist Sewall Wright in the 1920s. Variants have appeared in many fields. Within statistics, such models are known as directed graphical models; within cognitive science and artificial intelligence, such models are known as Bayesian networks. The name honors the Rev. Thomas Bayes (1702-1761), whose rule for updating probabilities in the light of new evidence is the foundation of the approach.

> Bayesian networks are also referred to as Bayesian Belief Networks (BBNs) or Bayes Nets. These terms are interchangeable, but throughout this book, we use "Bayesian network" exclusively.

Bayes' Theorem

Rev. Bayes addressed both the case of discrete probability distributions of data and the more complicated case of continuous probability distributions. In the discrete case, Bayes' theorem relates the conditional and marginal probabilities of events A and B, provided that the probability of B is not equal to zero:

$P(A \mid B) = \frac{P(A) \cdot P(B|A)}{P(B)}$

In Bayes' theorem, each probability has a conventional name:

- **Prior Probability.** $P(A)$ is the prior probability (or "unconditional" or "marginal" probability) of A. It is "prior" in the sense that it does not take into account any information about B; however, event B need not occur after event A. In the nineteenth century, the unconditional probability $P(A)$ in Bayes' rule was called the "antecedent" probability; in deductive logic, the antecedent set of propositions and the inference rule imply consequences. Sir Ronald A. Fisher called the unconditional probability $P(A)$ "a priori."
- **Posterior Probability.** $P(A \mid B)$ is the conditional probability of A, given B. It is also called the posterior probability because it is derived from or depends upon the specified value of B.
- **Likelihood.** $P(B \mid A)$ is the conditional probability of B given A. It is also called the likelihood.
- **Normalizing Constant.** $P(B)$ is the prior or marginal probability of B and acts as a normalizing constant.
- **Bayes Factor.** $\frac{P(B|A)}{P(B)}$ is the Bayes factor or likelihood ratio.

Bayes' theorem in this form represents how the conditional probability of event A given B is related to the converse conditional probability of B given A.

Motivation for Developing Bayesian Networks

The initial development of Bayesian networks in the late 1970s was motivated by the necessity of modeling top-down (semantic) and bottom-up (perceptual) combinations of evidence for inference. The capability for bidirectional inference, combined with a rigorous probabilistic foundation, led to the rapid emergence of Bayesian networks. They became the method of choice for uncertain reasoning in artificial intelligence and expert systems, replacing earlier, ad hoc rule-based schemes.

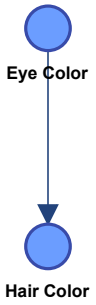
Bayesian Network Elements

Bayesian networks are models that consist of two parts: a qualitative and a quantitative part.

Once fully specified, a Bayesian network compactly represents a Joint Probability Distribution (JPD) and, thus, can be used for computing the posterior probabilities of any subset of variables given evidence about any other subset.

A Non-Causal Bayesian Network Example

Blue	Brown	Green	Hazel
94	124	43	39



Eye Color	Black	Blond	Brown	Red
Blue	11	42	34	7
Brown	37	3	68	16
Green	8	10	15	10
Hazel	5	5	21	8

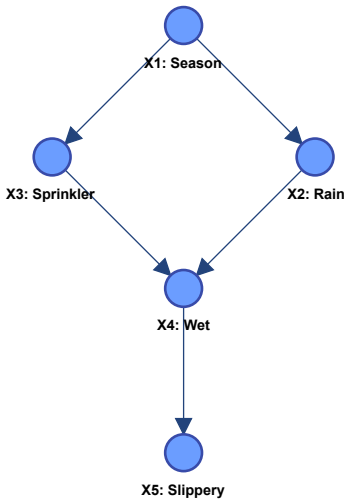
The above graph shows a simple Bayesian network consisting of only two nodes and one arc. It represents the Joint Probability Distribution (JPD) of the variables *Eye Color* and *Hair Color* in a population of students (Snee, 1974).

$P(Hair\ Color, Eye\ Color) = P(Eye\ Color) \cdot P(Hair\ Color \mid Eye\ Color)$

In this case, the conditional probabilities of *Hair Color*, given the values of its parent node, *Eye Color*, are provided in a Conditional Probability Table (CPT). It is important to point out that this Bayesian network does not contain any causal assumptions, i.e., we do not know the causal order between the variables. Thus, the interpretation of this network should be merely statistical (informational).

Download: `EyeColorHairColor.xbl`

A Causal Network Example



The above graph illustrates another simple yet typical Bayesian network. In contrast to the statistical relationships in the non-causal example, this graph describes the causal relationships among the seasons of the year X_1, whether it is raining X_2, whether the sprinkler is on X_3, whether the pavement is wet X_4, and whether the pavement is slippery X_5. Here, the absence of a direct link between X_1 and X_5, for example, captures our understanding that there is no direct influence of season on slipperiness. The influence is mediated by the wetness of the pavement (if freezing were possible, a direct link could be added).

Perhaps the most important aspect of Bayesian networks is that they are direct representations of the world, not of reasoning processes. The arrows in the diagram represent real causal connections and not the flow of information during reasoning (as in rule-based systems and neural networks). Reasoning processes can operate on Bayesian networks by propagating information in any direction. For example, if the sprinkler is on, the pavement is probably wet (prediction, simulation). If someone slips on the pavement, that will also provide evidence that it is wet (abduction, reasoning to a probable cause, or diagnosis). On the other hand, if we see that the pavement is wet, that will make it more likely that the sprinkler is on or that it is raining (abduction); but if we then observe that the sprinkler is on, that will reduce the likelihood that it is raining (explaining away). It is the latter form of reasoning, explaining away, that is especially difficult to model in rule-based systems and neural networks in a natural way because it seems to require the propagation of information in two directions.

Download: `Sprinkler-Network.xbl`

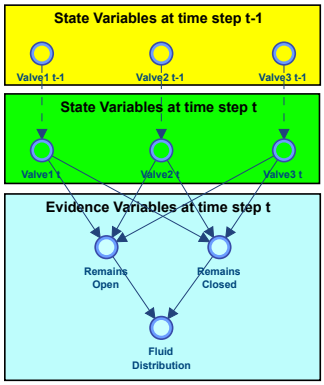
A Dynamic Bayesian Network Example

Entities that live in a changing environment must keep track of variables whose values change over time.

Dynamic Bayesian networks (DBN) capture this process by representing multiple copies of the state variables, one for each time step.

- **State Variables.** A set of variables X_{t-1} and X_t denotes the world state at times $t-1$ and t, respectively.
- **Evidence Variables.** A set of evidence variables E_t denotes the observations available at time t.
- **Sensor Model.** The sensor model $P(E_t \mid X_t)$ is encoded in the conditional probability distributions for the observable variables, given the state variables.
- **Transition Model.** The transition model $P(X_t \mid X_{t-1})$ relates the state at time $t-1$ to the state at time t. Keeping track of the world means computing the current probability distribution over world states given all past observations, i.e., $P(X_t \mid E_1, \ldots, E_t)$.

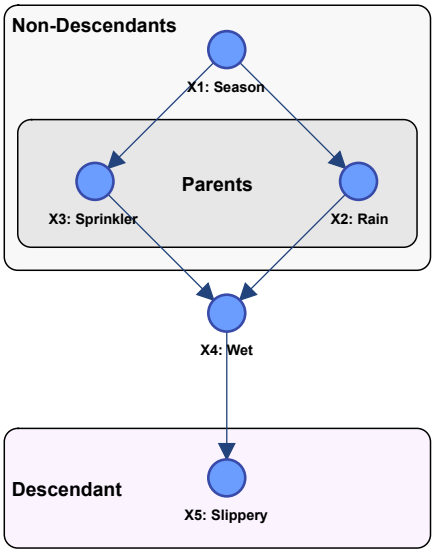
Dynamic Bayesian networks are a generalization of **Hidden Markov Models (HMM)** and **Kalman Filters (KF)**. Every **HMM** and **KF** can be represented with a **DBN**. Furthermore, the **DBN** representation of an **HMM** is much more compact and, thus, much easier to understand. The nodes in the **HMM** represent the states of the system, whereas the nodes in the **DBN** represent the dimensions of the system. For example, the **HMM** representation of the valve system shown in the following graph is made of 26 nodes and 36 arcs compared to 9 nodes and 11 arcs in the **DBN** (Weber and Jouffe, 2003).



Representation of the Joint Probability Distribution

Any complete probabilistic model of a domain must — either explicitly or implicitly — represent the Joint Probability Distribution (JPD), i.e., the probability of every possible event as defined by the combination of the values of all the variables. There are exponentially many such events, yet Bayesian networks achieve compactness by factoring the Joint Probability Distribution (JPD) into

local, conditional distributions for each variable given its parents. If x_i denotes some value of the variable X_i and pa_i denotes some set of values for the parents of X_i, then $P(x_i \mid pa_i)$ denotes this conditional probability distribution. For example, in the graph below, $P(x_4 \mid x_2, x_3)$ is the probability of Wetness given the values of Sprinkler and Rain.



Product Rule (Chain Rule)

The global semantics of Bayesian networks specifies that the full Joint Probability Distribution (JPD) is given by the product rule (or chain rule):

$P(x_1, \ldots, x_n) = \prod_i P(x_i \mid pa_i)$

In our example network, we have the following:

$P(x_1, x_2, x_3, x_4, x_5) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1)P(x_4 \mid x_2, x_3)P(x_5 \mid x_4)$

It becomes clear that the number of parameters grows linearly with the size of the network, i.e., the number of variables. In contrast, the size of the Joint Probability Distribution (JPD) itself grows exponentially. Given a discrete representation of the JPD with a Conditional Probability Table (CPT), the size of a local CPT grows exponentially with the number of parents. Savings can be achieved using compact JPD representations — such as noisy-OR models, trees, or neural networks.

The Joint Probability Distribution (JPD) representation with Bayesian networks also translates into local semantics, which asserts that each variable is independent of descendants in the network given its parents. For example, the parents of X_4 in the following graph are X_2 and X_3, and they render X_4 independent of the remaining non-descendant, X_1:

$P(x_4 \mid x_1, x_2, x_3) = P(x_4 \mid x_2, x_3)$

The collection of independence assertions formed in this way suffices to derive the global assertion of the product rule and vice versa. The local semantics is most useful for constructing Bayesian

networks because selecting as parents all the direct causes (or direct relationships) of a given variable invariably satisfies the local conditional independence conditions. The global semantics leads directly to a variety of algorithms for reasoning.

Evidential Reasoning

From the product rule, one can express the probability of any desired proposition in terms of the conditional probabilities specified in the network. For example, the probability that the Sprinkler is on given that the Pavement is slippery is:

$$P(X_3 = \text{on} \mid X_5 = \text{true})$$

$$= \frac{P(X_3 = \text{on}, X_5 = \text{true})}{P(X_5 = \text{true})}$$

$$= \frac{\sum_{x_1,x_2,x_4} P(x_1, x_2, X_3 = \text{on}, x_4, X_5 = \text{true})}{\sum_{x_1,x_2,x_3,x_4} P(x_1, x_2, x_3, x_4, X_5 = \text{true})}$$

$$= \frac{\sum_{x_1,x_2,x_4} P(x_1)P(x_2 \mid x_1)P(X_3 = \text{on} \mid x_1)P(x_4 \mid x_2, X_3 = \text{on})P(X_5 = \text{true} \mid x_4)}{\sum_{x_1,x_2,x_3,x_4} P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1)P(x_4 \mid x_2, x_3)P(X_5 = \text{true} \mid x_4)}$$

These expressions can often be simplified in ways that reflect the structure of the network itself. The first algorithms proposed for probabilistic calculations in Bayesian networks used a local distributed message-passing architecture, typical of many cognitive activities. Initially, this approach was limited to tree-structured networks but was later extended to general networks in Lauritzen and Spiegelhalter's (1988) method of junction tree propagation. A number of other exact methods have been developed and can be found in recent textbooks.

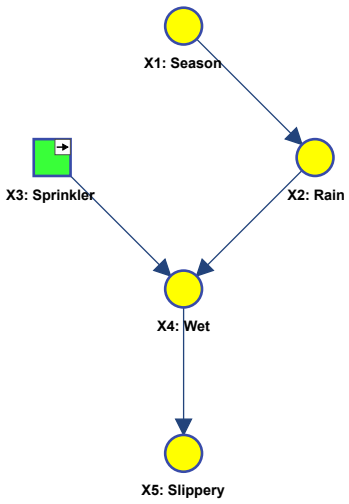
It is easy to show that reasoning in Bayesian networks subsumes the satisfiability problem in propositional logic and, therefore, exact inference is NP-hard. Monte Carlo simulation methods can be used for approximate inference (Pearl, 1988), giving gradually improving estimates as sampling proceeds. Unlike junction-tree methods, these methods use local message propagation on the original network structure. Alternatively, variational methods provide bounds on the true probability.

Causal Reasoning

Most probabilistic models, including general Bayesian networks, describe a Joint Probability Distribution (JPD) over possible observed events but say nothing about what will happen if a certain intervention occurs. For example, what if I turn the Sprinkler on instead of just observing that it is turned on? What effect does that have on the Season or the connection between Wet and Slippery? A causal network, intuitively speaking, is a Bayesian network with the added property that the parents of each node are its direct causes. In such a network, the result of an intervention is obvious: the Sprinkler node is set to $X_3 = \text{on}$, and the causal link between Season X_1 and the Sprinkler X_3 is removed. All other causal links and conditional probabilities remain intact, so the new model is:

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1)P(x_2 \mid x_1)P(x_4 \mid x_2, X_3 = \text{on})P(x_5 \mid x_4)$$

Notice that this differs from observing that $X_3 = \text{on}$, which would result in a new model that included the term $P(X_3 = \text{on} \mid X_1)$. This mirrors the difference between seeing and doing: after observing that the Sprinkler is on, we wish to infer that the Season is dry, that it probably did not rain, and so on. An arbitrary decision to turn on the Sprinkler should not result in any such beliefs.



Causal networks are more properly defined, then, as Bayesian networks in which the correct probability model — after intervening to fix any node's value — is given simply by deleting links from the node's parents. For example, Fire → Smoke is a causal network, whereas Smoke → Fire is not, even though both networks are equally capable of representing any Joint Probability Distribution (JPD) of the two variables.

Causal networks model the environment as a collection of stable component mechanisms. These mechanisms may be reconfigured locally by interventions, with corresponding local changes in the model. This, in turn, allows causal networks to be used very naturally for prediction by an agent that is considering various courses of action.

Learning Bayesian Network Parameters

Given a qualitative Bayesian network structure, the conditional probability tables, $P(x_i \mid pa_i)$, are typically estimated using Maximum-Likelihood Estimation from the observed frequencies in the dataset associated with the network.

In pure Bayesian approaches, Bayesian networks are designed from expert knowledge and include hyperparameter nodes. Data, usually scarce, is used as pieces of evidence for incrementally updating the distributions of the hyperparameters (Bayesian Updating).

Learning Bayesian Network Structure

It is also possible to machine learn the structure of a Bayesian network, and two families of methods are available for that purpose.

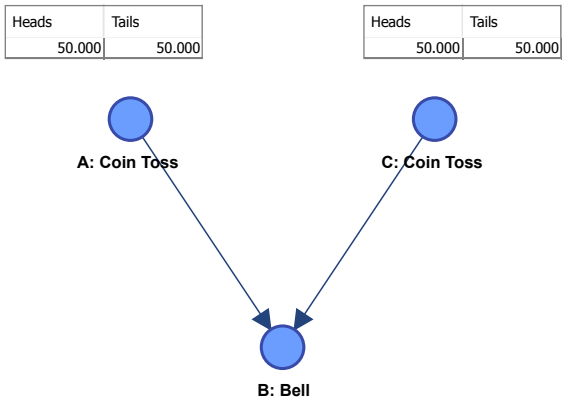
- **Constraint-Based Algorithms.** This family of methods is based on the probabilistic semantics of Bayesian networks. Links are added or deleted according to the results of statistical tests, which identify marginal and conditional independencies.
- **Score-Based Algorithms.** This family of methods is based on a metric, such as the Minimum Description Length Score, that measures the quality of candidate networks with respect to the observed data. This metric trades off network complexity against the degree of fit to the data,

which is typically expressed as the likelihood of the data given the network. This is the only method we use in this book.

As a substrate for learning, Bayesian networks have the advantage that it is relatively easy to encode prior knowledge in network form by fixing portions of the structure, forbidding relations, or using prior distributions over the network parameters. Such prior knowledge can allow a system to learn accurate models from much less data than is required for clean-sheet approaches.

Causal Discovery

One of the most exciting prospects in recent years has been the possibility of using Bayesian networks to discover causal structures in raw statistical data — a task previously considered impossible without controlled experiments. Consider, for example, the following intransitive pattern of dependencies among three events: A and B are dependent, B and C are dependent, yet A and C are independent. If you asked a person to supply an example of three such events, the example would invariably portray A and C as two independent causes and B as their common effect, namely $A \rightarrow B \leftarrow C$. For instance, A and C could be the outcomes of two fair coins, and B represents a bell that rings whenever either coin comes up heads.



A: Coin Toss	C: Coin Toss	Rings	Does Not Ring
Heads	Heads	100.000	0.000
	Tails	100.000	0.000
Tails	Heads	100.000	0.000
	Tails	0.000	100.000

Fitting this dependence pattern with a scenario where B is the cause and A and C are the effects is mathematically feasible but unnatural because it must entail fine-tuning the probabilities involved. The desired dependence pattern will be destroyed as soon as the probabilities change slightly.

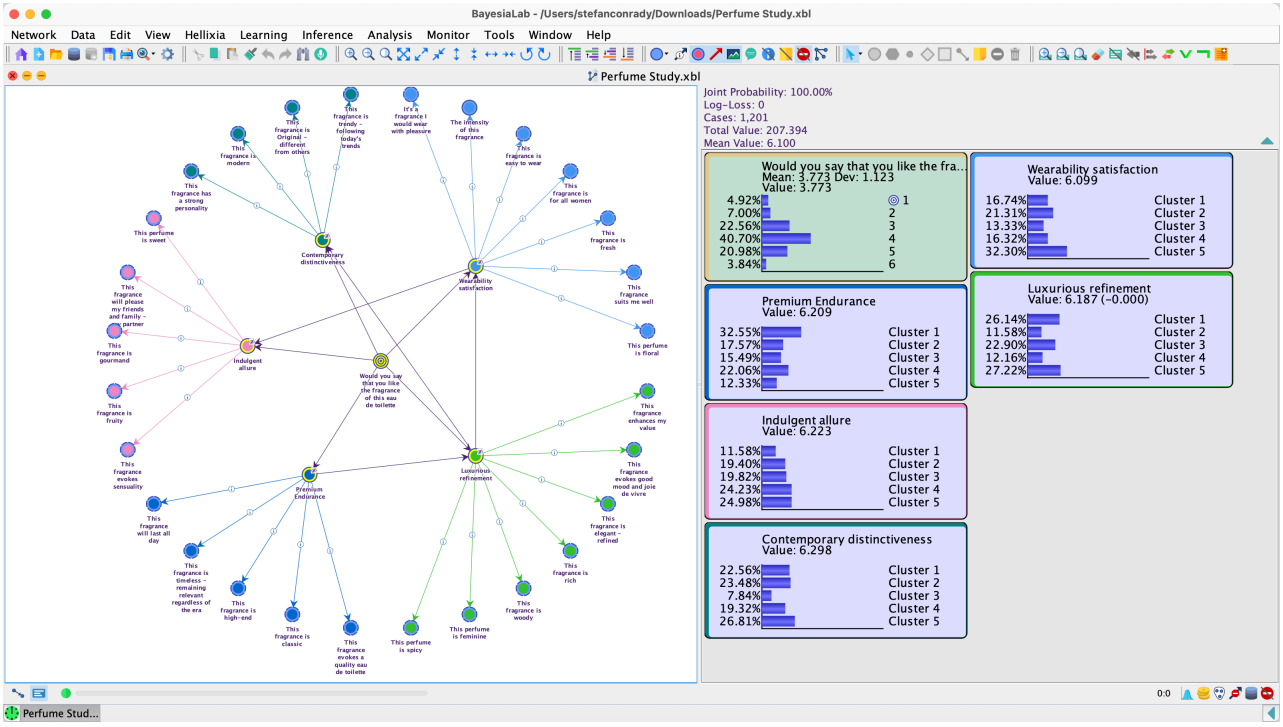
Such thought experiments tell us that certain patterns of dependency, which are totally void of temporal information, are conceptually characteristic of certain causal directionalities and not others. When put together systematically, such patterns can be used to infer causal structures from raw data and to guarantee that any alternative structure compatible with the data must be less stable than the one(s) inferred; namely, slight fluctuations in parameters will render that structure incompatible with the data.

Despite recent advances, causal discovery is an active research area with countless unresolved questions. Thus, no generally accepted causal discovery algorithms are currently available for applied researchers. As a result, all causal networks presented in this book are constructed from expert knowledge or machine learning and then validated as causal by experts. The assumptions necessary for a causal interpretation of a Bayesian network will be discussed in Chapter 10.

Chapter 3: BayesiaLab

While the conceptual advantages of Bayesian networks had been known in the world of academia for some time, leveraging these properties for practical research applications was very difficult for non-computer scientists before BayesiaLab's first release in 2002.

BayesiaLab is a powerful desktop application (Windows/macOS/Unix/Linux) with a sophisticated graphical user interface, which provides scientists with a comprehensive "laboratory" environment for machine learning, knowledge modeling, diagnosis, analysis, simulation, and optimization. With BayesiaLab, Bayesian networks have become practical for gaining deep insights into problem domains. BayesiaLab leverages the inherently graphical structure of Bayesian networks to explore and explain complex problems. The screenshot below shows a typical research project.

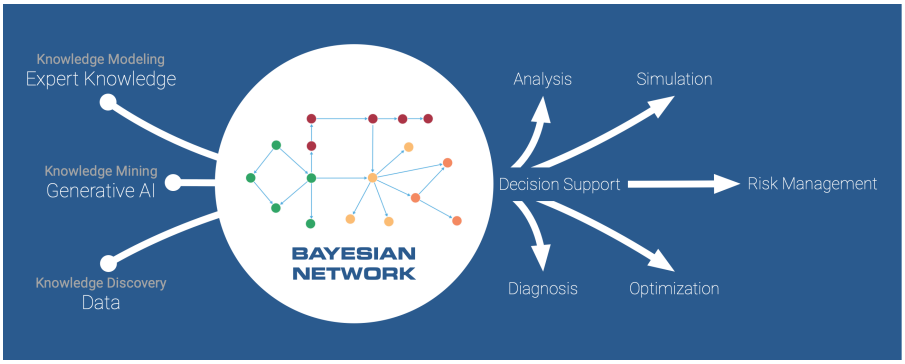


BayesiaLab is the result of nearly twenty years of research and software development by Dr. Lionel Jouffe and Dr. Paul Munteanu. In 2001, their research efforts led to the formation of Bayesia S.A.S., headquartered in Laval in northwestern France. Today, the company is the world's leading supplier of Bayesian network software, serving hundreds of major corporations and research organizations.

BayesiaLab's Methods, Features, and Functions

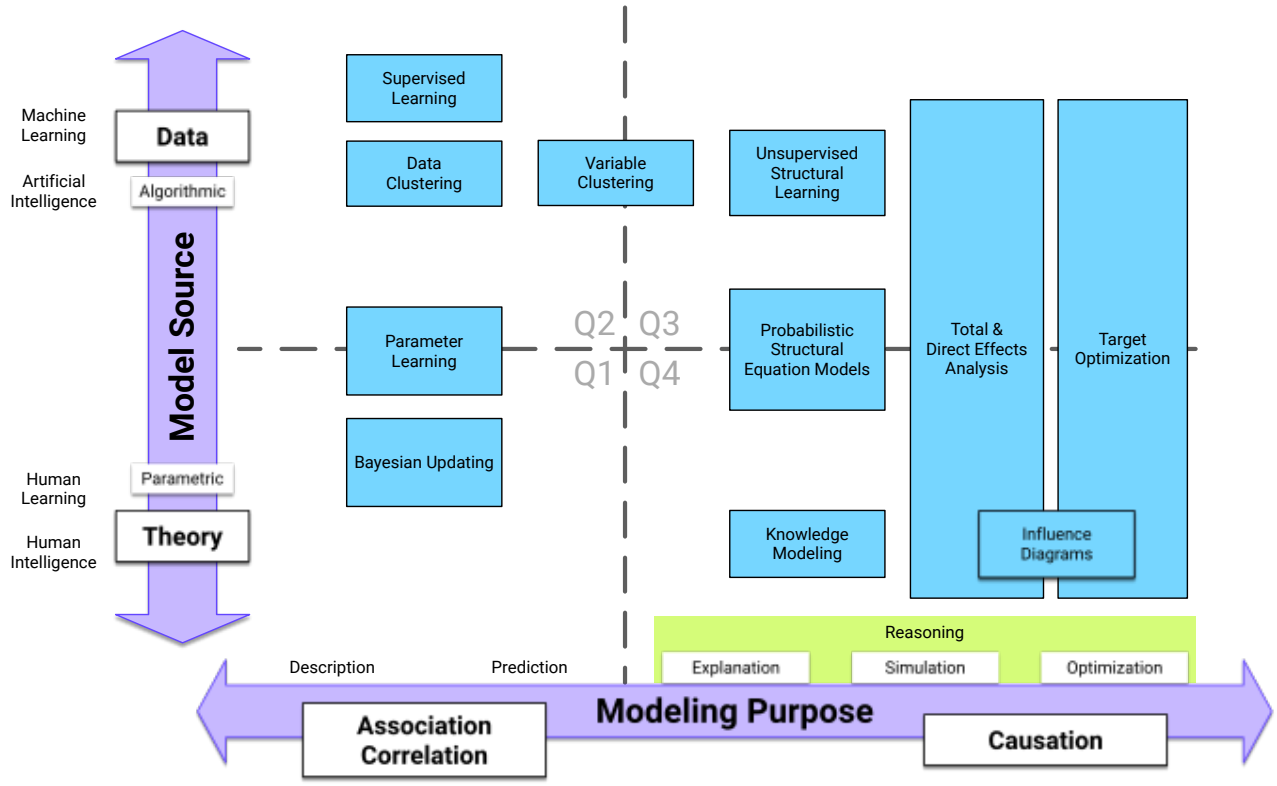
As conceptualized in the diagram below, BayesiaLab is designed around a prototypical workflow with a Bayesian network model at the center. BayesiaLab supports the research process from model

generation to analysis, simulation, and optimization. The entire process is fully contained in a uniform "lab" environment, which allows scientists to move back and forth between different elements of the research task.



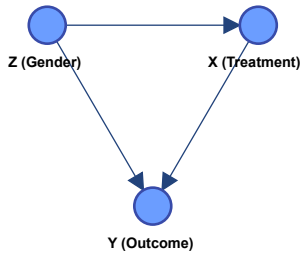
In Chapter 1, we presented our principal motivation for using Bayesian networks, namely their universal suitability across the entire "map" of analytic modeling: Bayesian networks can be modeled from pure theory, and they can be learned from data alone; Bayesian networks can serve as predictive models, and they can represent causal relationships.

The following "map of analytic modeling and reasoning" shows how our claim of "universal modeling capability" translates into specific functions provided by BayesiaLab, which are placed as blue boxes on this map.



Knowledge Modeling

Subject matter experts often express their causal understanding of a domain through diagrams with arrows indicating causal directions. This visual representation of causes and effects has a direct analog in the network graph in BayesiaLab. Nodes (representing variables) can be added and positioned on BayesiaLab's Graph Panel with a mouse click, and arcs (representing relationships) can be "drawn" between nodes. As in the following network graph, the causal direction can be encoded by orienting the arcs from cause to effect.

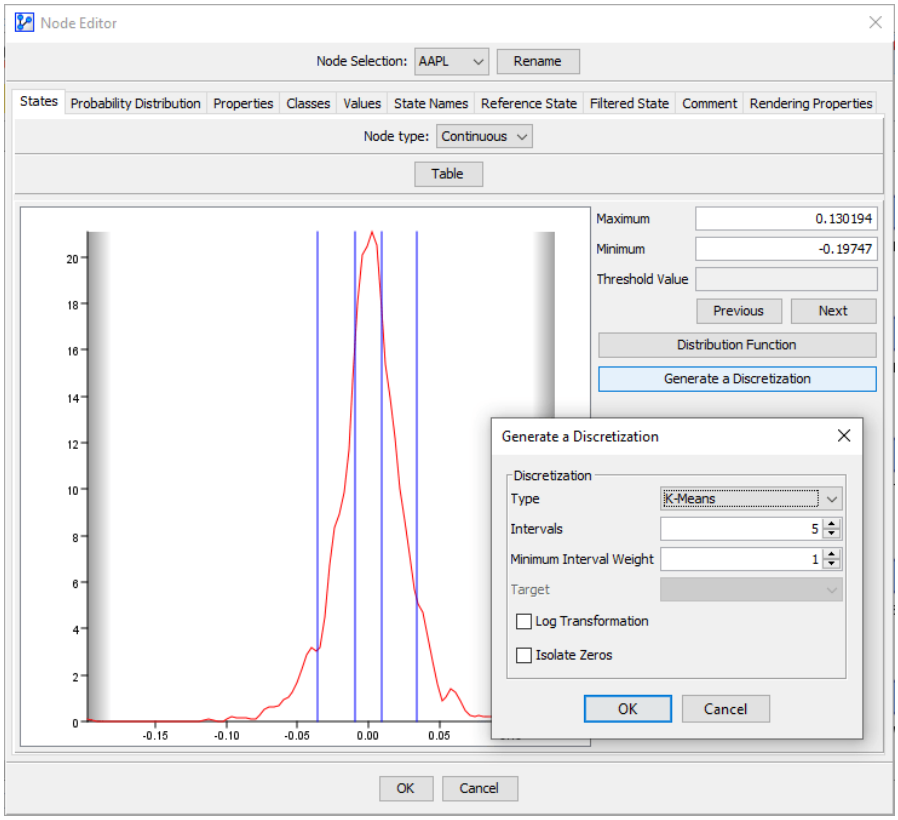


The quantitative nature of relationships between variables, plus many other attributes, can be managed in BayesiaLab's Node Editor. In this way, BayesiaLab facilitates the straightforward encoding of one's understanding of a domain. Simultaneously, BayesiaLab enforces internal consistency so that impossible conditions cannot be encoded accidentally. Chapter 4 will present a practical example of causal knowledge modeling, followed by probabilistic reasoning.

In addition to having individuals directly encode their explicit knowledge in BayesiaLab, the Bayesia Expert Knowledge Elicitation Environment (BEKEE) is available for acquiring the probabilities of a network from a group of experts. BEKEE offers a web-based interface for systematically eliciting explicit and tacit knowledge from multiple stakeholders.

Discrete, Nonlinear, and Nonparametric Modeling

BayesiaLab contains all "parameters" describing probabilistic relationships between variables in conditional probability tables (CPT), which means that no functional forms are utilized. Given this nonparametric, discrete approach, BayesiaLab can conveniently handle nonlinear relationships between variables. However, this CPT-based representation requires a preparation step for dealing with continuous variables, namely discretization. This consists of manually or automatically defining a discrete representation of all continuous values. BayesiaLab offers several tools for discretization, which are accessible in the Data Import Wizard, in the Node Editor (shown below), and in a standalone Discretization function. Univariate, bivariate, and multivariate discretization algorithms are available in this context.



Missing Values Processing

BayesiaLab offers a range of sophisticated methods for missing values processing. During network learning, BayesiaLab performs missing values processing automatically "behind the scenes." More specifically, the Structural EM algorithm or the Dynamic Imputation algorithms are applied after each network modification during learning, i.e., after every arc addition, suppression, and inversion. Bayesian networks provide a few fundamental advantages for dealing with missing values. In Chapter 9, we will focus exclusively on this topic.

Parameter Estimation

Parameter Estimation with BayesiaLab is at the intersection of theory-driven and data-driven modeling. For a network that was generated either from expert knowledge or through machine learning, BayesiaLab can use the observations contained in an associated dataset to populate the CPT via Maximum Likelihood Estimation.

Bayesian Updating

In general, Bayesian networks are nonparametric models. However, a Bayesian network can also serve as a parametric model if an expert uses equations for defining local CPDs and, additionally, specifies hyperparameters, i.e., nodes that explicitly represent parameters that are used in the equations.

As opposed to BayesiaLab's usual parameter estimation via Maximum Likelihood, the associated dataset provides pieces of evidence for incrementally updating the distributions of the hyperparameters via probabilistic inference.

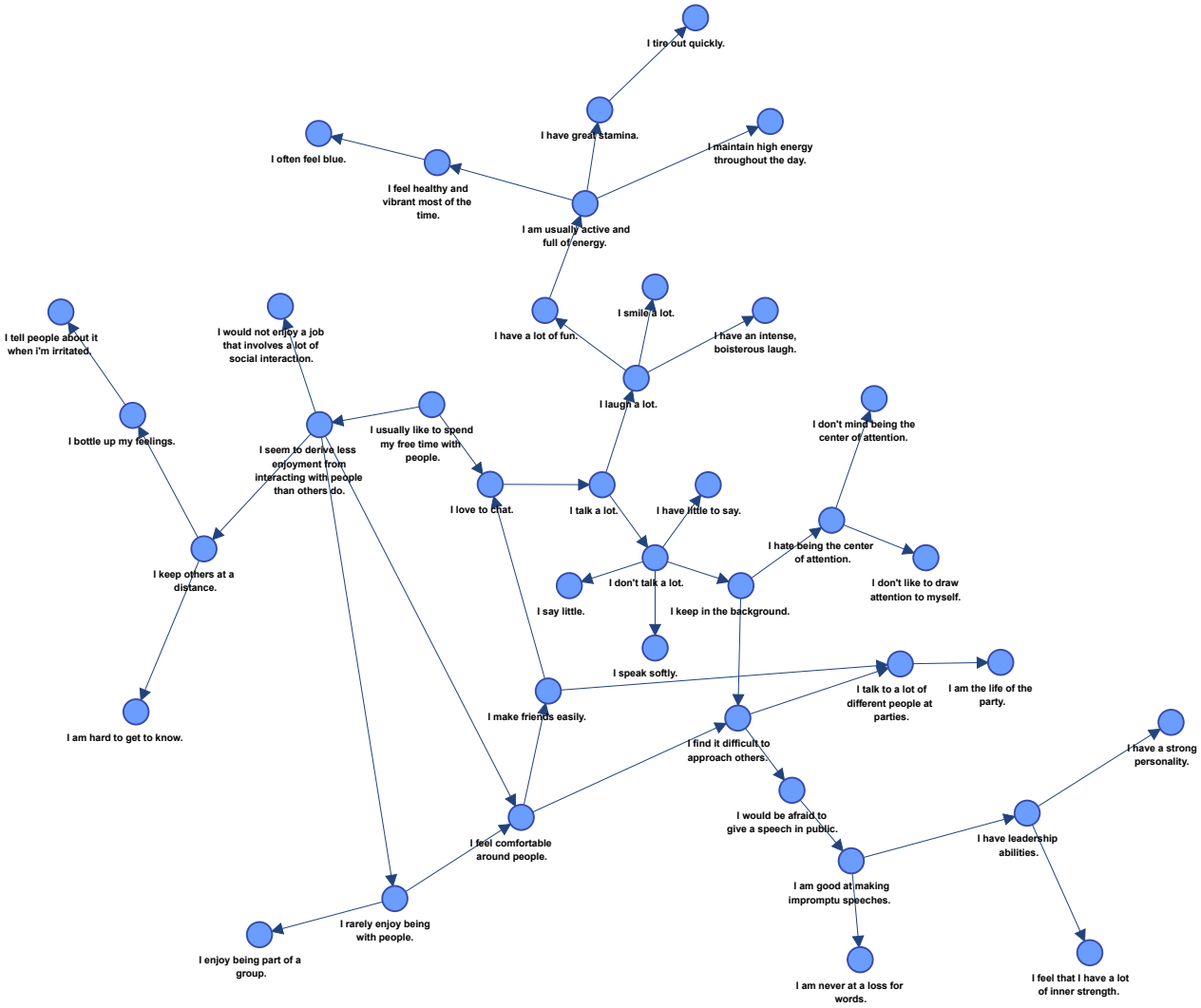
Machine Learning

Despite our repeated emphasis on the relevance of human expert knowledge, especially for identifying causal relations, much of this book is dedicated to acquiring knowledge from data through machine learning. BayesiaLab features a comprehensive array of highly optimized learning algorithms that can quickly uncover structures in datasets. The optimization criteria in BayesiaLab's learning algorithms are based on information theory (e.g., the Minimum Description Length). With that, no assumptions regarding the variable distributions are made. These algorithms can be used for all kinds and all sizes of problem domains, sometimes including thousands of variables with millions of potentially relevant relationships.

Unsupervised Structural Learning (Quadrant 2/3)

In statistics, "unsupervised learning" is typically understood to be a classification or clustering task. To make a clear distinction, we emphasize "structural" in "Unsupervised Structural Learning," which covers a number of important algorithms in BayesiaLab.

Unsupervised Structural Learning means that BayesiaLab can discover probabilistic relationships between many variables without having to specify input or output nodes. One might say that this is a quintessential form of knowledge discovery, as no assumptions are required to perform these algorithms on unknown datasets.

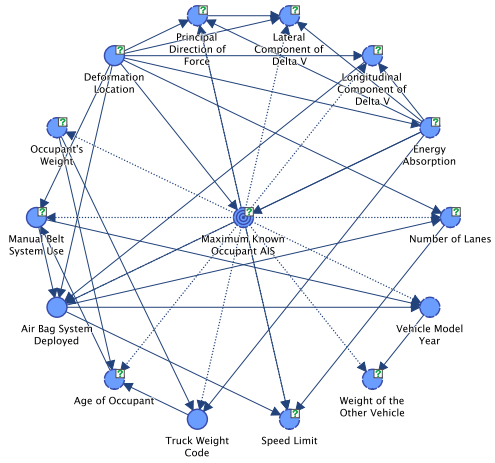


Supervised Learning (Quadrant 2)

Supervised Learning in BayesiaLab has the same objective as many traditional modeling methods, i.e., to develop a model for predicting a target variable. Note that numerous statistical packages also offer "Bayesian Networks" as a predictive modeling technique. However, in most cases, these packages are restricted in their capabilities to one type of network, i.e., the Naive Bayes network. BayesiaLab offers a much greater number of Supervised Learning algorithms to search for the Bayesian network that best predicts the target variable while also considering the complexity of the resulting network (see Chapter 6).

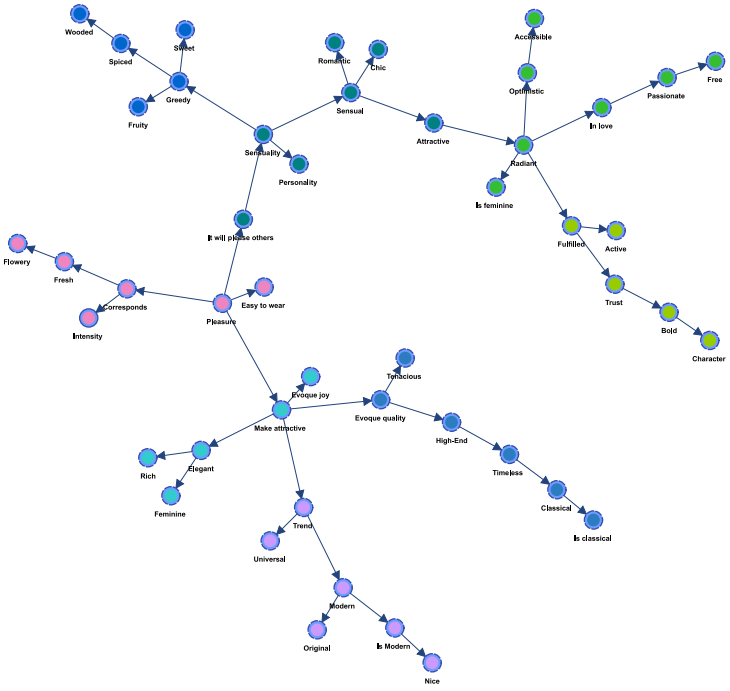
We should highlight the Markov Blanket algorithm for its speed, which is particularly helpful when dealing with many variables. The Markov Blanket algorithm can be an efficient variable selection

algorithm in this context. An example of Supervised Learning using this algorithm and the closely related Augmented Markov Blanket algorithm will be presented in Chapter 6.



Clustering (Quadrant 2/3)

Clustering in BayesiaLab covers both Data Clustering and Variable Clustering. The former applies to the grouping of records (or observations) in a dataset; the latter performs a grouping of variables according to the strength of their mutual relationships.



The third variation of this concept is of particular importance in BayesiaLab: Multiple Clustering can be characterized as a nonlinear, nonparametric, and nonorthogonal factor analysis. Multiple

Clustering often serves as the basis for developing Probabilistic Structural Equation Models (Quadrant 3/4) with BayesiaLab.

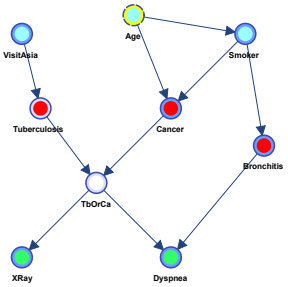
Inference: Diagnosis, Prediction, and Simulation

The inherent ability of Bayesian networks to explicitly model uncertainty makes them suitable for a broad range of real-world applications. In the Bayesian network framework, diagnosis, prediction, and simulation are identical computations. They all consist of observational inference conditional upon evidence.

This distinction, however, only exists from the perspective of the researcher, who would presumably see the symptom of a disease as the effect and the disease itself as the cause. Hence, carrying out inference based on observed symptoms is interpreted as a "diagnosis."

Observational Inference (Quadrant 1/2)

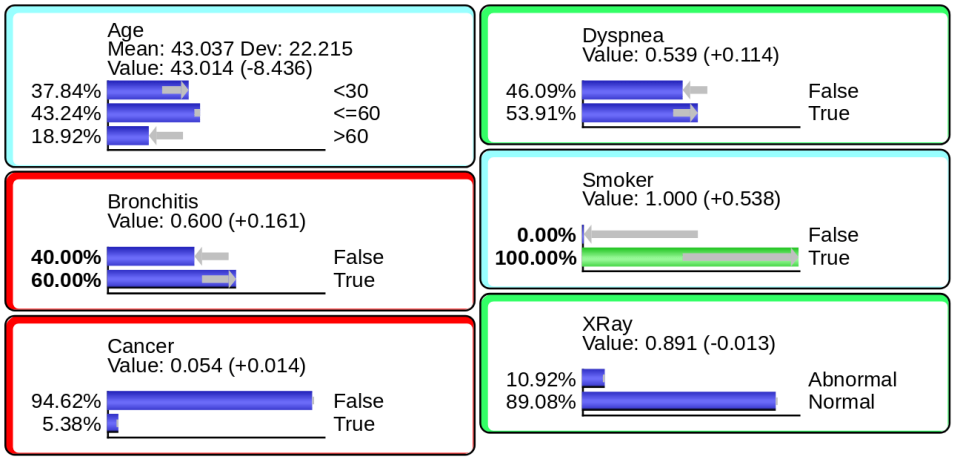
One of the central benefits of Bayesian networks is that they compute inference "omnidirectionally." Given an observation with any type of evidence on any of the networks' nodes (or a subset of nodes), BayesiaLab can compute the posterior probabilities of all other nodes in the network, regardless of arc direction. Both exact and approximate observational inference algorithms are implemented in BayesiaLab. We briefly illustrate evidence-setting and inference with the expert system network shown below:



Types of Evidence

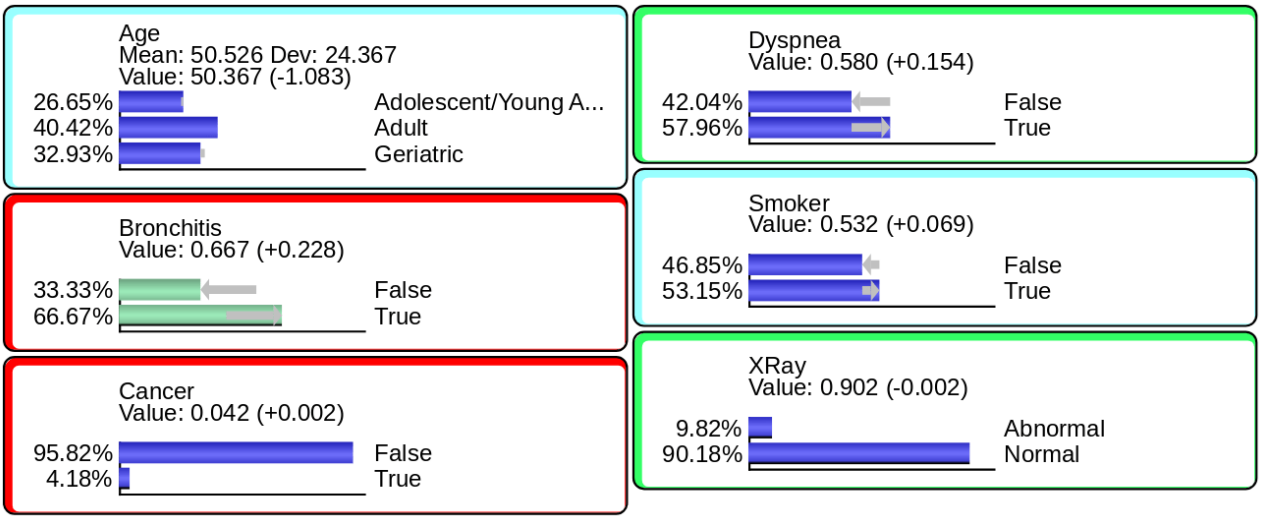
Hard Evidence

Hard Evidence has no uncertainty regarding the state of the variable (node), e.g., $P(Smoker = True) = 100\%$.



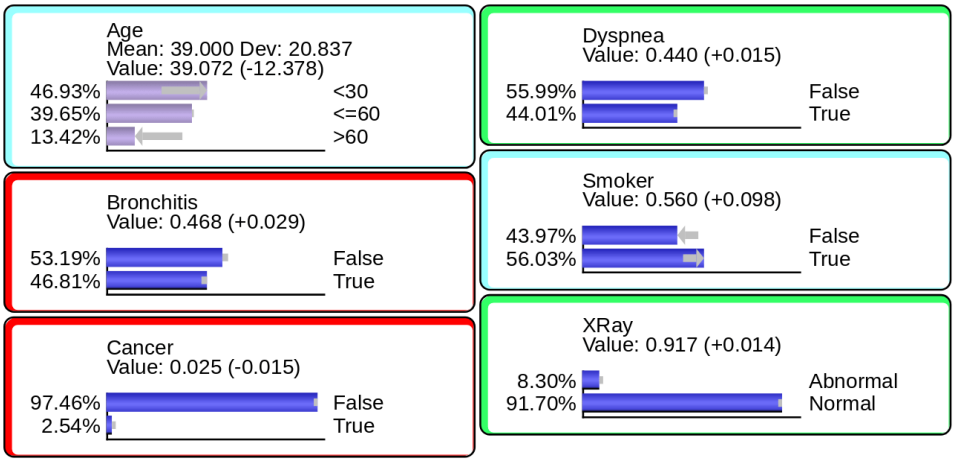
Probabilistic Evidence

Probabilistic Evidence (or Soft Evidence) is defined by marginal probability distributions: $P(Bronchitis = True) = 66.67\%$.



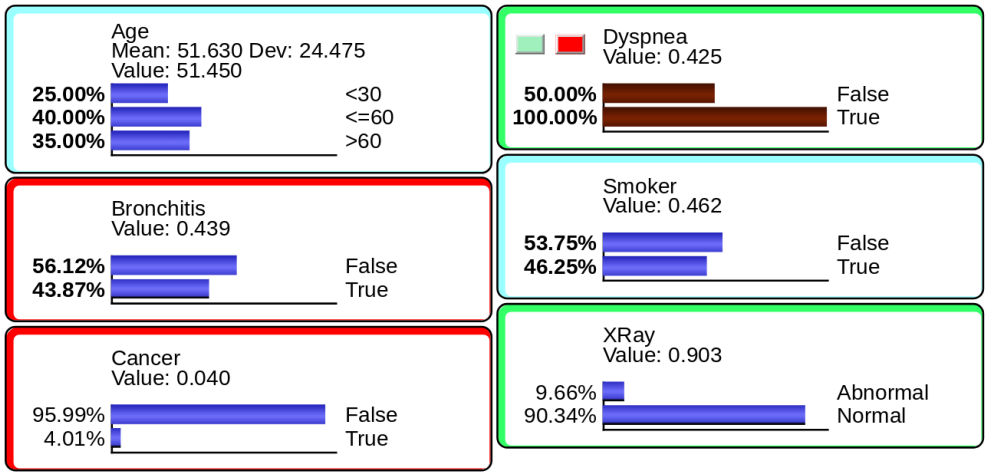
Numerical Evidence

Numerical Evidence applies to numerical variables or to categorical/symbolic variables that have associated numerical values. BayesiaLab computes a marginal probability distribution to generate the specified expected value: $E(Age) = 39$.



Likelihood Evidence

Likelihood Evidence (or Virtual Evidence) is defined by the likelihood of each state, ranging from 0% (impossible) to 100% (which means that no evidence reduces the probability of the state). The sum of the likelihoods must be greater than 0 to be valid as evidence. Also, note that the upper boundary for the sum of the likelihoods equals the number of states. Setting the same likelihood to all states corresponds to no evidence.



Causal Inference (Quadrant 3/4)

Beyond observational inference, BayesiaLab can also perform causal inference to compute the impact of intervening on a subset of variables instead of merely observing these variables. Pearl's Do-Operator and Jouffe's Likelihood Matching are available for this purpose. We will provide a detailed discussion of causal inference in Chapter 10.

Effects Analysis (Quadrants 3/4)

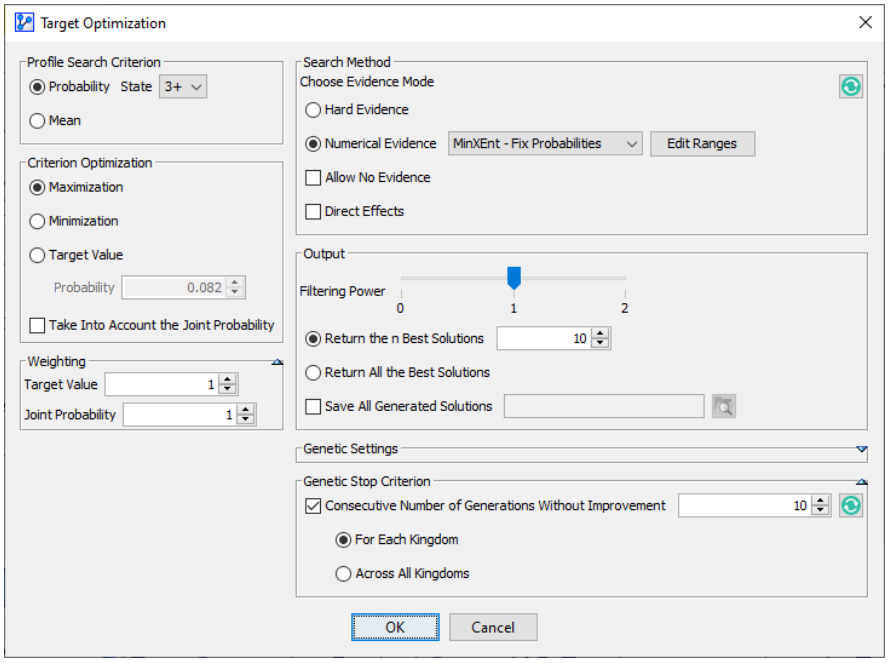
Many research activities focus on estimating the size of an effect, e.g., to establish the treatment effect of a new drug or to determine the sales boost from a new advertising campaign. Other studies attempt to decompose observed effects into their causes, i.e., they perform attribution.

BayesiaLab performs simulations to compute effects, as parameters as such do not exist in this nonparametric framework. As all the domain dynamics are encoded in discrete CPTs, effect sizes only manifest themselves when different conditions are simulated. Total Effects Analysis, Target Mean Analysis, and several other functions offer ways to study effects, including nonlinear and variable interactions.

Optimization (Quadrant 4)

BayesiaLab's ability to perform inference over all possible states of all nodes in a network also provides the basis for searching for node values that optimize a target criterion. BayesiaLab's Target Dynamic Profile and Target Optimization are a set of tools for this purpose.

Using these functions in combination with Direct Effects is particularly interesting when searching for the optimum combination of variables with a nonlinear relationship with the target, plus interdependencies among them. A typical example would be searching for the optimum mix of marketing expenditures to maximize sales. BayesiaLab's Target Optimization will search, within the specified constraints, for those scenarios that optimize the target criterion. An example of Target Dynamic Profile will be presented in Chapter 8.



Model Utilization

BayesiaLab provides a range of functions for systematically utilizing the knowledge contained in a Bayesian network. They make a network accessible as an expert system that can be queried interactively by an end-user or through an automated process.

The Adaptive Questionnaire guides users through the optimum sequence for seeking evidence. BayesiaLab determines dynamically, given the evidence already gathered, the next best piece of evidence to obtain in order to maximize the information gain with respect to the target variable while minimizing the cost of acquiring such evidence. In a medical context, for instance, this would allow for the optimal "escalation" of diagnostic procedures from "low-cost/small-gain" evidence (e.g., measuring the patient's blood pressure) to "high-cost/large-gain" evidence (e.g., performing an MRI scan). The Adaptive Questionnaire will be presented in the context of an example of tumor classification in Chapter 6.

The WebSimulator is a platform for publishing interactive models and Adaptive Questionnaires via the web, which means that any Bayesian network model built with BayesiaLab can be shared privately with clients or publicly with a broader audience. Once a model is published via the WebSimulator, end users can try out scenarios and examine the dynamics of that model.

Watch the video online

Batch Inference is available to automatically perform inference on a large number of records in a dataset. For example, Batch Inference can be used to produce a predictive score for all customers in a database. With the same objective, BayesiaLab's optional Export function can translate predictive network models into static code that can run in external programs. Modules are available that can generate code for R, SAS, PHP, VBA, and JavaScript.

Developers can also access many of BayesiaLab's functions outside the graphical user interface using the Bayesia Engine API. The Bayesia Modeling Engine allows the construction and editing of networks. The Bayesia Inference Engine can access network models programmatically to perform automated inference, e.g., as part of a real-time application with streaming data. The Bayesia Engine API was recently augmented with a model learning capability, which allows programmatic access to BayesiaLab's learning algorithms. This functionality is ideally suited for machine-learning models from streaming data.

The Bayesia Engine API is implemented as pure Java class libraries (jar files), which can be integrated into any software project.

Knowledge Communication

While generating a Bayesian network, either by expert knowledge modeling or through machine learning, is all about a computer acquiring knowledge, a Bayesian network can also be a remarkably powerful tool for humans to extract or "harvest" knowledge. Given that a Bayesian network can serve as a high-dimensional representation of a real-world domain, BayesiaLab allows us to interactively, even playfully, engage with this domain to learn about it. Through visualization, simulation, and analysis functions, plus the graphical nature of the network model itself, BayesiaLab becomes an instructional device that can effectively retrieve and communicate the knowledge contained within the Bayesian network. As such, BayesiaLab becomes a bridge between artificial intelligence and human intelligence.

Watch the video online

Chapter 4: Knowledge Modeling and Probabilistic Reasoning

This chapter introduces a practical workflow for encoding expert knowledge and conducting omnidirectional probabilistic inference in the context of a real-world reasoning problem. While Chapter 1 outlined the general motivation for using Bayesian networks as an analytical framework, this chapter emphasizes their often overlooked relevance for modeling everyday reasoning. The example illustrates that what we perceive as common sense reasoning can, in fact, be quite complex. Yet representing such knowledge in a Bayesian network proves to be remarkably straightforward. Our goal is to show that reasoning with Bayesian networks can be as intuitive and accessible as performing calculations in a spreadsheet.

Background & Motivation

Complexity & Cognitive Challenges

Reasoning in complex environments presents significant cognitive challenges for humans. Uncertainty in our observations, or in the structure of the domain itself, further complicates the task. When such uncertainty obscures the underlying premises, it becomes particularly difficult to establish a shared reasoning framework among stakeholders.

No Data, No Analytics

If we had concrete data from the domain, it would be straightforward to develop a traditional analytical model for decision support. In practice, however, we often encounter fragmented datasets–or sometimes no data at all. In such cases, we must rely on the judgments of individuals who are familiar with the problem to varying degrees.

To an Analyst With Excel, Every Problem Looks Like Arithmetic

In the business world, spreadsheets are commonly used to model relationships among variables within a problem domain. When hard data are lacking, experts are often expected to provide assumptions in place of observations. This expert knowledge is typically encoded as single-point estimates and formulas. However, relying solely on fixed values and equations can oversimplify the complexity of the domain. First, it renders the variables and their relationships deterministic. Second, the structure of formulas, defined by a left-hand side and a right-hand side, restricts inference to a single direction.

Taking No Chances!

Since the cells and formulas in spreadsheets are deterministic and operate using single-point values, they are well-suited to encoding "hard" logic. However, they struggle with "soft" probabilistic knowledge, which inherently involves uncertainty. As a result, any uncertainty must be managed through workarounds such as testing multiple scenarios or using simulation add-ons.

It Is a One-Way Street!

However, the lack of omnidirectional inference may be the more fundamental limitation of spreadsheets. As soon as we create a formula linking two cells, e.g., $B1 = function(A1)$, we preclude evaluation in the opposite direction, from *B1* to *A1*.

Even assuming that *A1* is the cause and *B1* the effect, we can only use a spreadsheet for inference in the causal direction (i.e., simulation). Yet unidirectionality remains a concern even when the causal direction is known. For instance, if we can observe only the effect *B1*, we are unable to infer the cause *A1*–that is, we cannot perform diagnostic reasoning. This one-way structure is inherent to spreadsheet computations.

Bayesian Networks to the Rescue!

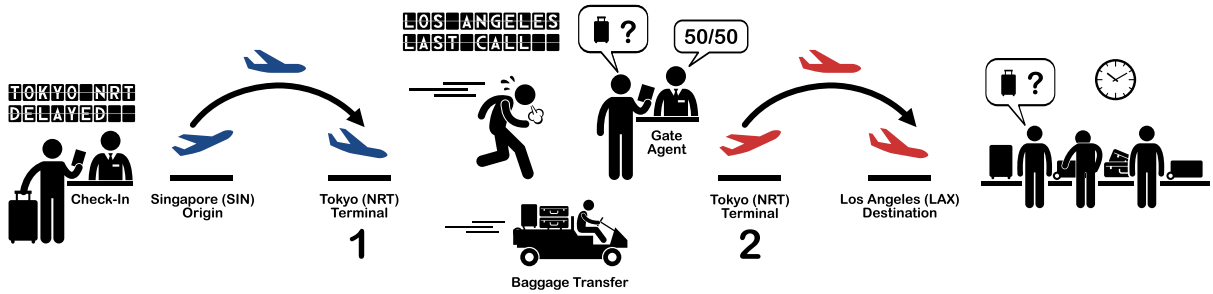
Bayesian networks are probabilistic by design and handle uncertainty natively. A Bayesian network model can work directly with probabilistic inputs and relationships, producing correctly computed probabilistic outputs. In contrast to traditional models and spreadsheets, which are typically expressed as $y = f(x)$, Bayesian networks do not require a strict distinction between independent and dependent variables. Instead, they represent the full joint probability distribution of the system under study. This representation enables omnidirectional inference, which is often essential for reasoning in complex problem domains–such as the one presented in this chapter.

Example: Where is My Bag?

While most examples in this book are drawn from formal research contexts, we introduce probabilistic reasoning with Bayesian networks through a more casual, everyday scenario. It reflects a familiar situation from daily life–one where a "common-sense" interpretation may seem more natural than a formal analytic approach. As we shall see, however, using formal methods to represent informal knowledge offers a powerful foundation for reasoning under uncertainty.

Did My Checked Luggage Make the Connection?

Most travelers will be familiar with the following scenario, or something like it: You're flying from Singapore to Los Angeles with a connecting flight in Tokyo. Your flight from Singapore is significantly delayed, and you arrive in Tokyo with just enough time to make the connection. You sprint from Terminal 1, where you landed, to Terminal 2, where your flight to Los Angeles is already boarding as you reach the gate.



Problem #1

Out of breath, you check in with the gate agent, who tells you that your checked luggage from Singapore may or may not have made the connection. She apologetically says there's only a 50/50 chance that your bag will arrive in Los Angeles.

After landing in Los Angeles, you head straight to baggage claim. Bags begin arriving steadily on the carousel. Five minutes go by, and you watch other passengers collect their luggage. You start to wonder: what are the chances that your bag will appear?

You reason that if your bag did make it onto the plane, it becomes more likely to show up as time passes. But you still don't know whether it was ever loaded. Should you wait longer? Or is it time to get in line at the baggage office to file a claim? How should you update your expectations as the minutes tick by and your bag hasn't yet appeared?

Problem #2

As you contemplate your next move, you see a colleague picking up his suitcase. As it turns out, your colleague was traveling on the very same itinerary as you, i.e., Singapore - Tokyo - Los Angeles. His luggage made it, so you conclude that you better wait at the carousel for the very last piece to be delivered. How does the observation of your colleague's suitcase change your belief in the arrival of your bag? Does all that even matter? After all, the bag either made the connection or not. The fact that you now observe something after the fact cannot influence what happened earlier, right?

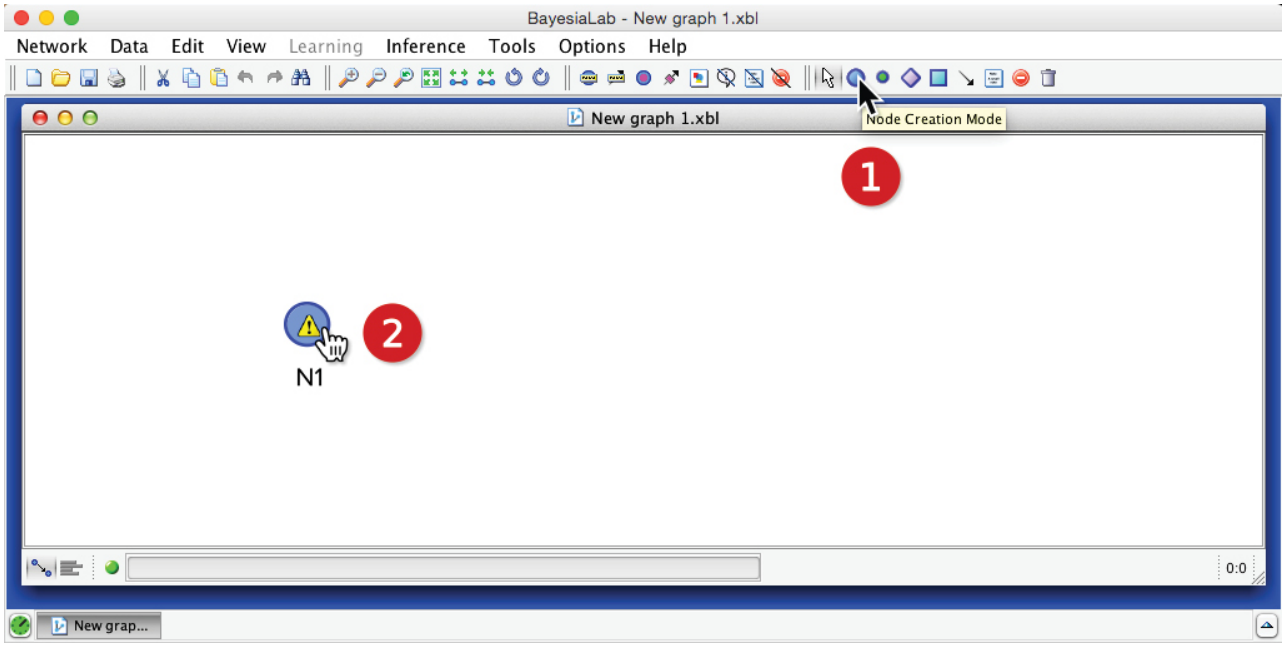
Knowledge Modeling for Problem #1

This problem domain can be explained by a causal Bayesian network using only a few commonsense assumptions. We demonstrate how to combine different pieces of available — but uncertain — knowledge into a network model. Our objective is to calculate the correct degree of belief in the arrival of your luggage as a function of time and your own observations.

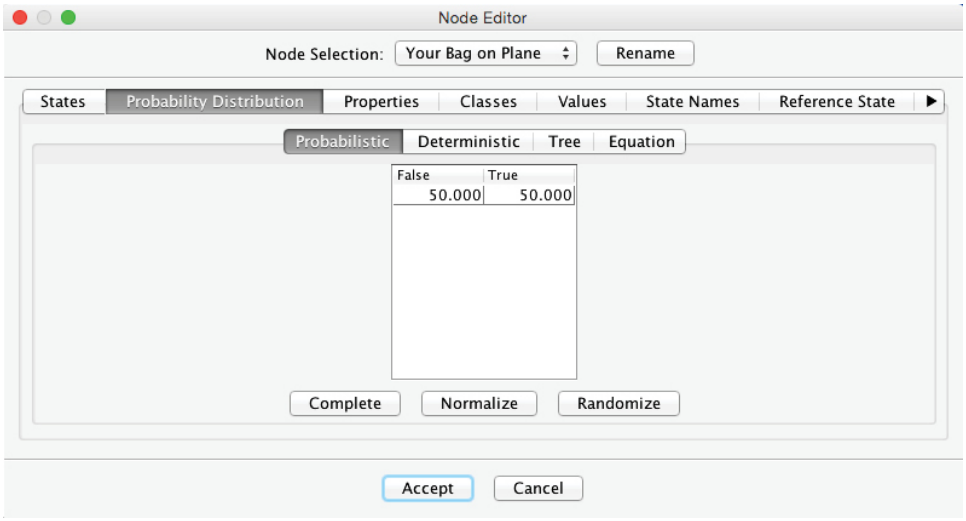
Per our narrative, we obtain the first piece of information from the gate agent in Tokyo who manages the departure to Los Angeles. She says there is a 50/50 chance that your bag is on the plane. More formally, we express this as:

$P(\textit{Your Bag on Plane} = \text{TRUE}) = 0.5$

We encode this probabilistic knowledge in a Bayesian network by creating a node. In BayesiaLab, we click the Node Creation Mode icon ● and then point to the desired position on the Graph Panel.



Once the node is in place, we update its name to *Your Bag on Plane* by double-clicking the default name *N1*. Then, by double-clicking the node itself, we open BayesiaLab's Node Editor. Under the tab `Probability Distribution > Probabilistic`, we define the probability that *Your Bag on Plane* = TRUE, which is 50%, as per the gate agent's statement. Given that these probabilities do not depend on any other variables, we speak of marginal probabilities. Note that in BayesiaLab, probabilities are always expressed as percentages:



Assuming there is no other opportunity for losing luggage within the destination airport, your chance of ultimately receiving your bag should be identical to the probability of your bag being on the plane, i.e., on the flight segment to your final destination airport. More simply, if it is on the plane, then you will get it:

$$P(\textit{Your Bag on Carousel} = \textsc{true} \mid \textit{Your Bag on Plane} = \textsc{true}) = 1$$

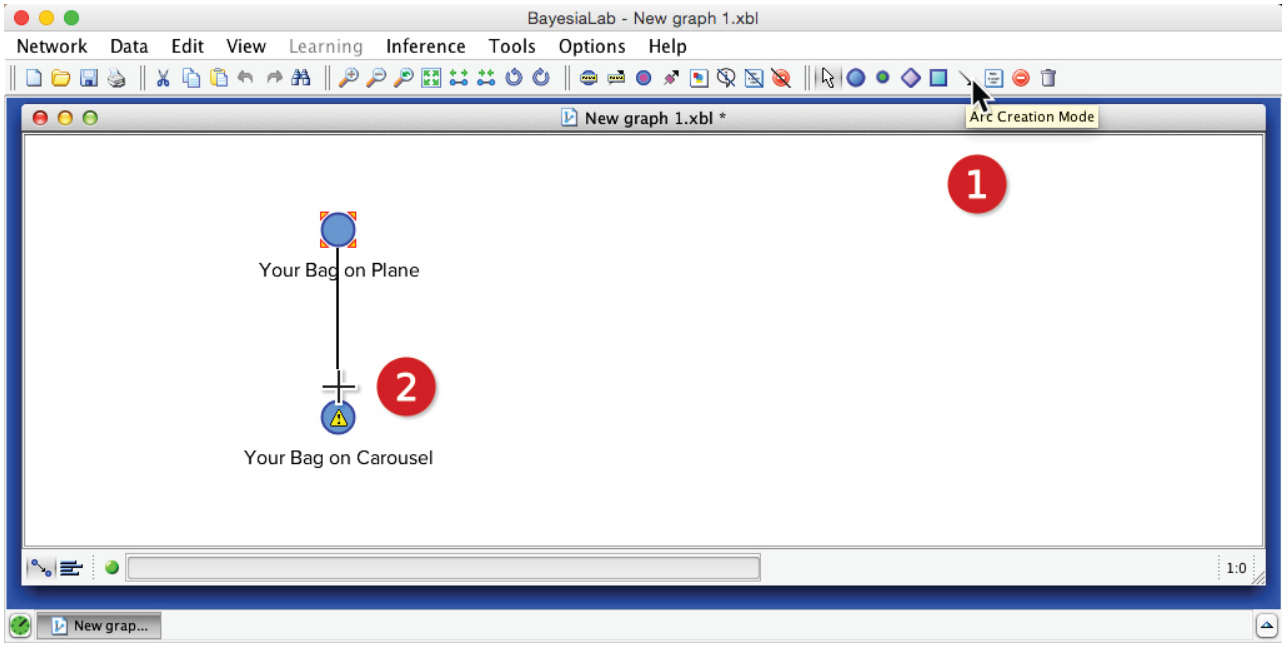
$$P(\textit{Your Bag on Carousel} = \textsc{false} \mid \textit{Your Bag on Plane} = \textsc{true}) = 0$$

Conversely, the following must hold too:

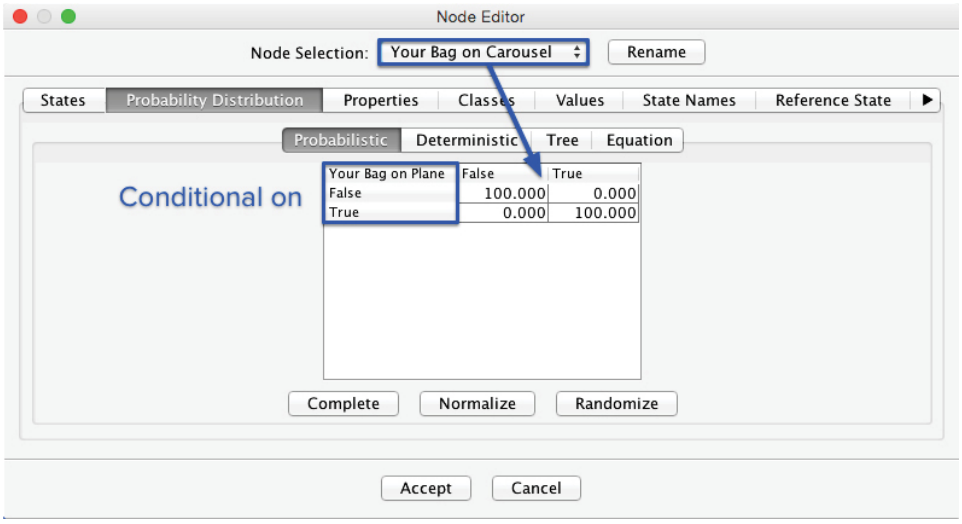
$$P(\textit{Your Bag on Carousel} = \textsc{false} \mid \textit{Your Bag on Plane} = \textsc{false}) = 1$$

$$P(\textit{Your Bag on Carousel} = \textsc{true} \mid \textit{Your Bag on Plane} = \textsc{false}) = 0$$

We now encode this knowledge into our network. We add a second node, *Your Bag on Carousel*, and then click the Arc Creation Mode icon ↘. Next, we click and hold the cursor on *Your Bag on Plane*, drag the cursor to *Your Bag on Carousel*, and finally release. This produces a simple, manually specified Bayesian network:



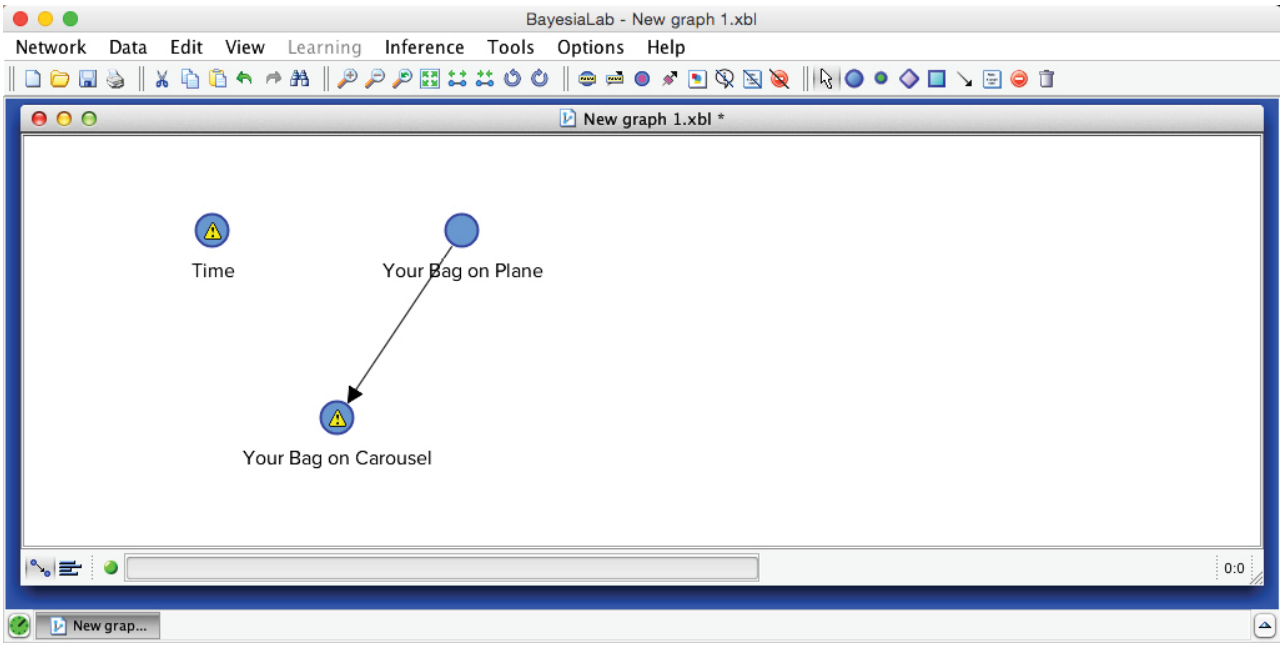
The yellow warning triangle ⚠ indicates that probabilities need to be defined for the node *Your Bag on Carousel*. Unlike the previous instance, where we only had to enter marginal probabilities, we now need to define the probabilities of the states of the node *Your Bag on Carousel* conditional on the states of *Your Bag on Plane*. In other words, we need to fill the Conditional Probability Table to quantify this parent-child relationship. We open the Node Editor and enter the values from the equations above.



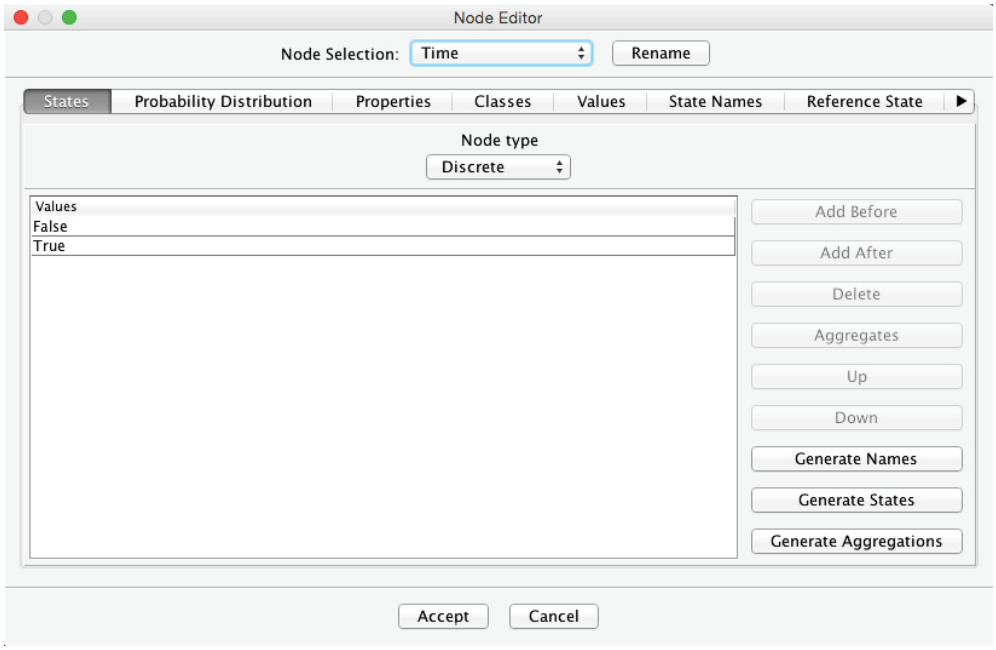
Introduction of Time

Now we add another piece of contextual information that has not been mentioned yet in our story. From the baggage handler who monitors the carousel, you learn that 100 pieces of luggage in total were on your final flight segment, from the hub to the destination. After you wait for one minute, 10 bags have appeared on the carousel, and they keep coming out at a very steady rate. However, yours is not among the first ten that were delivered in the first minute. At the current rate, it would now take 9 more minutes for all bags to be delivered to the baggage carousel.

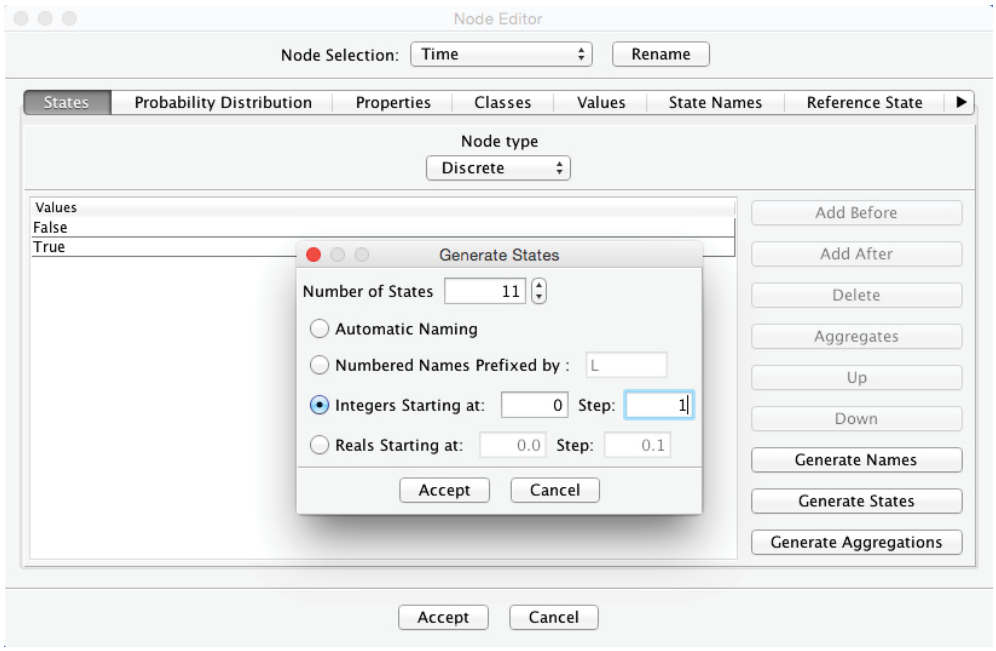
Given that your bag was not delivered in the first minute, what is your new expectation of ultimately getting your bag? How about after the second minute of waiting? Quite obviously, we need to introduce a time variable into our network. We create a new node *Time* and define discrete time intervals [0,...,10] to serve as its states.



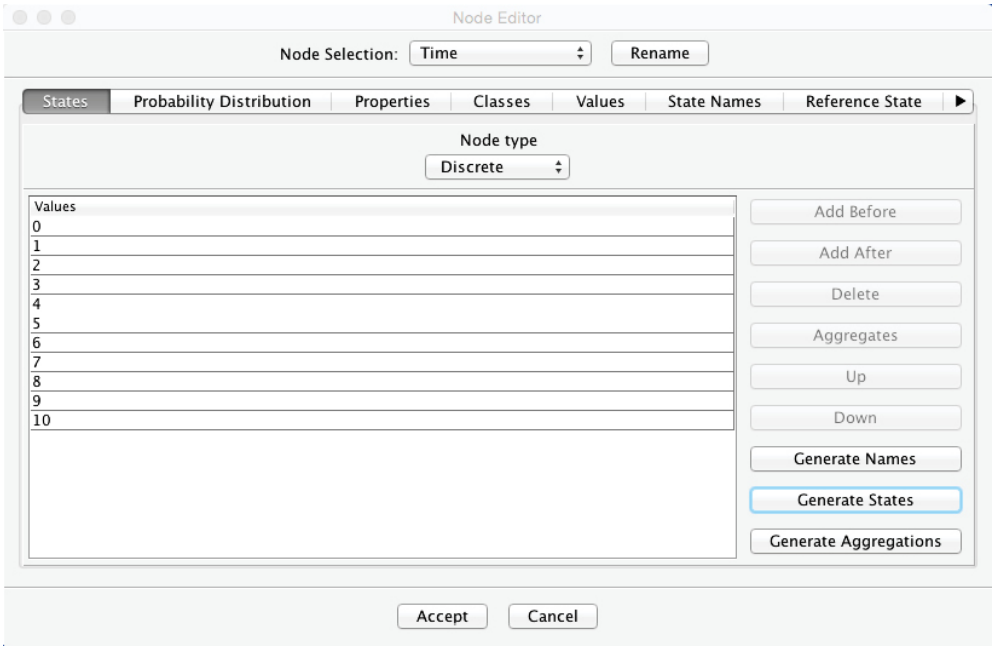
By default, all new nodes initially have two states, TRUE and FALSE. We can see this by opening the Node Editor and selecting the `States` tab:



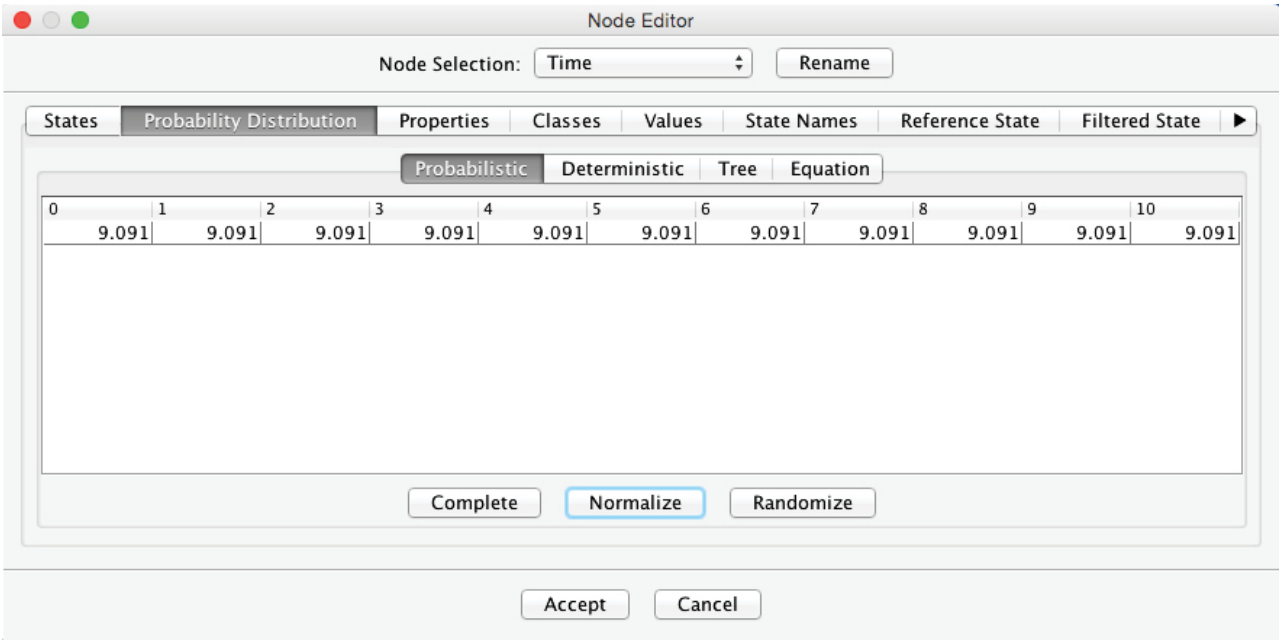
By clicking on the `Generate States` button, we create the states we need for our purposes. Here, we define 11 states, starting at 0 and increasing by 1 step:



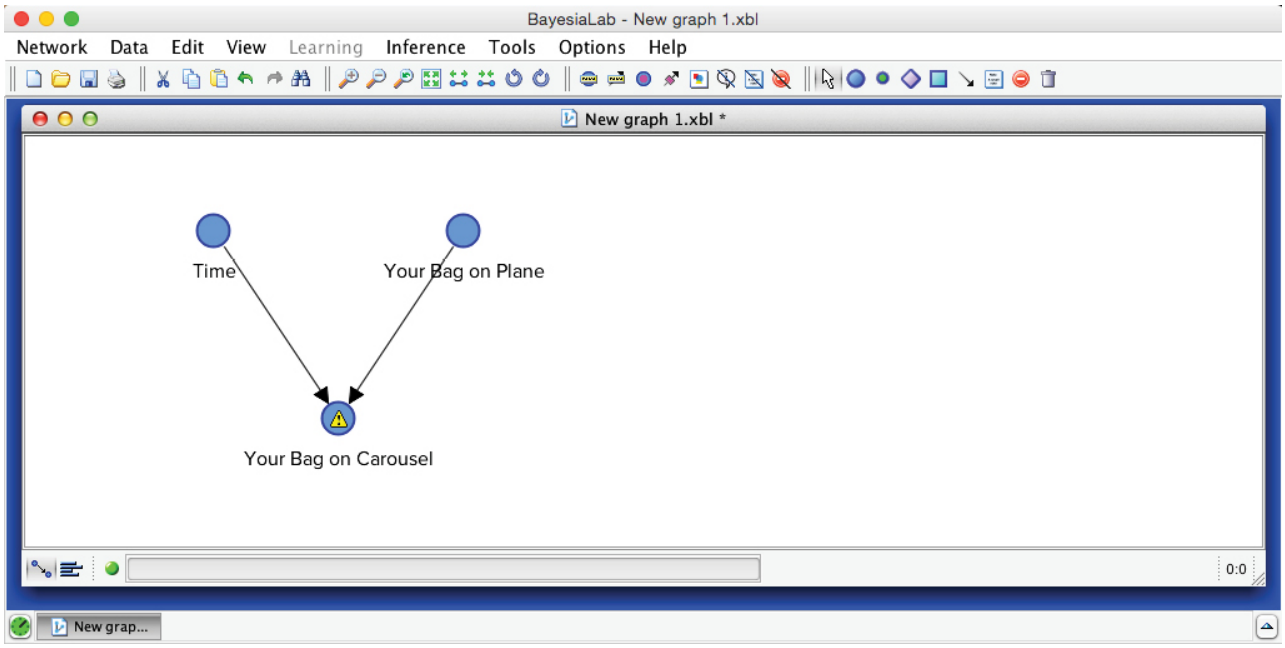
The Node Editor now shows the newly-generated states:



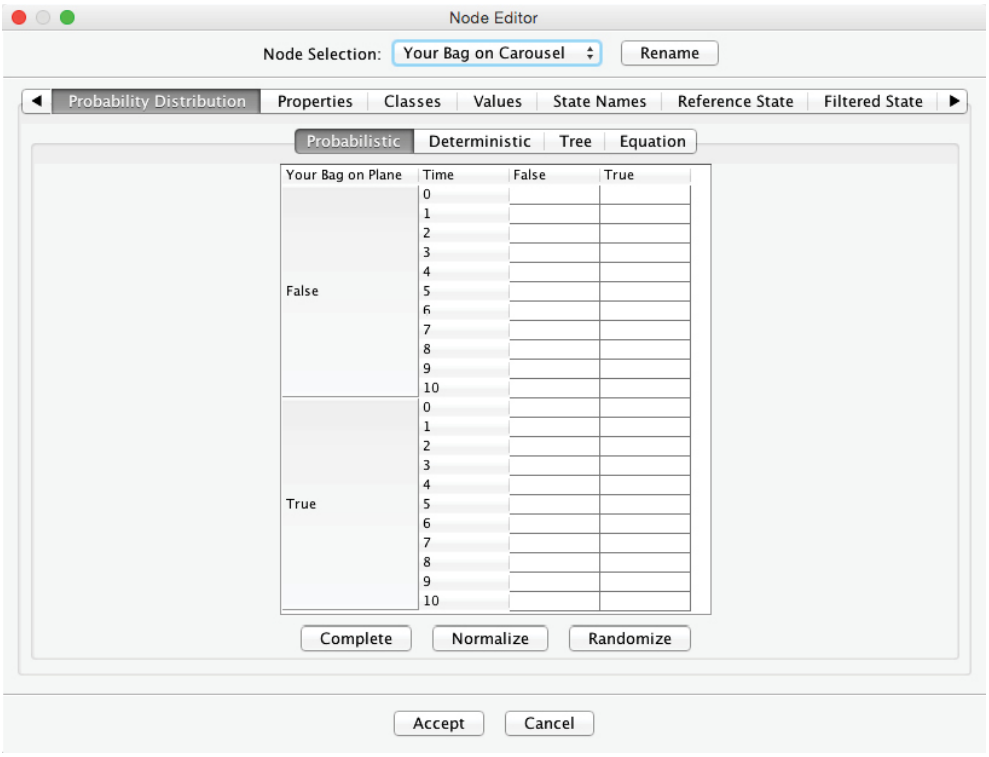
Beyond defining the states of *Time*, we also need to define their marginal probability distribution. For this, we select the tab `Probability Distribution > Probabilistic`. Naturally, no time interval is more probable than another one, so we should apply a uniform distribution across all states of *Time*. BayesiaLab provides a convenient shortcut for this purpose. Clicking the `Normalize` button places a uniform distribution across all cells, i.e., 9.091% per cell.



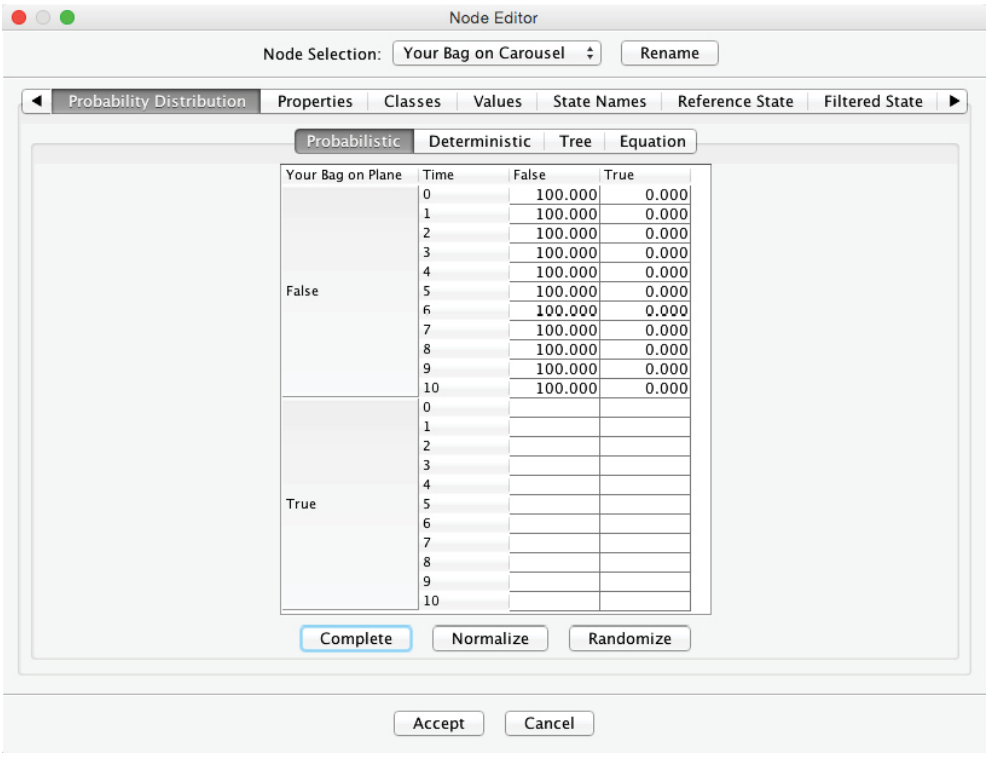
Once *Time* is defined, we draw an arc from *Time* to *Your Bag on Carousel*. By doing so, we introduce a causal relationship, stating that *Time* influences the status of your bag.



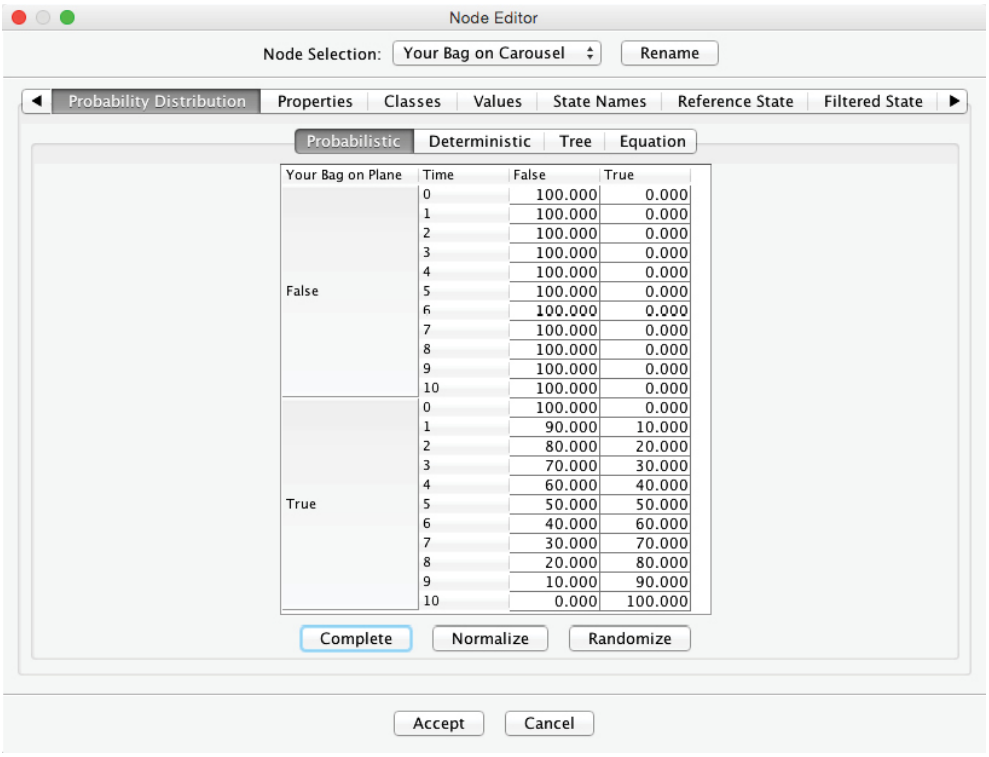
The warning triangle ⚠ once again indicates that we need to define further probabilities concerning *Your Bag on Carousel*. We open the Node Editor to enter these probabilities into the Conditional Probability Table:



Note that the probabilities of the states TRUE and FALSE now depend on two-parent nodes. For the upper half of the table, it is still quite simple to establish the probabilities. If the bag is not on the plane, it will not appear on the baggage carousel under any circumstance, regardless of *Time*. Hence, we set FALSE to 100 (%) for all rows in which *Your Bag on Plane* = FALSE.



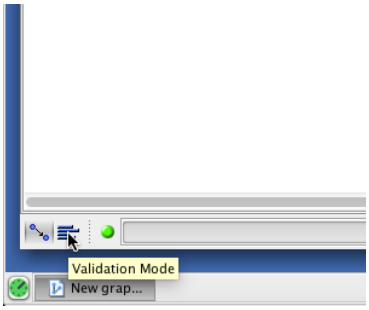
However, given that *Your Bag on Plane* = TRUE, the probability of seeing it on the carousel depends on the time elapsed. Now, what is the probability of seeing your bag at each time step? Assuming that all luggage is shuffled extensively through the loading and unloading processes, there is a uniform probability distribution that the bag is anywhere in the pile of luggage to be delivered to the carousel. As a result, there is a 10% chance that your bag is delivered in the first minute, i.e., within the first batch of 10 out of 100 luggage pieces. Over a period of two minutes, there is a 20% probability that the bag arrives, and so on. Only when the last batch of 10 bags remains undelivered, we can be certain that your bag is in the final batch, i.e. there is a 100% probability of the state TRUE in the tenth minute. We can now fill out the Conditional Probability Table in the Node Editor with these values. Note that we only need to enter the values in the TRUE column and then highlight the remaining empty cells. Clicking `Complete` prompts BayesiaLab to automatically fill in the FALSE column to achieve a row sum of 100%:



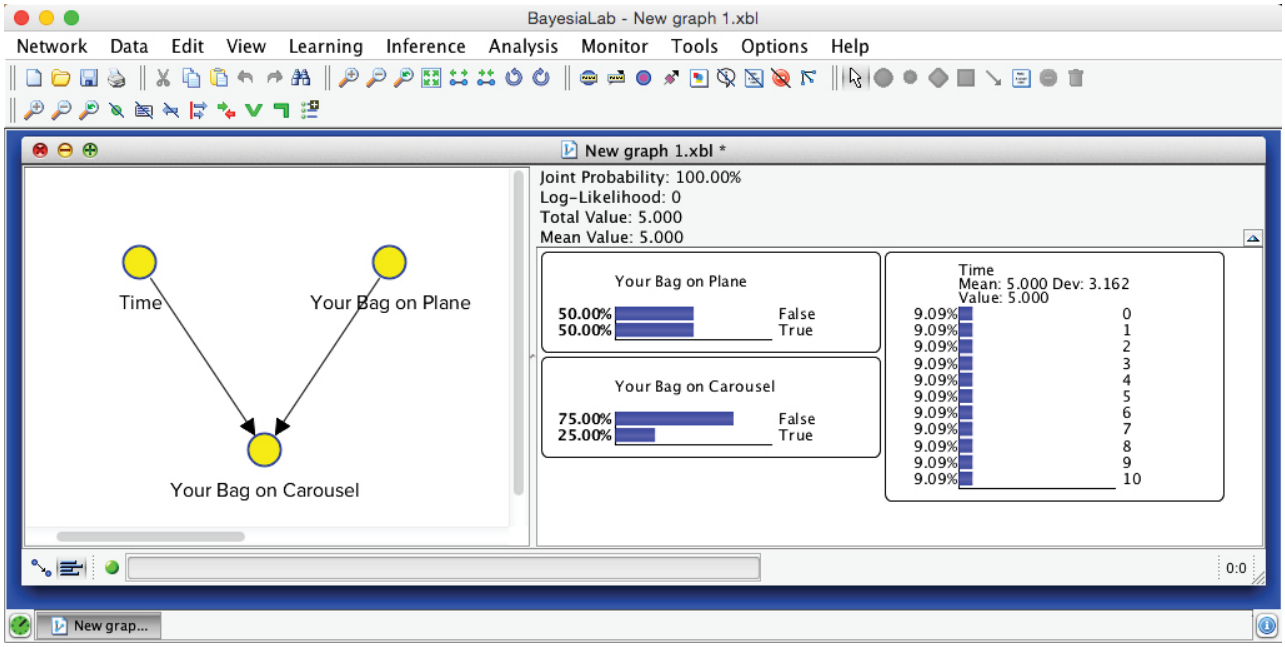
Now we have a fully specified Bayesian network, which we can evaluate immediately.

Evidential Reasoning for Problem #1

BayesiaLab's Validation Mode F5 provides the tools for using the Bayesian network we built for omnidirectional inference. We switch to the Validation Mode F5 via the corresponding icon in the lower left-hand corner of the main window, or via the keyboard shortcut F5:



Upon switching to this mode, we double-click on all three nodes to bring up their associated Monitors, which show the nodes' current marginal probability distributions. We find these Monitors inside the Monitor Panel on the right-hand side of the Graph Window:

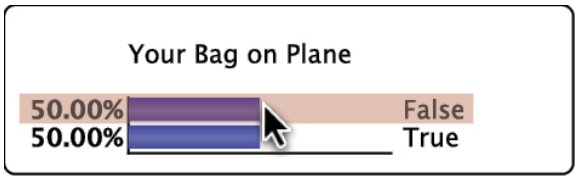


Inference Tasks

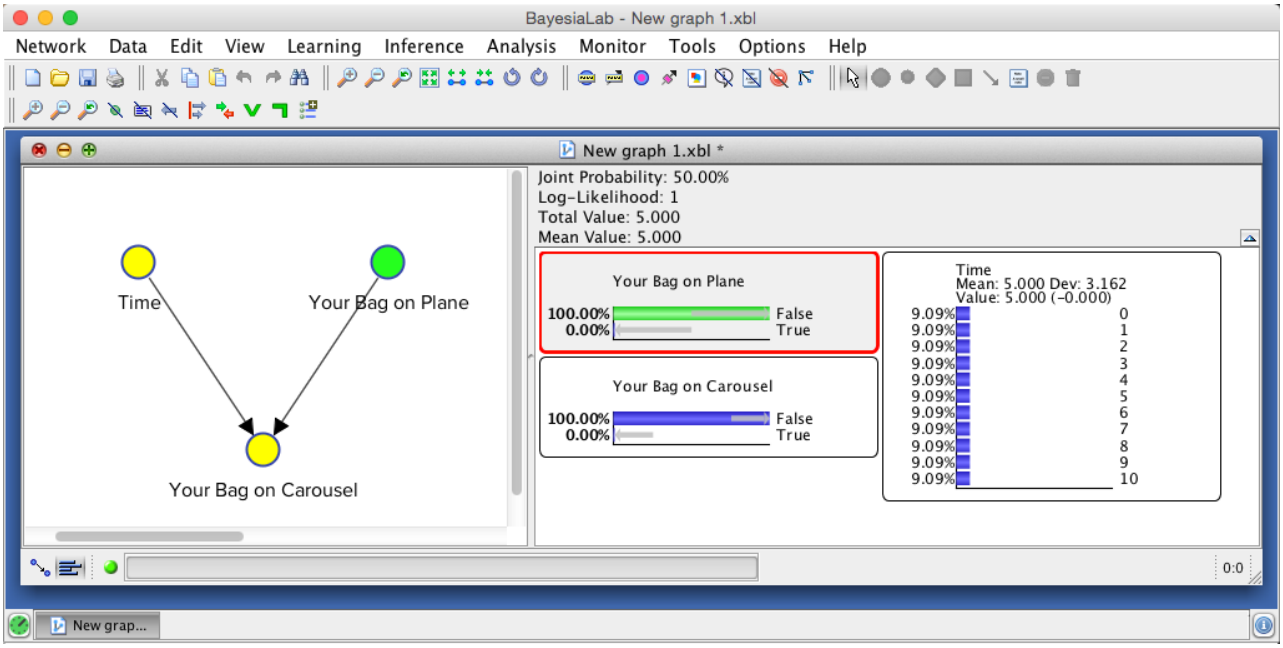
If we filled the Conditional Probability Table correctly, we should now be able to validate at least the trivial cases straight away, e.g. for *Your Bag on Plane* = FALSE.

Inference from Cause to Effect: *Your Bag on Plane* = FALSE

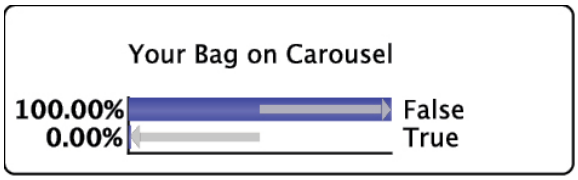
We perform inference by setting such evidence via the corresponding Monitor in the Monitor Panel. We double-click the bar that represents the State FALSE:



The setting of the evidence turns the node and the corresponding bar in the Monitor green:



The Monitor for *Your Bag on Carousel* shows the result. The small gray arrows overlaid on top of the horizontal bars furthermore indicate how the probabilities have changed by setting this most recent piece of evidence:



Indeed, your bag could not possibly be on the carousel because it was not on the plane in the first place. The inference we performed here is indeed trivial, but it is reassuring to see that the Bayesian network properly "plays back" the knowledge we entered earlier.

Omnidirectional Inference: *Your Bag on Carousel* = FALSE, *Time* = 1

The next question, however, typically goes beyond our intuitive reasoning capabilities. We wish to infer the probability that your bag made it onto the plane, given that we are now in minute 1, and the bag has not yet appeared on the carousel. This inference is tricky because we now have to reason along multiple paths in our network.

Diagnostic Reasoning

The first path is from *Your Bag on Carousel* to *Your Bag on Plane*. This type of reasoning from effect to cause is more commonly known as diagnosis. More formally, we can write:

$P(\textit{Your Bag on Plane} = \text{TRUE} \mid \textit{Your Bag on Carousel} = \text{FALSE})$

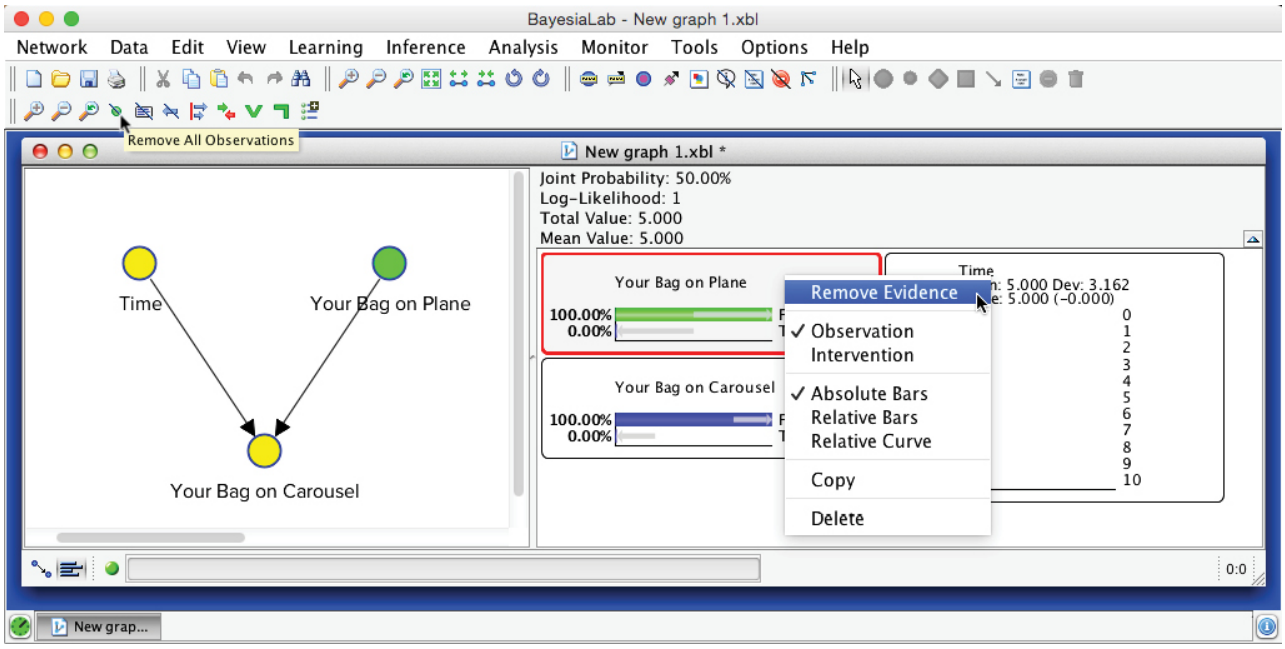
Inter-Causal Reasoning

The second reasoning path is from *Time* via *Your Bag on Carousel* to *Your Bag on Plane*. Once we condition on *Your Bag on Carousel*, i.e. by observing the value, we open this path, and information can flow from one cause, *Time*, via the common effect, *Your Bag on Carousel*, to the other cause, *Your Bag on Plane*. Hence, we speak of "inter-causal reasoning" in this context. The specific computation task is:

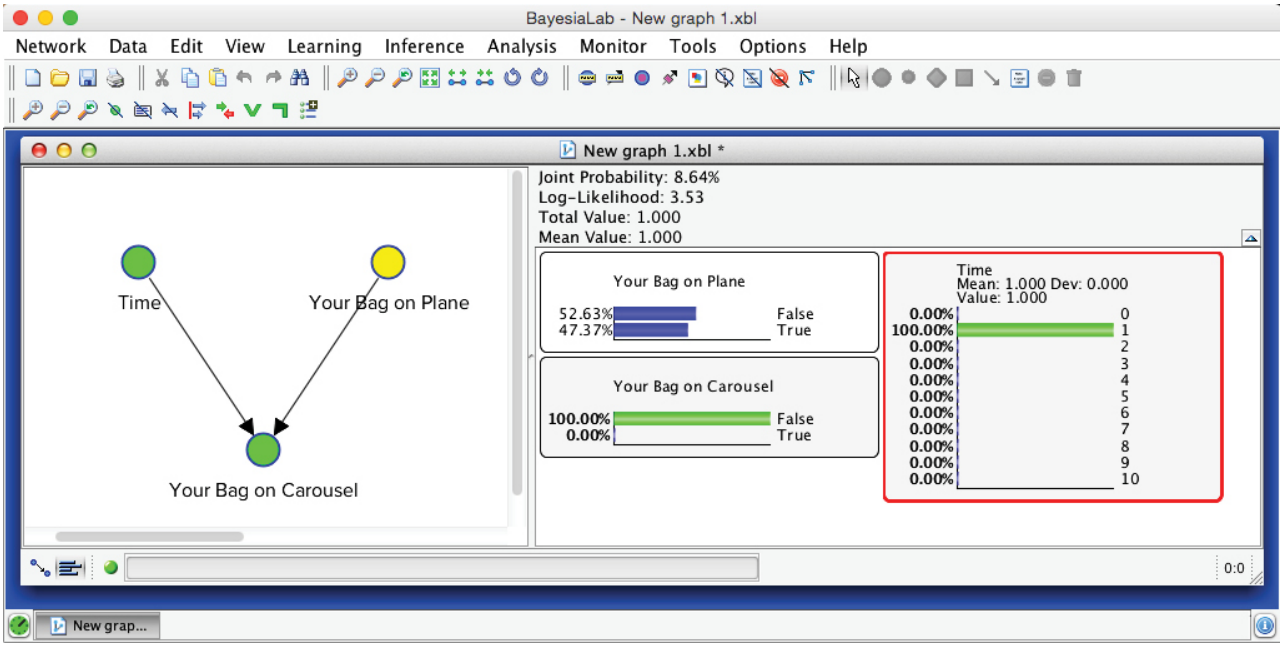
$P(\textit{Your Bag on Plane} = \text{TRUE} \mid \textit{Your Bag on Carousel} = \text{FALSE}, \textit{Time} = 1)$

Bayesian Networks as Inference Engine

How do we go about computing this probability? We do not attempt to perform this computation ourselves. Rather, we rely on the Bayesian network we built and BayesiaLab's exact inference algorithms. However, before we can perform this inference computation, we need to remove the previous piece of evidence, i.e. *Your Bag on Plane* = TRUE. We do this by right-clicking the relevant node and then selecting `Remove Evidence` from the Monitor Contextual Menu. Alternatively, we can remove all evidence by clicking the `Remove All Observations` icon .



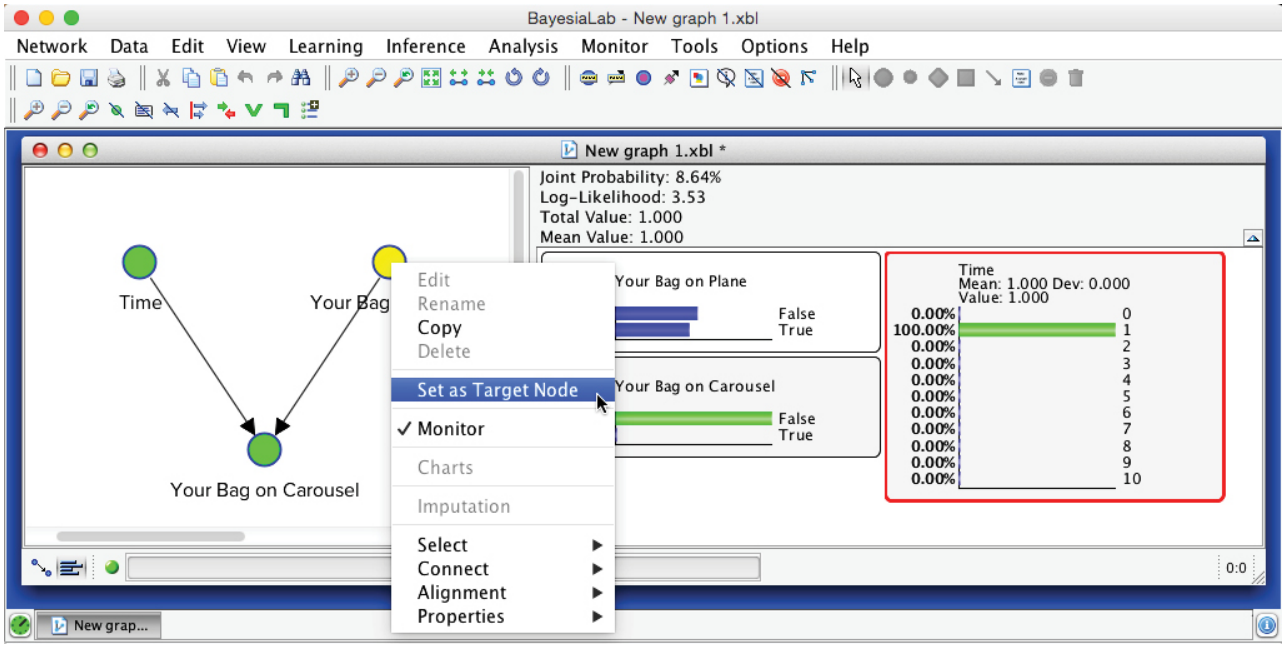
Then, we set the new observations via the Monitors in the Monitor Panel. The inference computation then happens automatically.



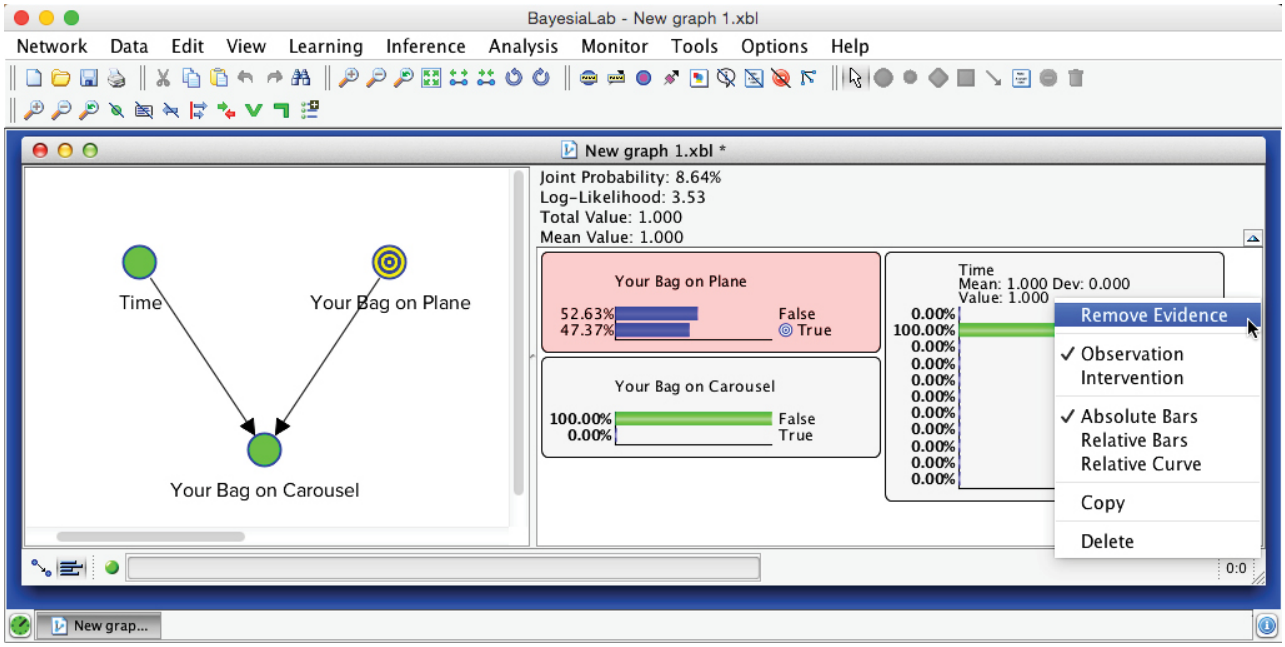
Given that you do not see your bag in the first minute, the probability that your bag made it onto the plane is now no longer at the marginal level of 50% but is reduced to 47.37%.

Inference as a Function of Time

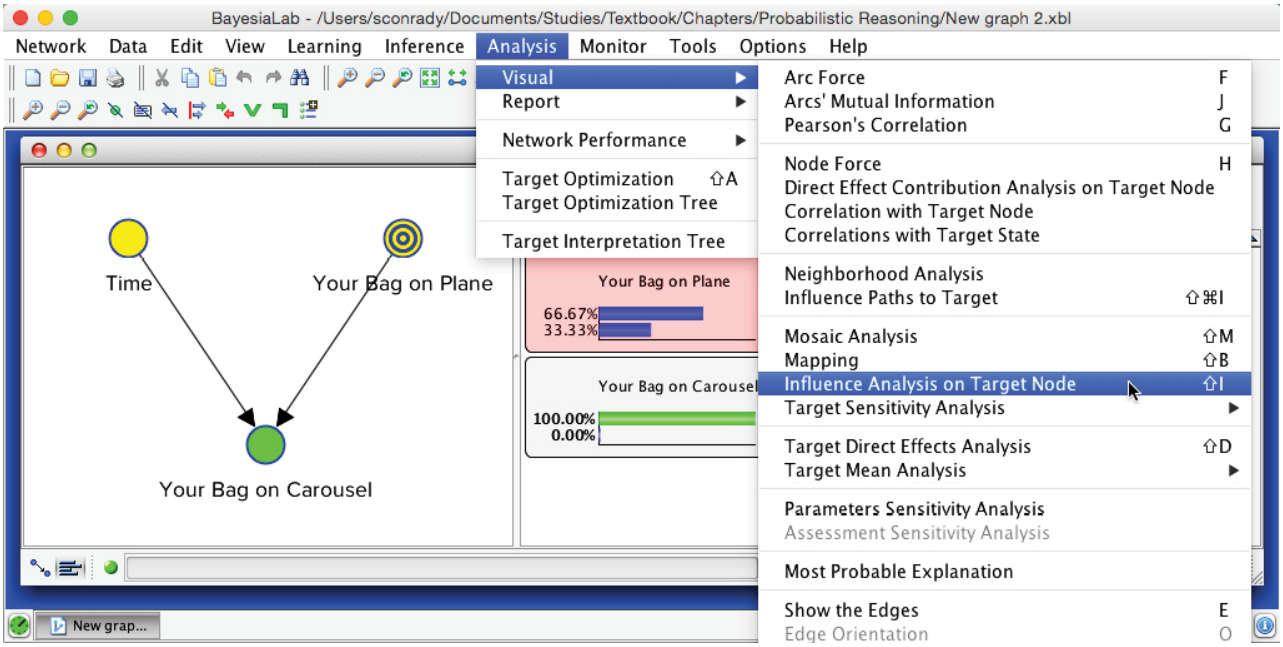
Continuing with this example, how about if the bag has not shown up in the second minute, in the third minute, etc.? We can use one of BayesiaLab's built-in visualization functions to analyze this automatically. To prepare the network for this type of analysis, we first need to set a Target Node, which, in our case, is *Your Bag on Plane*. Upon right-clicking this node, we select `Set as Target Node`. Alternatively, we can double-click the node, or one of its states in the corresponding Monitor, while holding `T`.



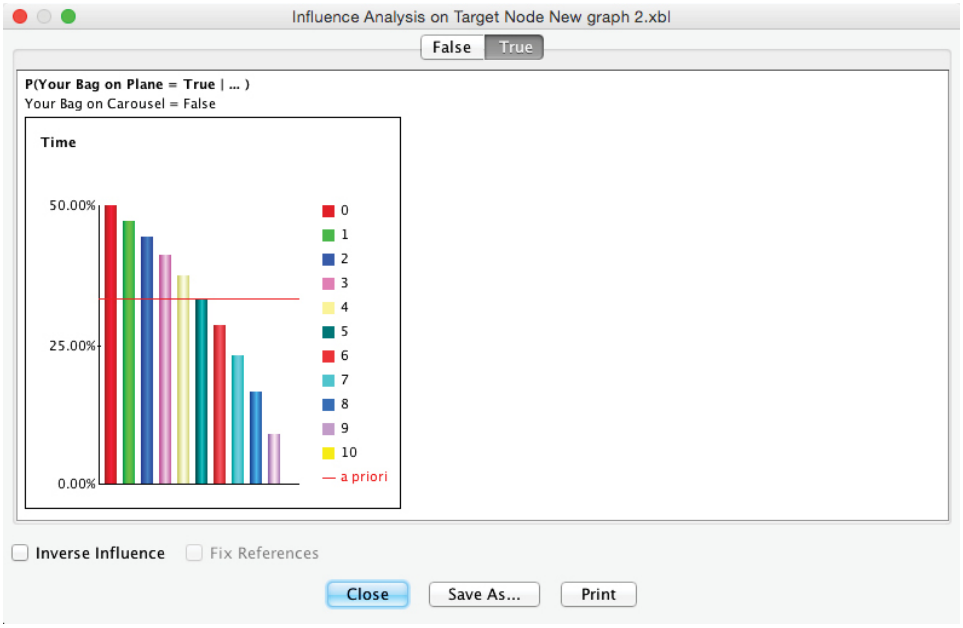
Upon setting the Target Node, *Your Bag on Plane* is marked with a bullseye symbol. Also, the corresponding Monitor is now highlighted in red. Before we continue, however, we need to remove the evidence from the Monitor of *Time*. We do so by right-clicking the Monitor and selecting `Remove Evidence` from the Monitor Context Menu.



Then, we select `Menu > Analysis > Visual > Target > Target Posterior > Histograms`.



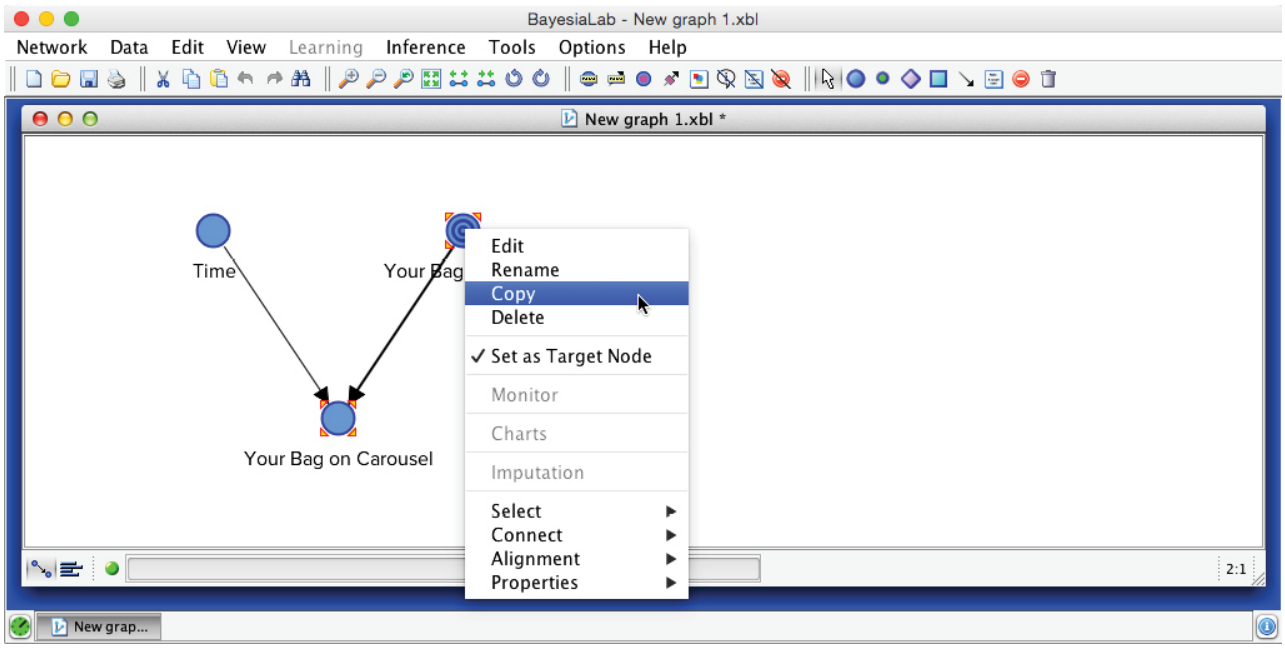
The resulting graph shows the probabilities of receiving your bag as a function of the discrete time steps. To see the progression of the TRUE state, we select the corresponding tab at the top of the window.



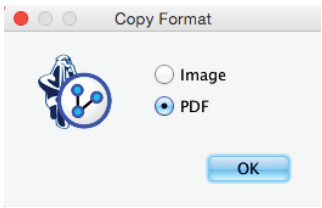
Knowledge Modeling for Problem #2

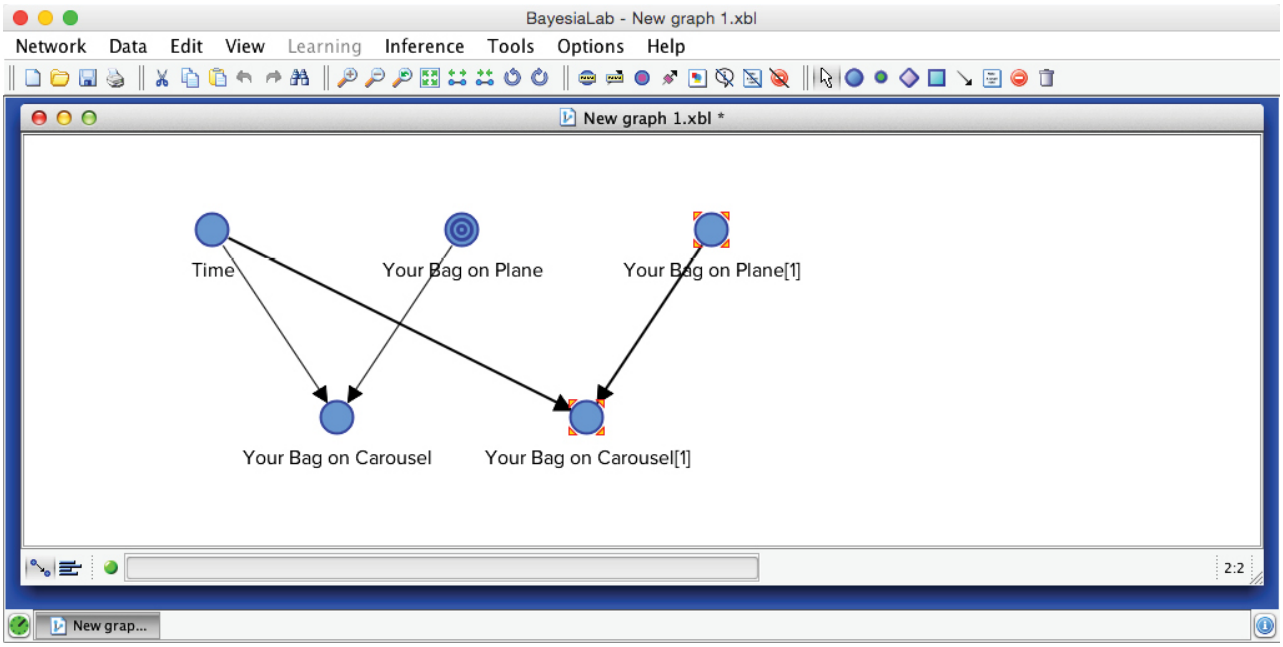
Continuing with our narrative, you now notice a colleague of yours in the baggage claim area. As it turns out, your colleague was traveling on the same itinerary as you, i.e., Singapore - Tokyo - Los Angeles, so he had to make the same tight connection. Unlike you, he has already retrieved his bag from the carousel. You assume that his luggage being on the airplane is not independent of your luggage being on the same plane, so you take this as a positive sign. How do we formally integrate this assumption into our existing network?

To encode any new knowledge, we first need to switch back to the Modeling Mode `F4`. Then, we duplicate the existing nodes *Your Bag on Plane* and *Your Bag on Carousel* by copying and pasting them into the same Graph Panel using the common shortcuts, `Ctrl+C` and `Ctrl+V`.



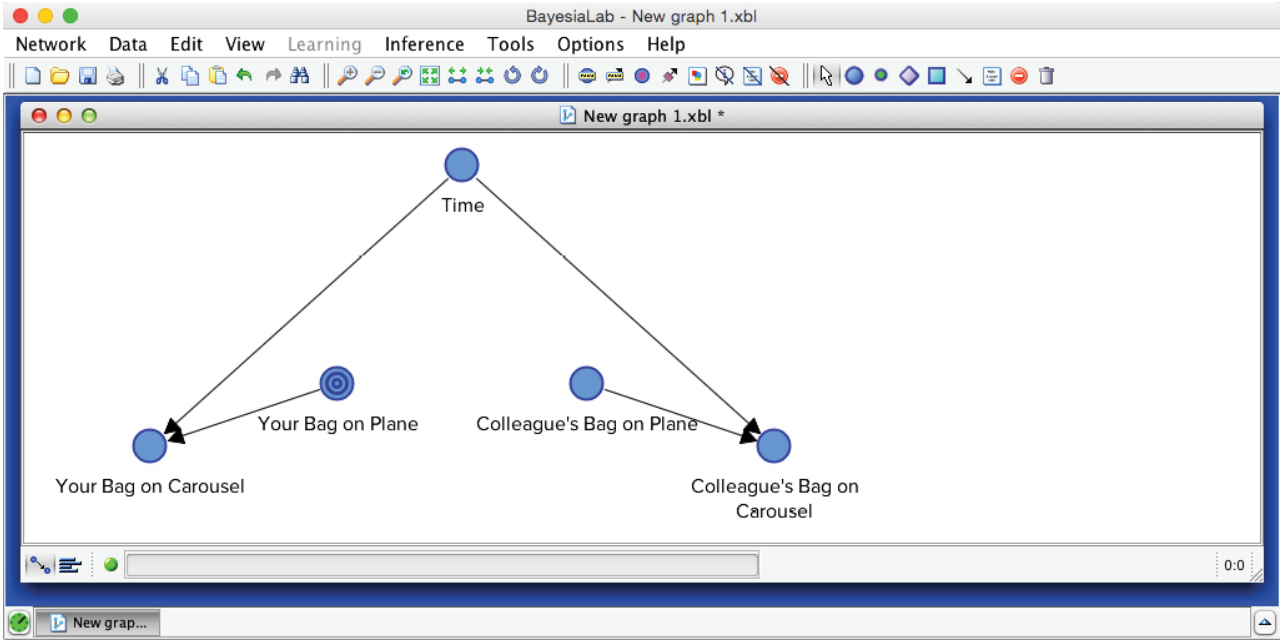
In the copy process, BayesiaLab prompts us for a Copy Format, which would only be relevant if we intended to paste the selected portion of the network into another application, such as PowerPoint. As we paste the copied nodes into the same Graph Panel, the format does not matter.



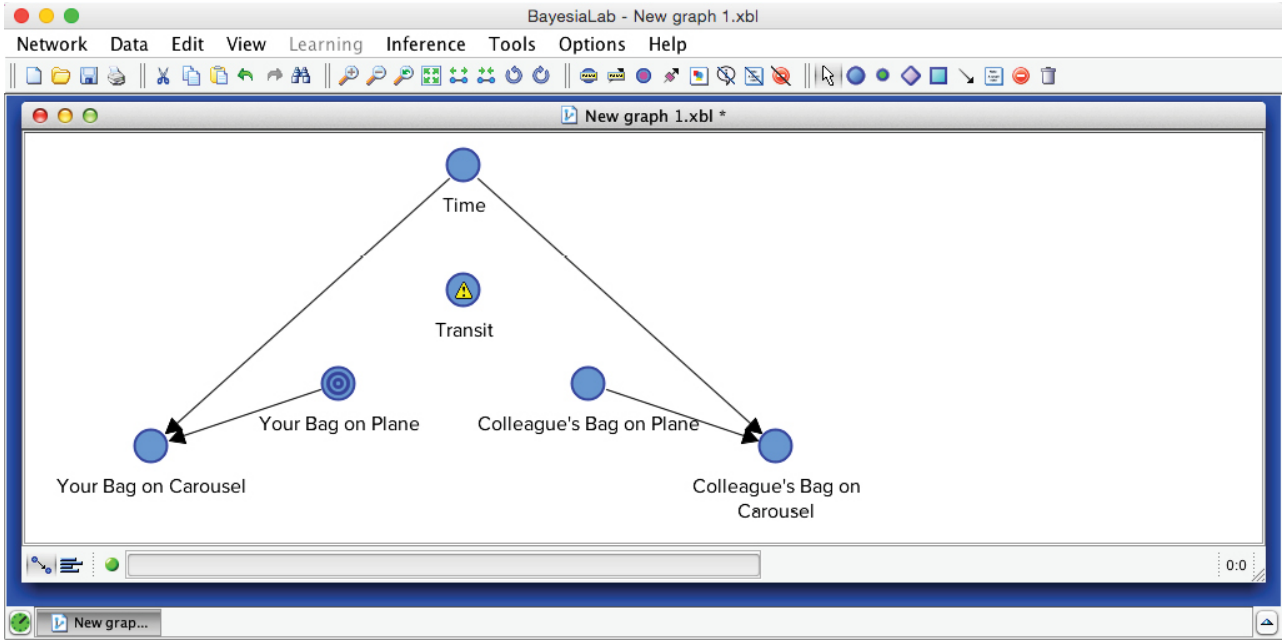


Upon pasting, by default, the new nodes have the same names as the original ones plus the suffix "[1]".

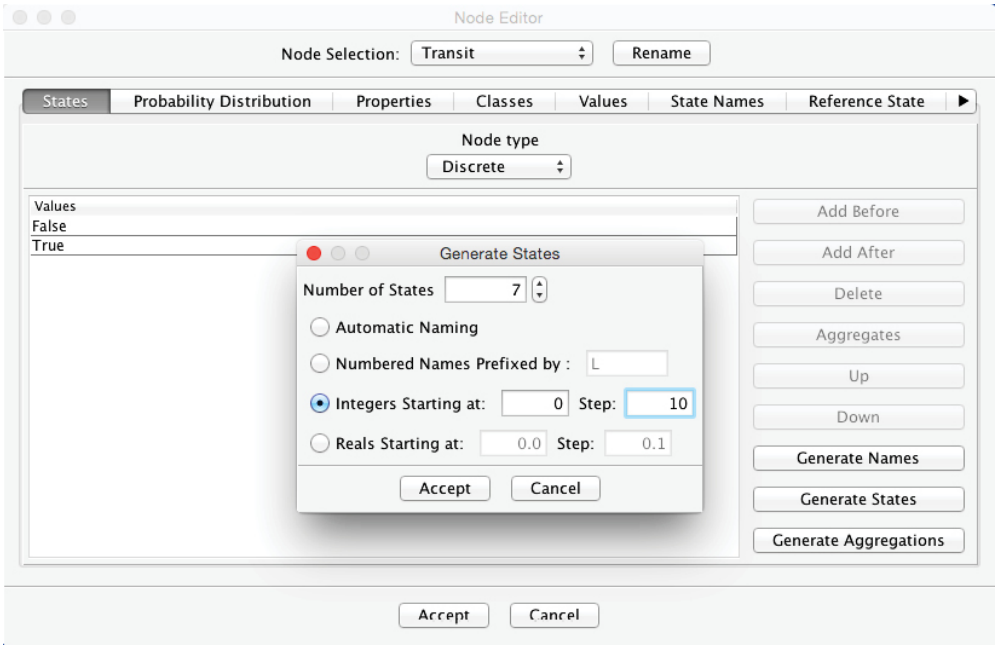
Next, we reposition the nodes on the Graph Panel and rename them to show that the new nodes relate to your colleague's situation, rather than yours. To rename the nodes we double-click the Node Names and overwrite the existing label.

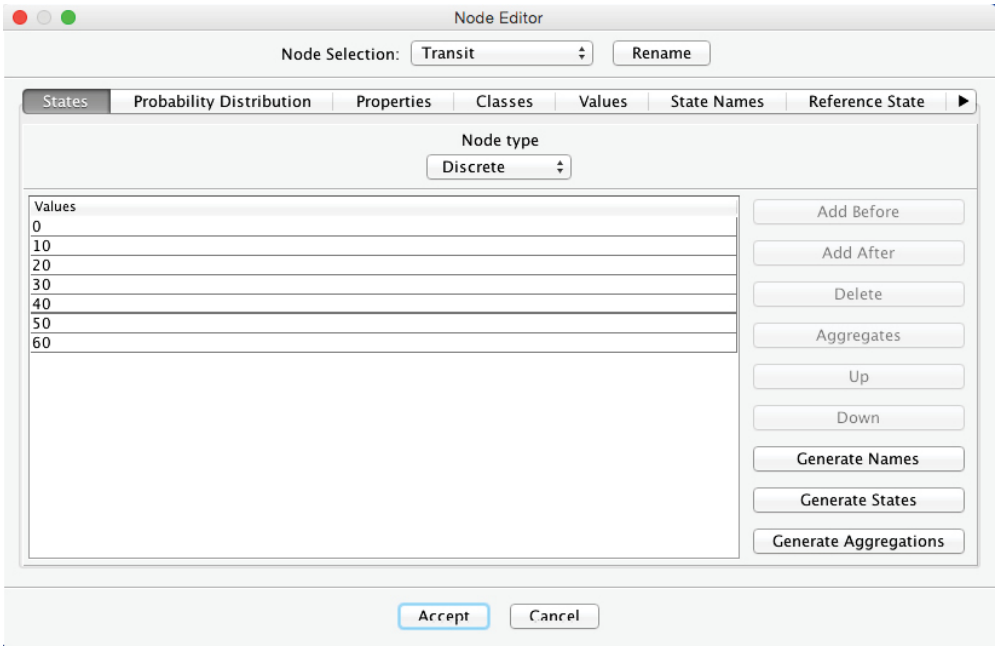


The next assumption is that your colleague's bag is subject to exactly the same forces as your luggage. More specifically, the successful transfer of your and his luggage is a function of how many bags could be processed in Tokyo given the limited transfer time. To model this, we introduce a new node and name it *Transit*.

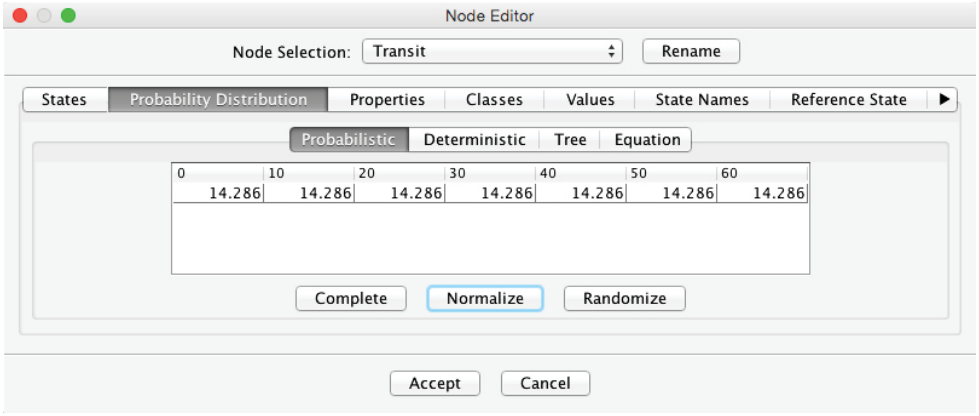


We create seven states of ten-minute intervals for this node, which reflect the amount of time available for the transfer, i.e., from 0 to 60 minutes.

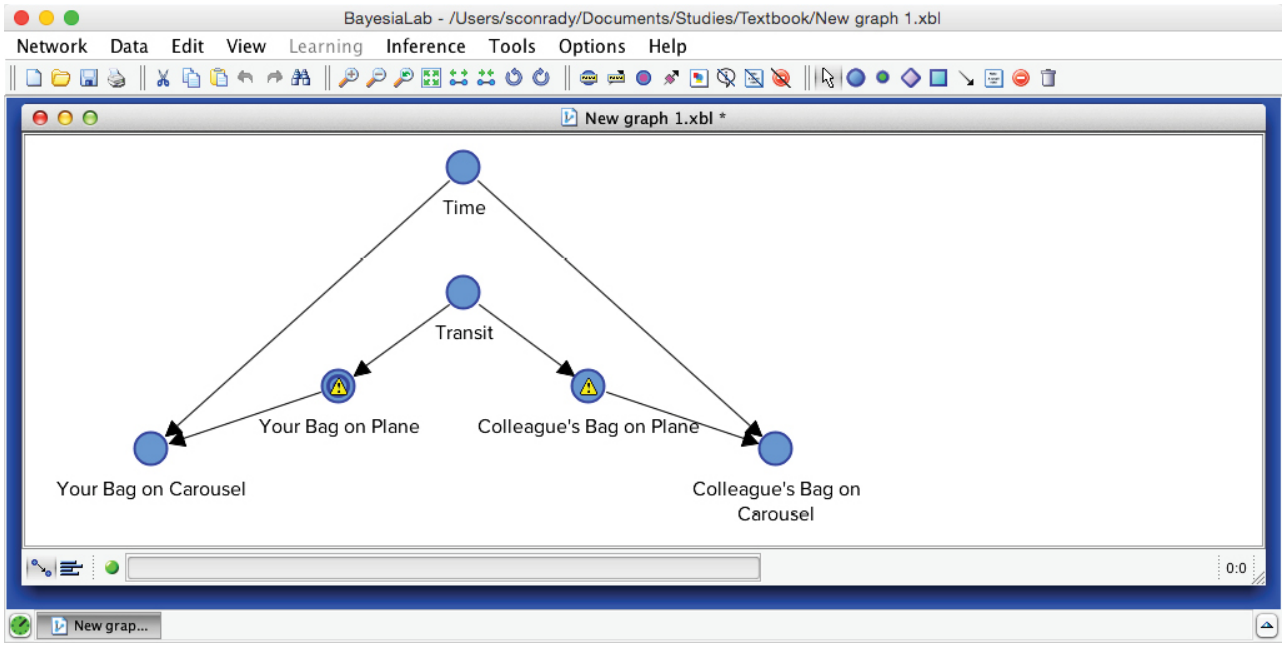




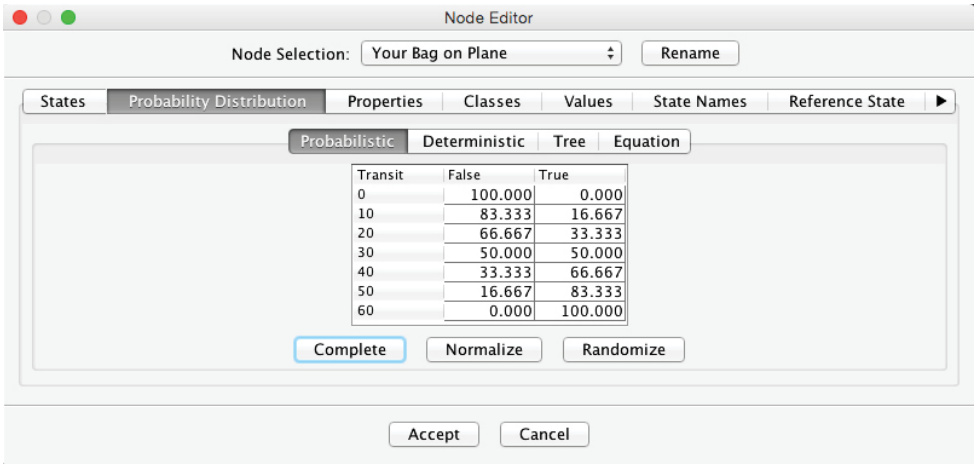
Furthermore, we set the probability distribution for *Transit*. To keep the example simple, we apply a uniform distribution using the `Normalize` button.



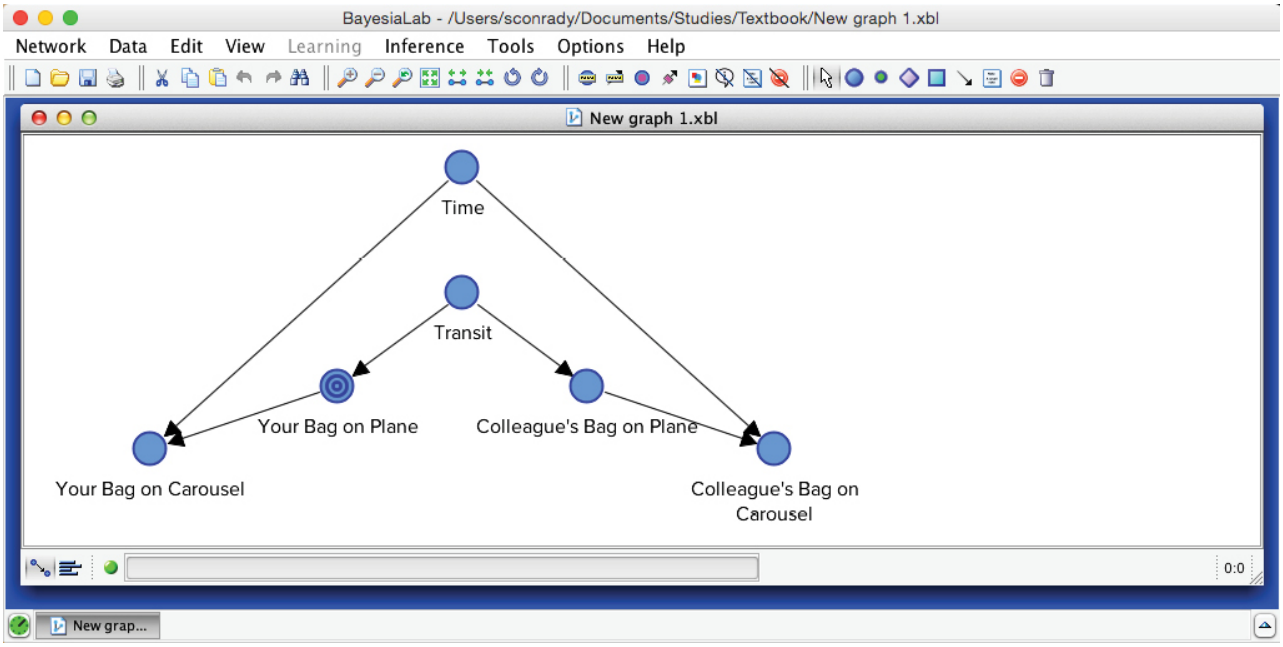
Now that the *Transit* node is defined, we can draw the arcs connecting it to *Your Bag on Plane* and *Colleague's Bag on Plane*.



The yellow warning triangles ⚠ indicate that the Conditional Probability Tables of *Your Bag on Plane* and *Colleague's Bag on Plane* have yet to be filled. Thus, we need to open the Node Editor and set these probabilities. We will assume that the probability of your bag making the connection is 0% given a *Transit* time of 0 minutes and 100% with a *Transit* time of 60 minutes. Between those values, the probability of a successful transfer increases linearly with time.

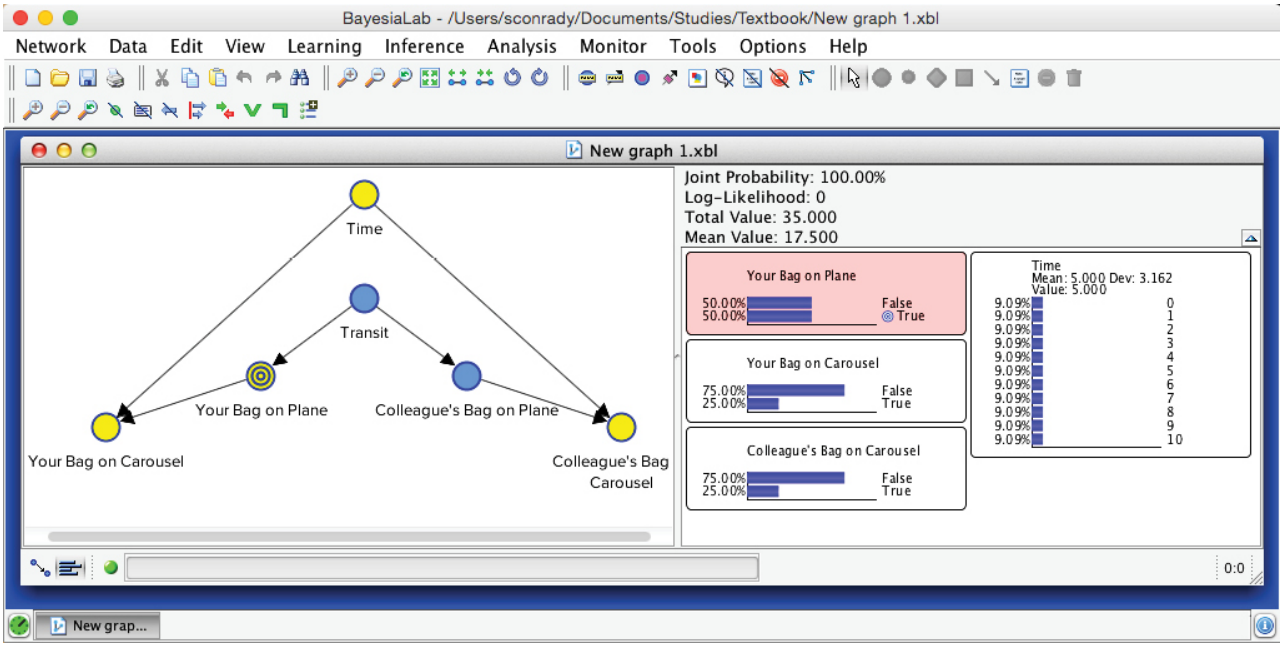


The very same function also applies to your colleague's bag, so we enter the same conditional probabilities for the node *Colleague's Bag on Plane* by copying and pasting the previously entered table.

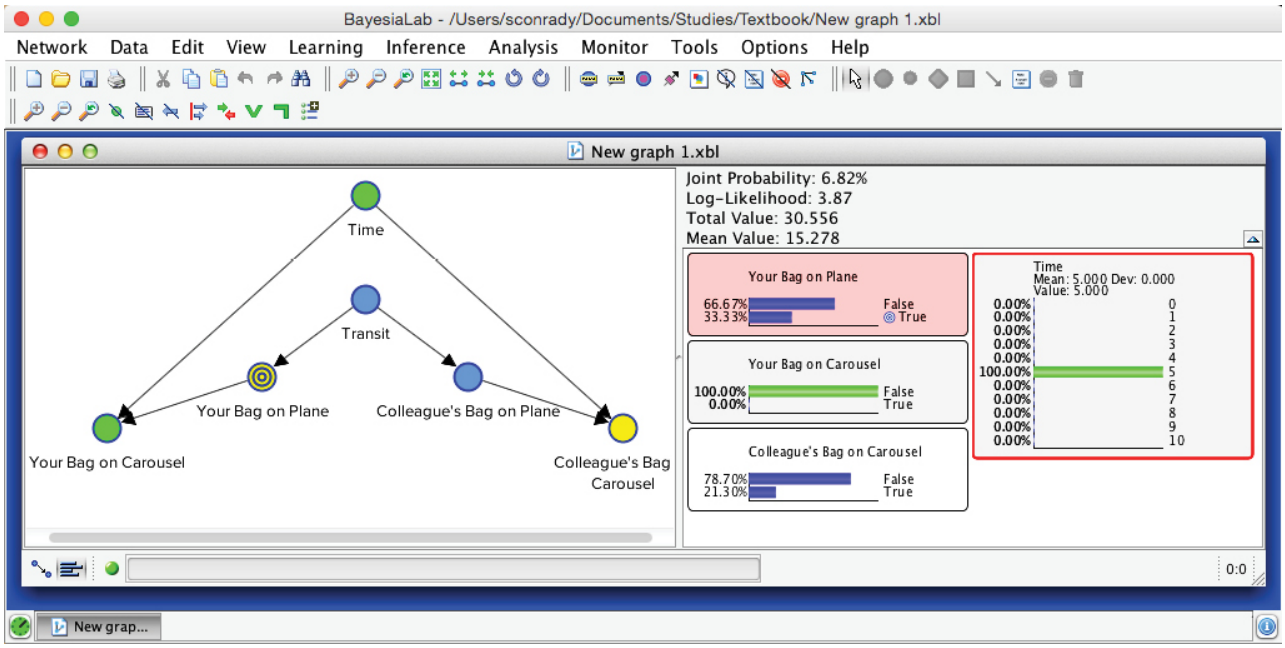


Evidential Reasoning for Problem #2

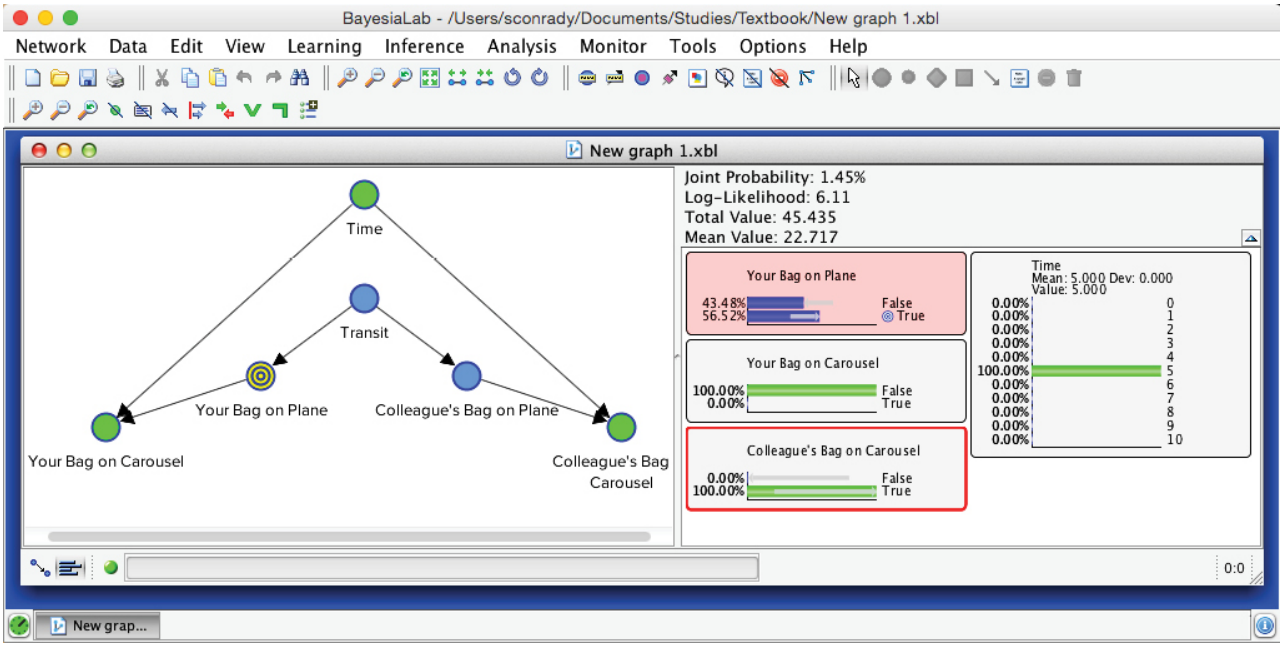
Now that the probabilities are defined, we switch to the Validation Mode F5; our updated Bayesian network is ready for inference again.



We simulate a new scenario to test this new network. For instance, we move to the fifth minute and set evidence that your bag has not yet arrived.

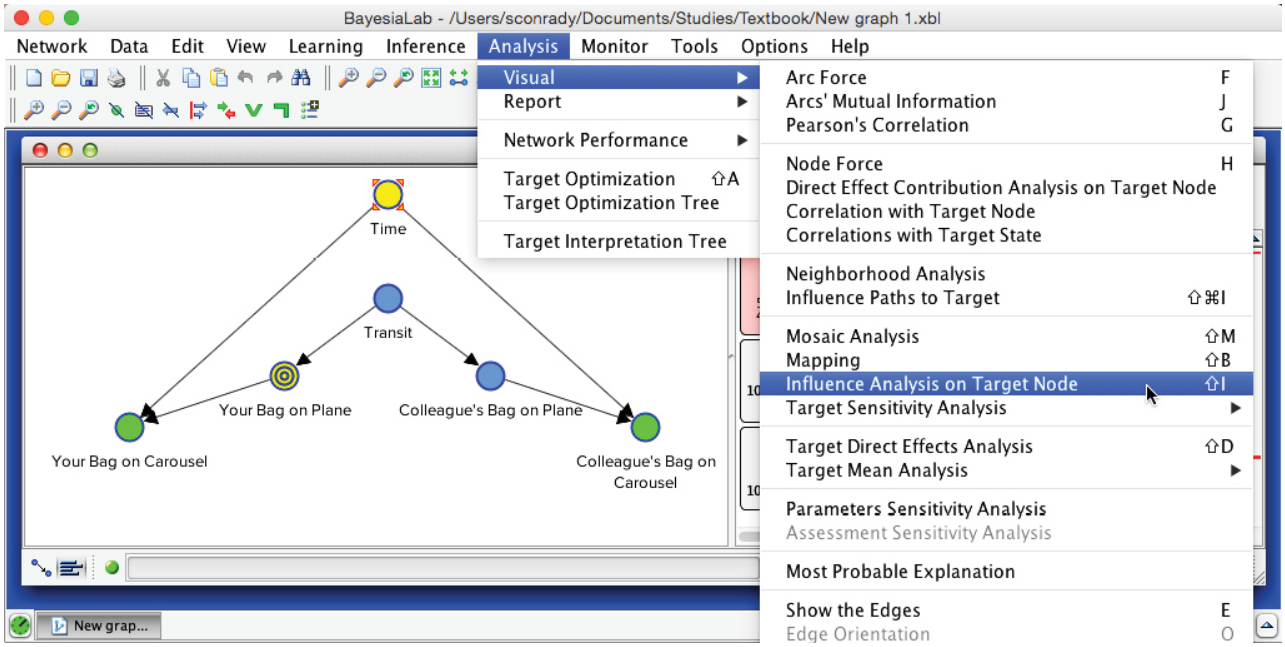


Given these observations, the probability of *Your Bag on Plane* = TRUE is now 33.33%. Interestingly, the probability of *Colleague's Bag on Plane* has also changed. As evidence propagates omnidirectionally through the network, our two observed nodes do indeed influence *Colleague's Bag on Plane*. A further iteration of the scenario in our story is that we observe *Colleague's Bag on Carousel* = TRUE, also in the fifth minute.

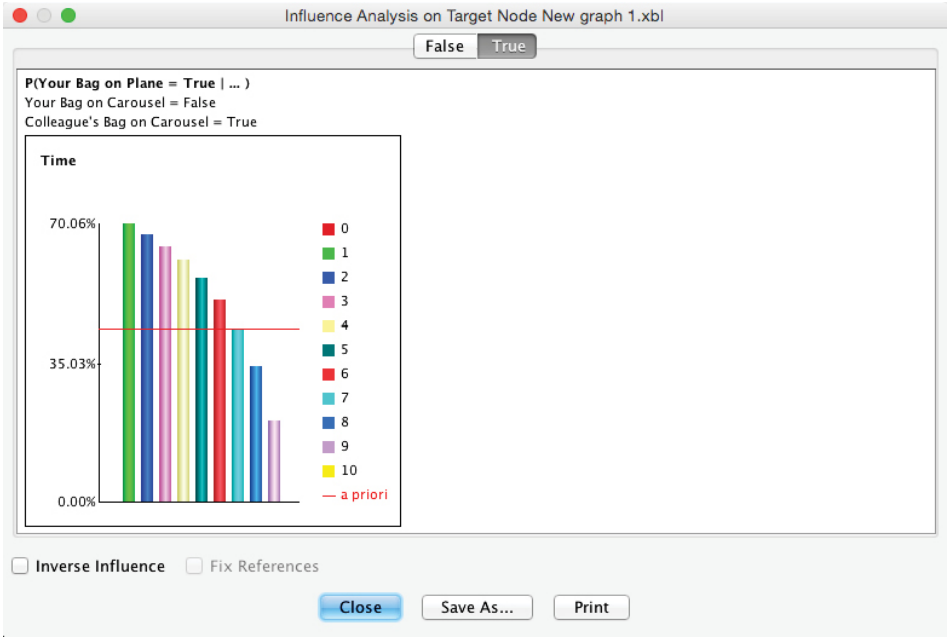


Given the observation of *Colleague's Bag on Carousel*, even though we have not yet seen *Your Bag on Carousel*, the probability of *Your Bag on Plane* increases to 56.52%. Indeed, this observation should change your expectation quite a bit. The small gray arrows on the blue bars inside the Monitor for *Your Bag on Plane* indicate the impact of this observation.

After removing the evidence from the *Time* Monitor, we can perform Influence Analysis on Target again in order to see the probability of *Your Bag on Plane* = TRUE as a function of *Time*, given *Your Bag on Carousel* = FALSE and *Colleague's Bag on Carousel* = TRUE. To focus our analysis on *Time* alone, we select the *Time* node and then select `Menu > Analysis > Visual > Influence Analysis on Target`.



As before, we select the TRUE tab in the resulting window to see the evolution of probabilities given *Time*.



Summary

This chapter provided a brief introduction to knowledge modeling and evidential reasoning with Bayesian networks in BayesiaLab. Bayesian networks can formally encode available knowledge, deal

with uncertainties, and perform omnidirectional inference. As a result, we can properly reason about a problem domain despite many unknowns.

Chapter 5: Bayesian Networks and Data

In the previous chapter, we described the application of Bayesian networks for evidential reasoning. All available knowledge was manually encoded in the Bayesian network in that example. In this chapter, we additionally use data for defining Bayesian networks. This provides the basis for the following chapters, which will present applications that utilize machine-learning for generating Bayesian networks entirely from data.

For machine learning with BayesiaLab, concepts derived from information theory, such as entropy and mutual information, are particularly important and should be understood by the researcher. However, these measures are not nearly as familiar to most scientists as common statistical measures, e.g., covariance and correlation.

Example: House Prices in Ames, Iowa

We present a straightforward research task to introduce these presumably unfamiliar information-theoretic concepts. The objective is to establish the predictive importance of a range of variables concerning a target variable. The domain of this example is residential real estate, and we wish to examine the relationships between home characteristics and sales prices. In this context, it is natural to ask questions related to variable importance, such as, which is the most important predictive variable pertaining to home value? By attempting to answer this question, we can explain what entropy and mutual information mean in practice and how BayesiaLab computes these measures. In this process, we also demonstrate a number of BayesiaLab's data-handling functions.

The dataset for this chapter's exercise describes the sale of individual residential properties in Ames, Iowa, from 2006 to 2010. It contains a total of 2,930 observations and a large number of explanatory variables (23 nominal, 23 ordinal, 14 discrete, and 20 continuous). This dataset was first used by De Cock (2011) as an educational tool for statistics students. The objective of their study was the same as ours, i.e., modeling sale prices as a function of the property attributes.

To make this dataset more convenient for demonstration purposes, we reduced the total number of variables to 49. This pre-selection was fairly straightforward as numerous variables essentially do not apply to homes in Ames, e.g., variables relating to pool quality and pool size (there are practically no pools) or roof material (it is the same for virtually all homes).

The Workflow in Detail

- **Data Import and Discretization**
- **Node Names, Long Names, and Node Comments**
- **Uncertainty, Entropy, and Mutual Information**
- **Parameter Estimation**
- **Naive Bayes Network**

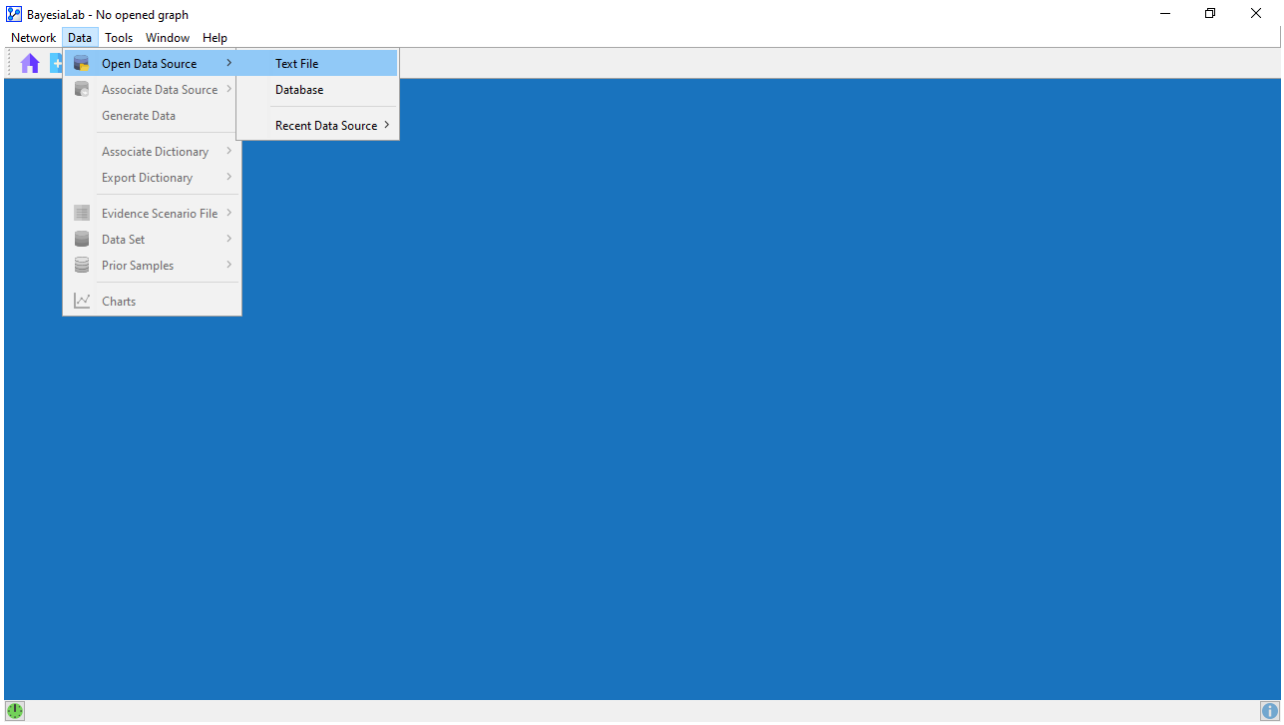
References

Dean De Cock. Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project. Journal of Statistical Education, 19(3), 2011.

Data Import and Discretization

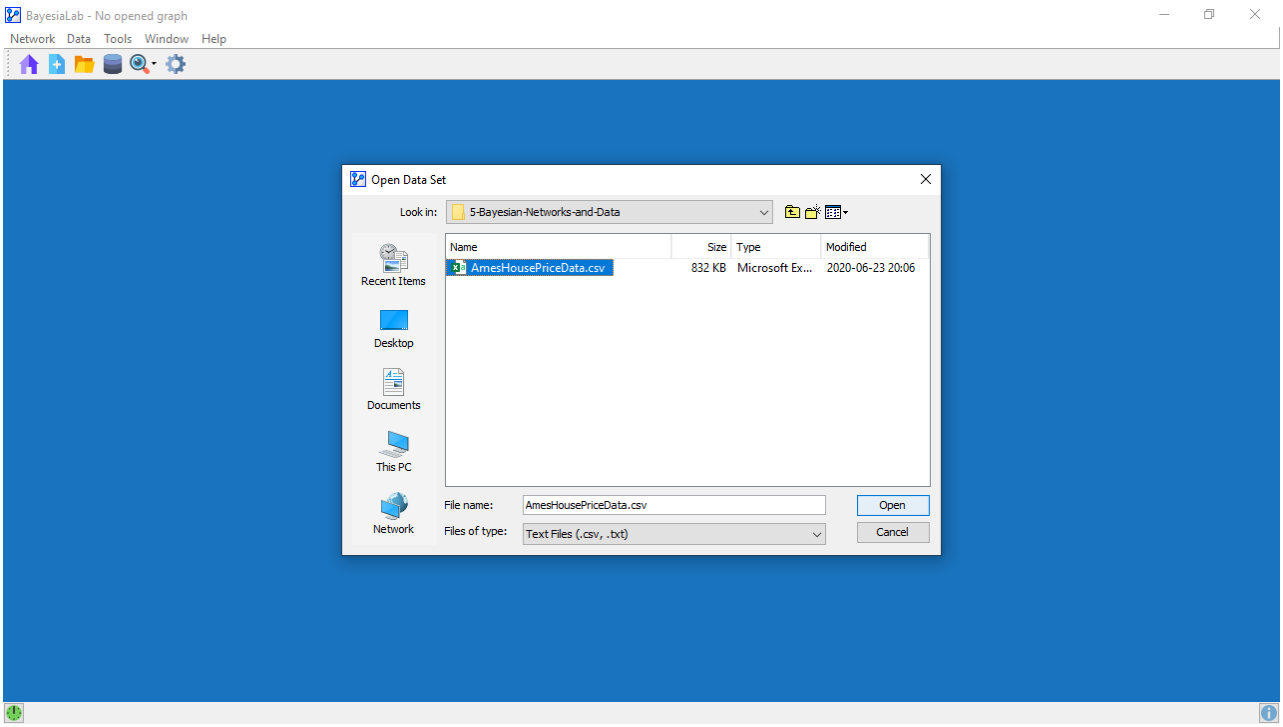
Open Data

As the first step, we start BayesiaLab's Data Import Wizard by selecting `Menu > Data > Open Data Source > Text File`.



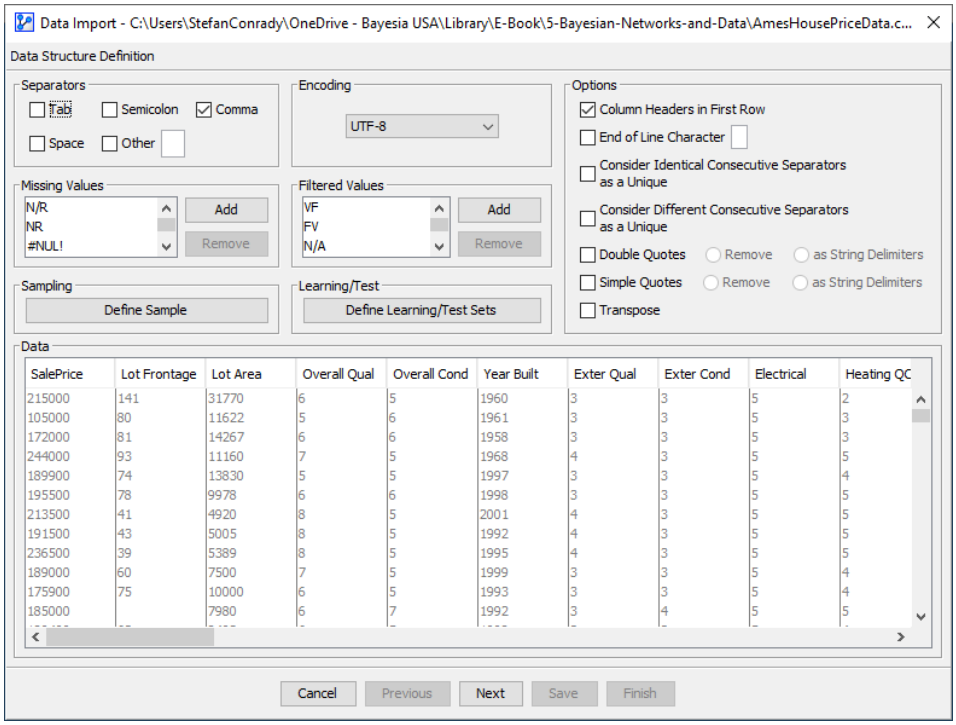
Then, we select the file `AmesHousePriceData.csv`, a comma-delimited, flat text file, which you can download here:

Download: `AmesHousePriceData.csv`

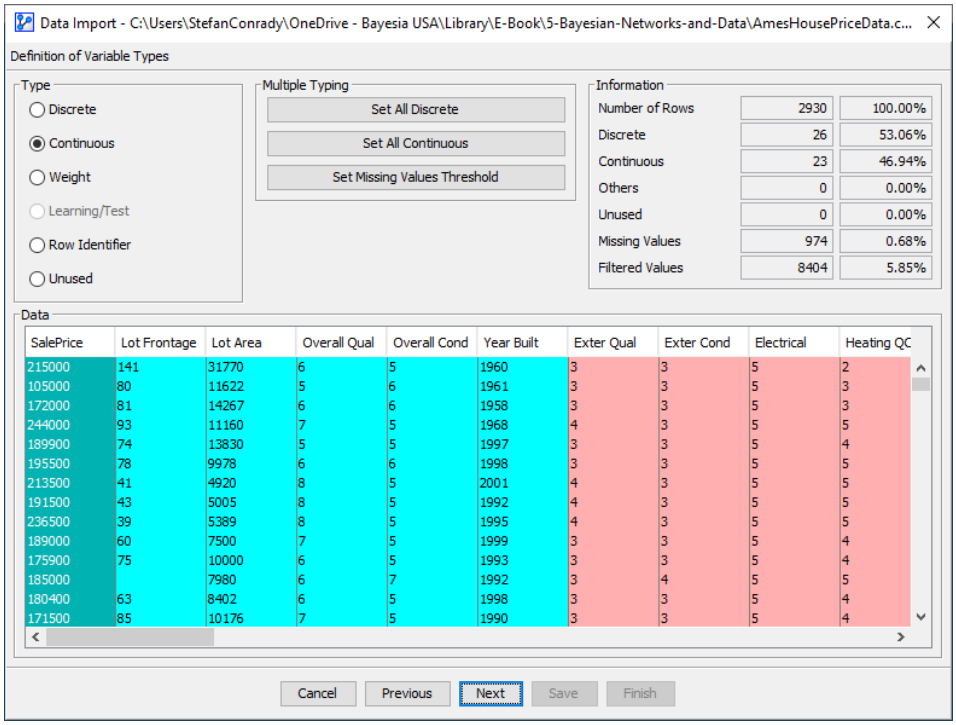


Data Import Wizard

This brings up the first screen of the Data Import Wizard, which previews the to-be-imported dataset. For this example, the coding options for Missing Values and Filtered Values are particularly important. By default, BayesiaLab lists commonly used codes that indicate an absence of data, e.g., #NUL! or NR (non-response). In the Ames dataset, a blank field (“ ”) indicates a Missing Value, and “FV” stands for Filtered Value. These are recognized automatically. If other codes were used, we could add them to the respective lists on this screen.



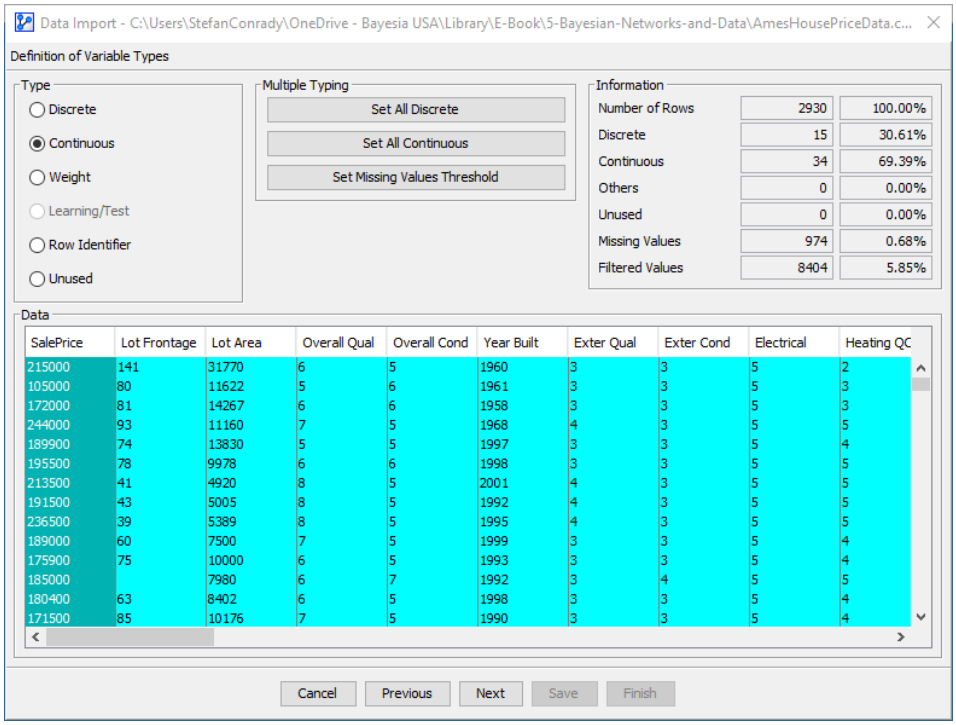
Clicking `Next`, we proceed to the screen that allows us to define variable types.



BayesiaLab scans all variables in the database and provides a best guess regarding the variable type. Variables identified as Continuous are shown in turquoise, and those identified as Discrete are highlighted in pastel red.

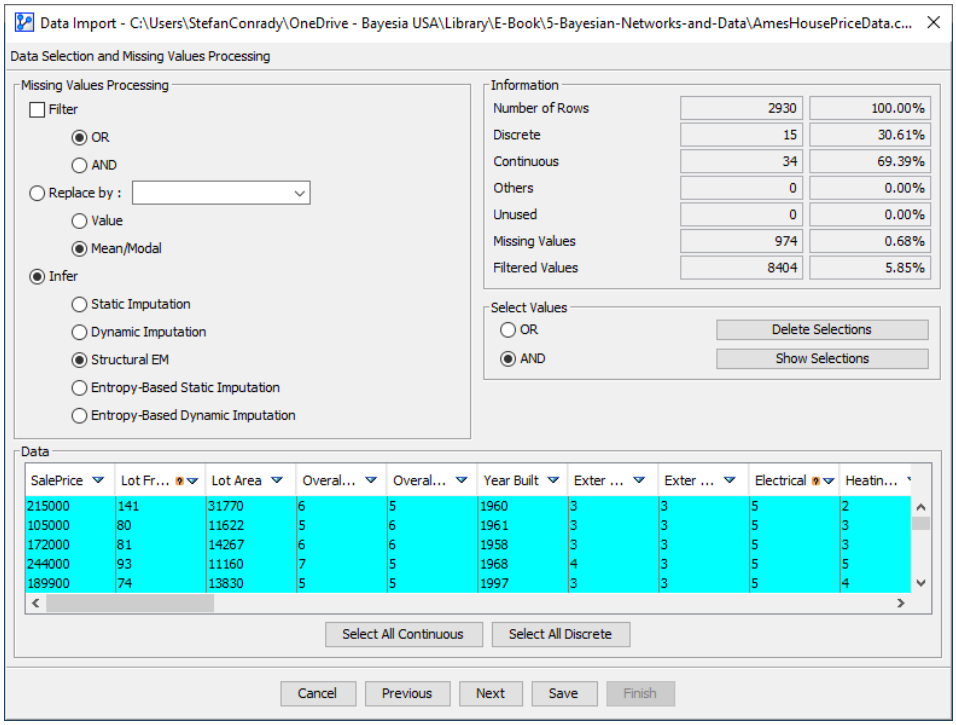
In BayesiaLab, a Continuous variable contains a wide range of numerical values (discrete or continuous), which need to be transformed into a more limited number of discrete states. Some other variables in the database only have very few distinct numerical values to begin with, e.g., [1,2,3,4,5], and BayesiaLab automatically recognizes such variables as Discrete. For them, the number of numerical states is small enough that creating bins of values is unnecessary. Also, variables containing text values are automatically considered Discrete.

For this dataset, however, we need to make a number of adjustments to the suggested data types. For instance, we set all numerical variables to Continuous Nodes, including those highlighted in red that were originally identified as Discrete Nodes. As a result, all columns in the data preview of the Data Import Wizard are now shown in turquoise.



Given that our database contains some missing values, we need to select the type of Missing Values Processing in the next step. Instead of using ad hoc methods, such as pairwise or listwise deletion, BayesiaLab can leverage more sophisticated techniques and provide estimates (or temporary placeholders) for such missing values–without discarding any original data.

We will discuss Missing Values Processing in detail in Chapter 9. For this example, however, we leave the default setting of Structural EM.



Filtered Values

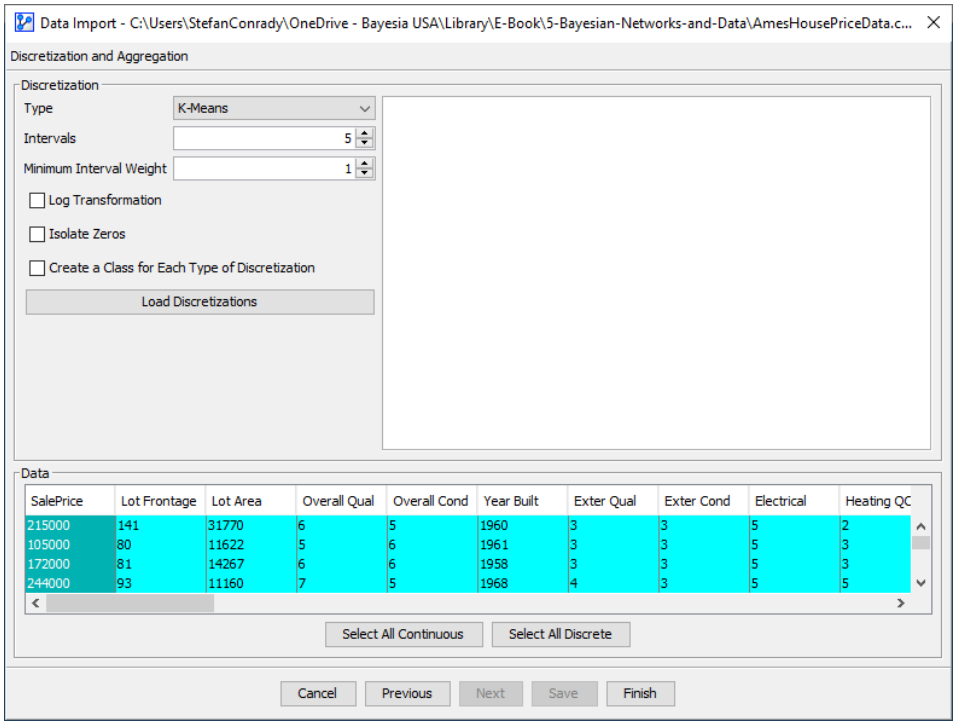
At this point, however, we must introduce a very special type of missing value for which we must not generate any estimates. We are referring to so-called Filtered Values. These are "impossible" values that do not or cannot exist–given a specific set of evidence, as opposed to values that do exist but are not observed. For example, for a home that does not have a garage, there cannot be any value for the variable *Garage Type*, such as Attached to Home, Detached from Home, or Basement Garage. If there is no garage, there cannot be a garage type. As a result, it makes no sense to calculate an estimate of a Filtered Value. In a database, unfortunately, a Filtered Value typically looks identical to a "true" missing value that does exist but is not observed. The database typically contains the same code, such as a blank, NULL, N/A, etc., for both cases.

Therefore, instead of "normal" missing values, which can be left as-is in the database, we must mark Filtered Values with a specific code, e.g., "FV." The Filtered Value declaration should be done during data preparation before importing any data into BayesiaLab. BayesiaLab will then add a Filtered State (marked with "*") to the discrete states of the variables with Filtered Values and utilize a special approach for actively disregarding such Filtered States so that they are not taken into account during machine learning or for estimating effects.

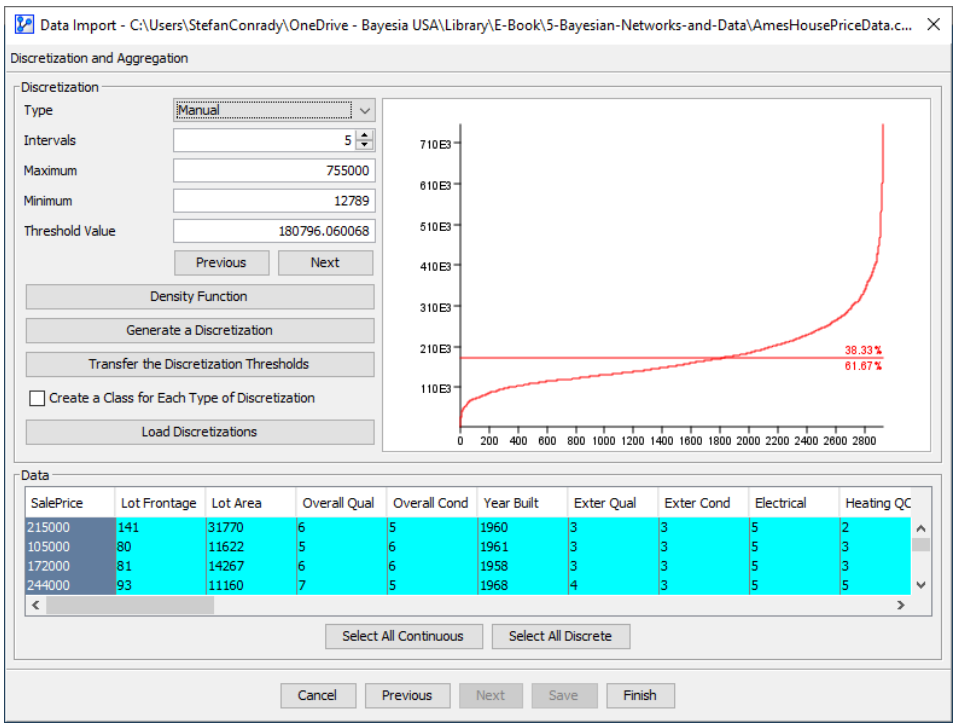
> See also Filtered Values in Chapter 9 — Missing Values Processing.

Discretization

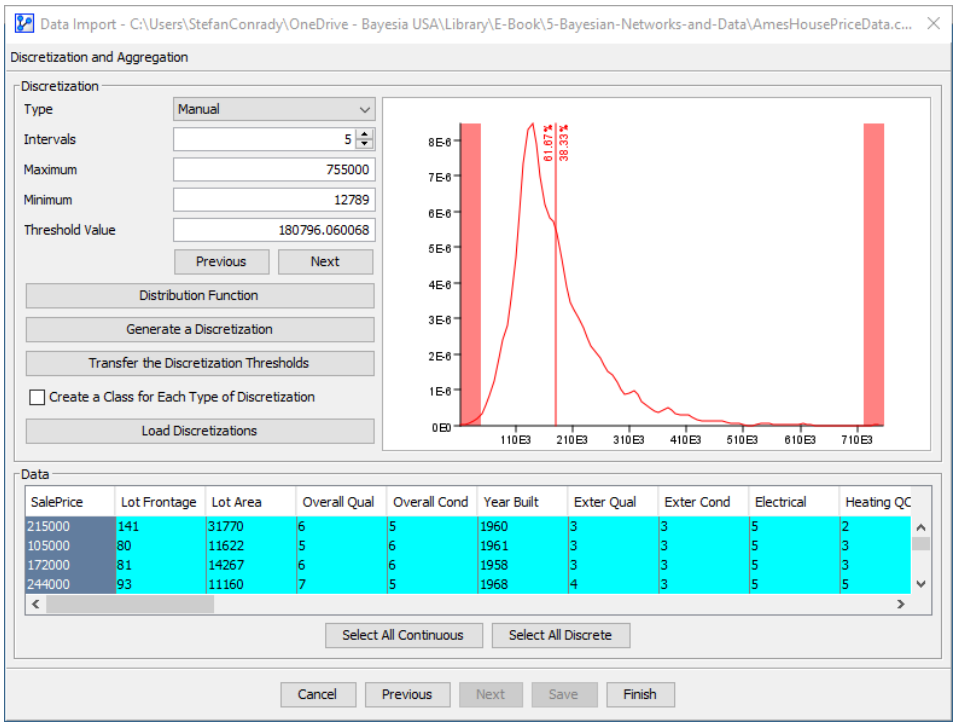
As the next step in the Data Import Wizard, all Continuous values must be discretized (or binned). We show a sequence of screenshots to highlight the necessary steps. The initial view of the Discretization and Aggregation step appears.



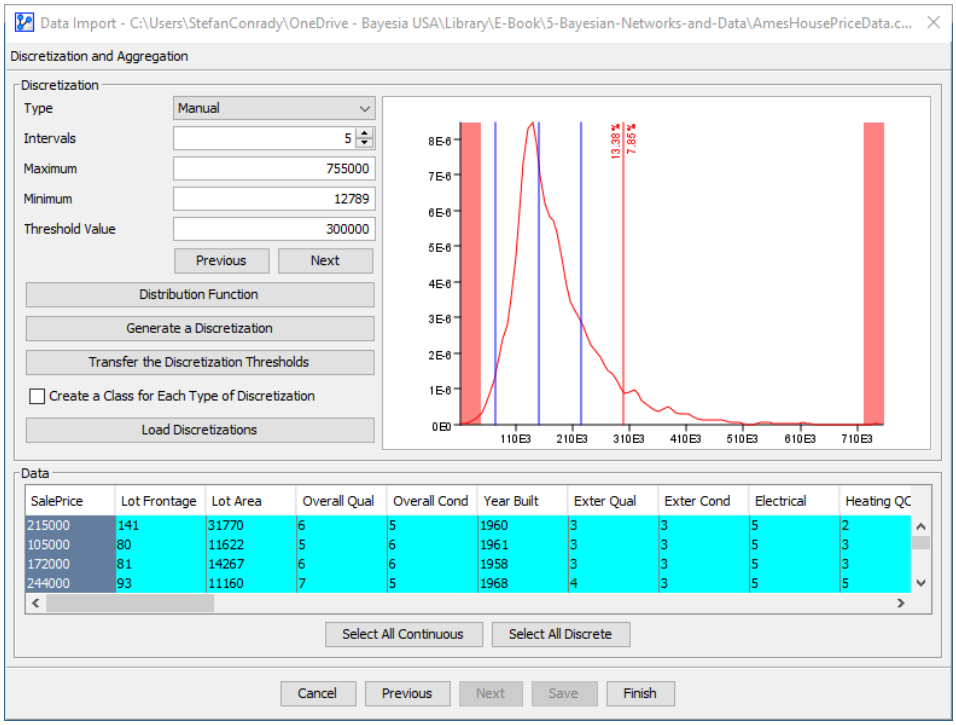
By default, the first column is highlighted, which happens to be *SalePrice*, the variable of principal interest in this example. Instead of selecting any available automatic discretization algorithms, we pick `Manual` from the `Type` drop-down menu, which brings up the Cumulative Distribution Function (CDF) of the *SalePrice* variable.



By clicking `Density Function`, we can bring up the Probability Density Function (PDF) of *SalePrice*.



Either view allows us to examine the distribution and identify any salient points. We stay on the current screen to set the thresholds for each discretization bin. In many instances, we would use an algorithm to define bins automatically unless the variable will serve as the target variable. In that case, we usually rely on available expert knowledge to define the binning. In this example, we wish to have evenly-spaced, round numbers for the interval boundaries. We add boundaries by right-clicking on the plot (right-clicking on an existing boundary removes it again). Furthermore, we can fine-tune a threshold's position by entering a precise value in the `Threshold Value` field. We use {75000, 150000, 225000, 300000} as the interval boundaries.



Tree Discretization

Now that we have manually discretized the target variable *SalePrice* (column highlighted), we still need to discretize the remaining continuous variables. However, we will take advantage of an automatic discretization algorithm for those variables.

We click `Select All Continuous`. BayesiaLab automatically excludes *SalePrice* from this selection because we have already discretized it.

Numerous automatic discretization algorithms are available, but for the purpose of this example, we only consider the bivariate Tree discretization algorithm.

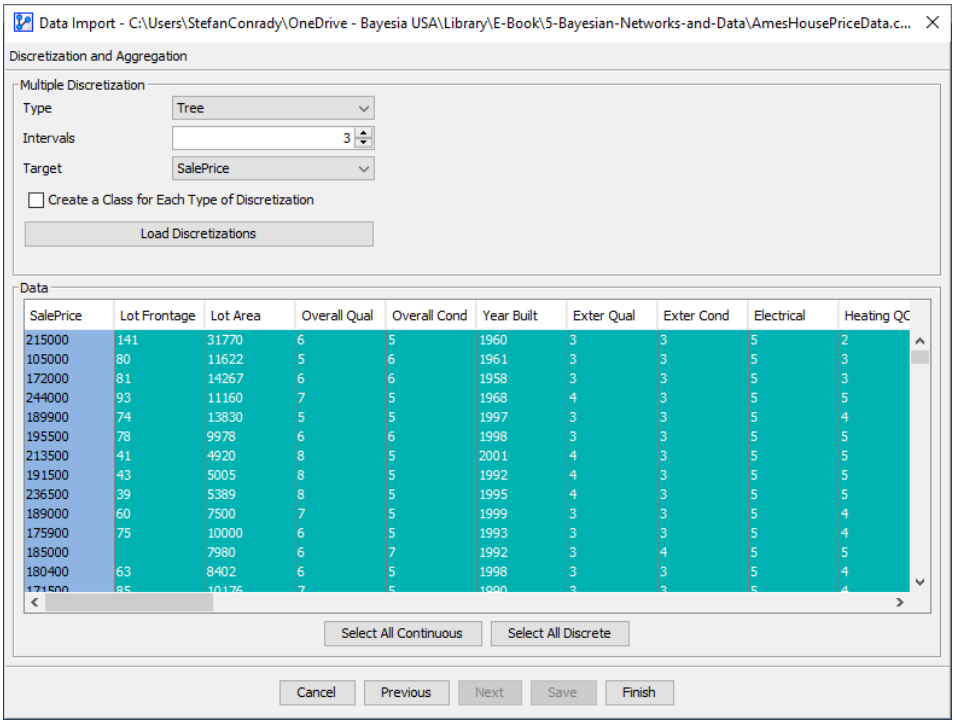
> Please see the main entry for Discretization in this library for a detailed description of all available algorithms.

As its name suggests, the Tree discretization algorithm machine-learns a decision tree that uses the to-be-discretized variable for representing the conditional probability distributions of the target variable given the to-be-discretized variable. Once the Tree is learned, it is analyzed to extract the most useful thresholds. This is the method of choice in the context of Supervised Learning, i.e., when planning to machine-learn a model to predict the target variable.

At the same time, we do not recommend using Tree in the context of Unsupervised Learning. The Tree algorithm creates bins that are biased toward the designated target variable. Naturally, emphasizing one particular variable would run counter to the intent of Unsupervised Learning.

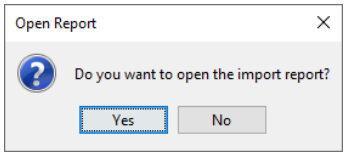
Note that if the to-be-discretized variable is independent of the target variable, it will be impossible to build a tree, and BayesiaLab will prompt the selection of a univariate discretization algorithm.

In this example, we focus our analysis on *SalePrice*, which can be considered a type of Supervised Learning. Therefore, we discretize all continuous variables with the Tree algorithm, using *SalePrice* as the Target variable. Note the Target must either be a Discrete variable or a Continuous variable that has already been manually discretized, which is the case for *SalePrice*.

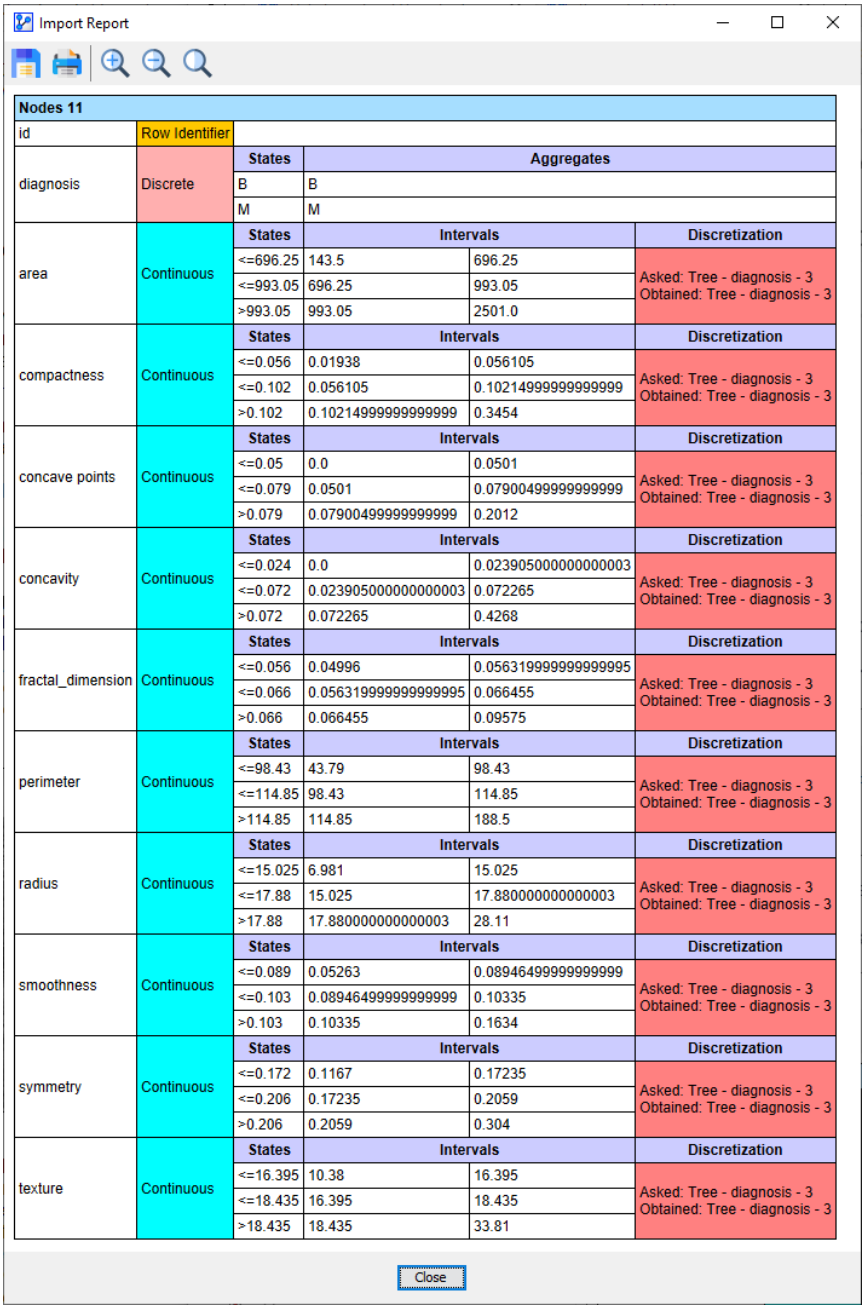


Clicking `Finish` completes the import process.

The import process concludes with a pop-up window that offers to display the Import Report.



Clicking `Yes` brings up the Import Report, which can be saved in HTML format. It lists the discretization intervals of the Continuous variables, the States of the Discrete variables, and the discretization method used for each variable.



Import Report

Nodes 11

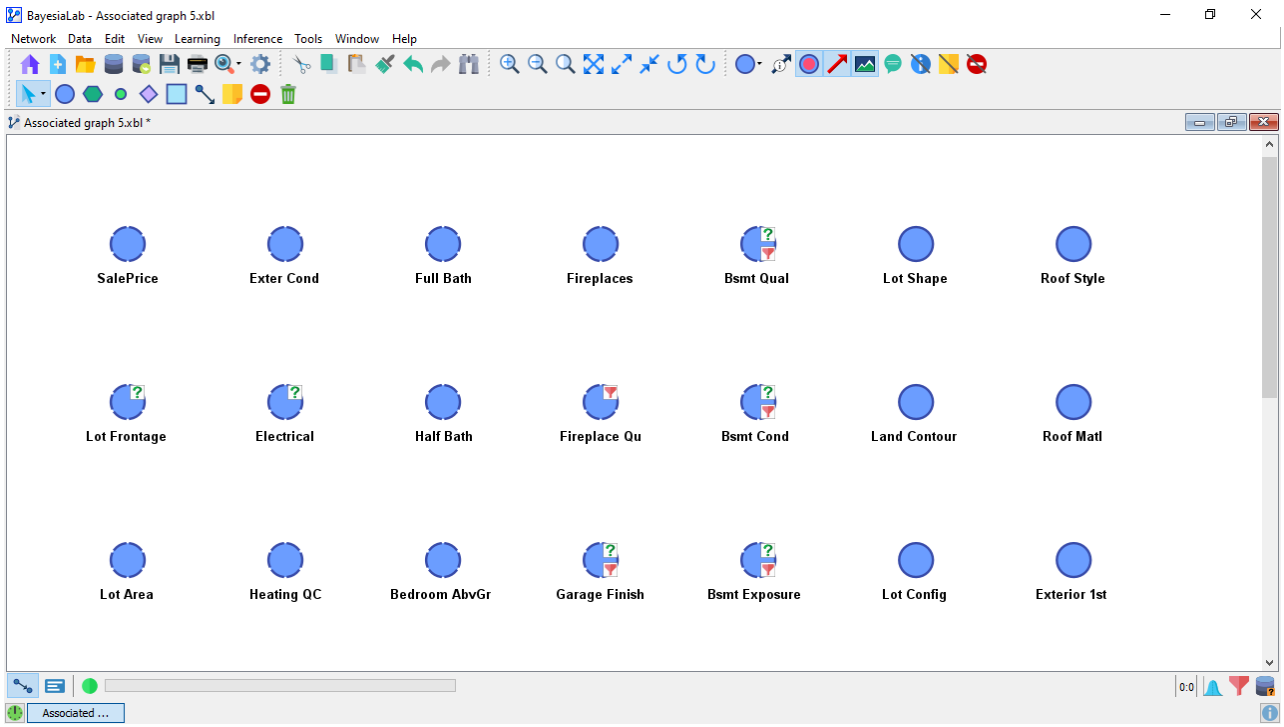
id	Row Identifier				
diagnosis	Discrete	States	Aggregates		
		B	B		
		M	M		
area	Continuous	States	Intervals		Discretization
		<=696.25	143.5	696.25	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=993.05	696.25	993.05	
		>993.05	993.05	2501.0	
compactness	Continuous	States	Intervals		Discretization
		<=0.056	0.01938	0.056105	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.102	0.056105	0.10214999999999999	
		>0.102	0.10214999999999999	0.3454	
concave points	Continuous	States	Intervals		Discretization
		<=0.05	0.0	0.0501	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.079	0.0501	0.07900499999999999	
		>0.079	0.07900499999999999	0.2012	
concavity	Continuous	States	Intervals		Discretization
		<=0.024	0.0	0.023905000000000003	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.072	0.023905000000000003	0.072265	
		>0.072	0.072265	0.4268	
fractal_dimension	Continuous	States	Intervals		Discretization
		<=0.056	0.04996	0.056319999999999995	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.066	0.056319999999999995	0.066455	
		>0.066	0.066455	0.09575	
perimeter	Continuous	States	Intervals		Discretization
		<=98.43	43.79	98.43	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=114.85	98.43	114.85	
		>114.85	114.85	188.5	
radius	Continuous	States	Intervals		Discretization
		<=15.025	6.981	15.025	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=17.88	15.025	17.880000000000003	
		>17.88	17.880000000000003	28.11	
smoothness	Continuous	States	Intervals		Discretization
		<=0.089	0.05263	0.08946499999999999	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.103	0.08946499999999999	0.10335	
		>0.103	0.10335	0.1634	
symmetry	Continuous	States	Intervals		Discretization
		<=0.172	0.1167	0.17235	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.206	0.17235	0.2059	
		>0.206	0.2059	0.304	
texture	Continuous	States	Intervals		Discretization
		<=16.395	10.38	16.395	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=18.435	16.395	18.435	
		>18.435	18.435	33.81	

Close

Graph Panel

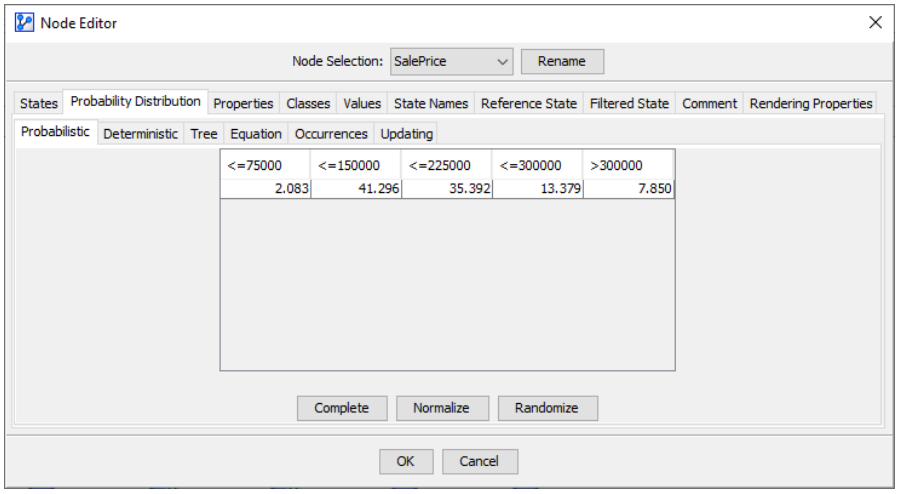
Once we close out this report, we can see the result of the import process. All the imported variables are now represented as nodes on the Graph Panel. The dashed borders of some nodes indicate that the corresponding variables were discretized during data import.

Furthermore, we can see icons that indicate the presence of Missing Values ? and Filtered Values ▽ in the respective nodes.

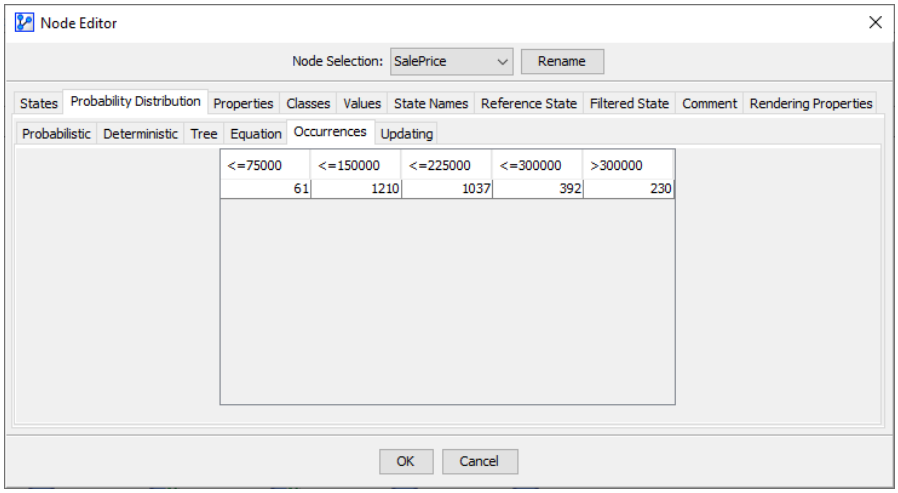


The lack of warning icons on any nodes indicates that all their parameters, i.e., their marginal probability distributions, were automatically estimated upon data import.

To verify, we open the Node Editor of *SalePrice* (`Node Context Menu > Edit > Probability Distribution > Probabilistic`) and check the node's marginal distribution.

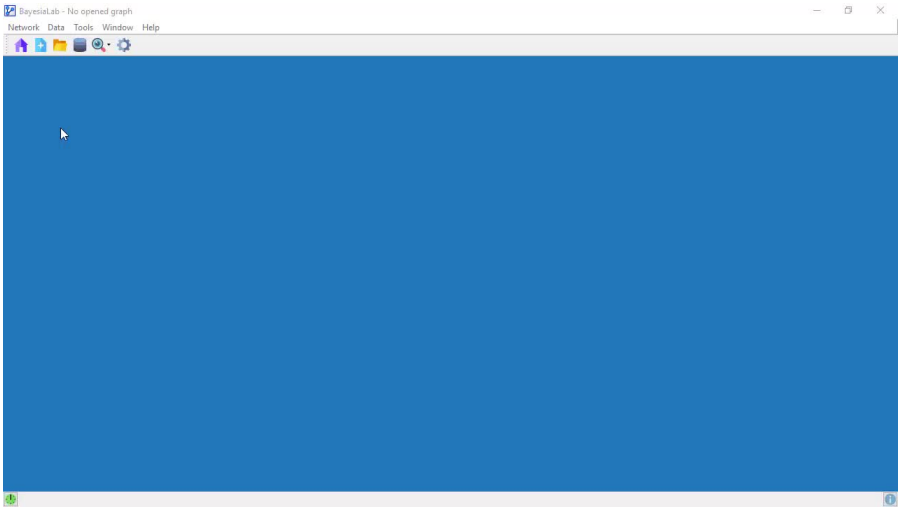


Clicking on the Occurrences tab shows the observations per cell, which were used for the Maximum Likelihood Estimation of the marginal distribution.



Workflow Animation

The following animation shows all the above steps in a continuous workflow.

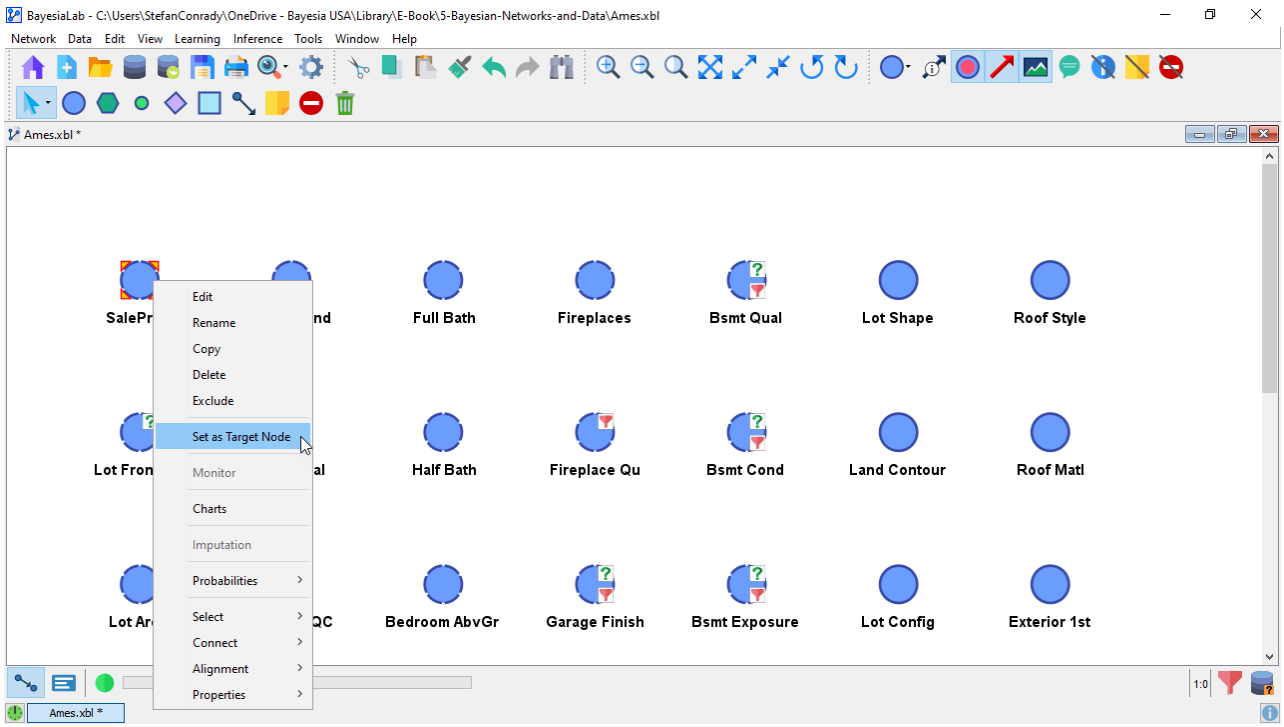


Click to view video

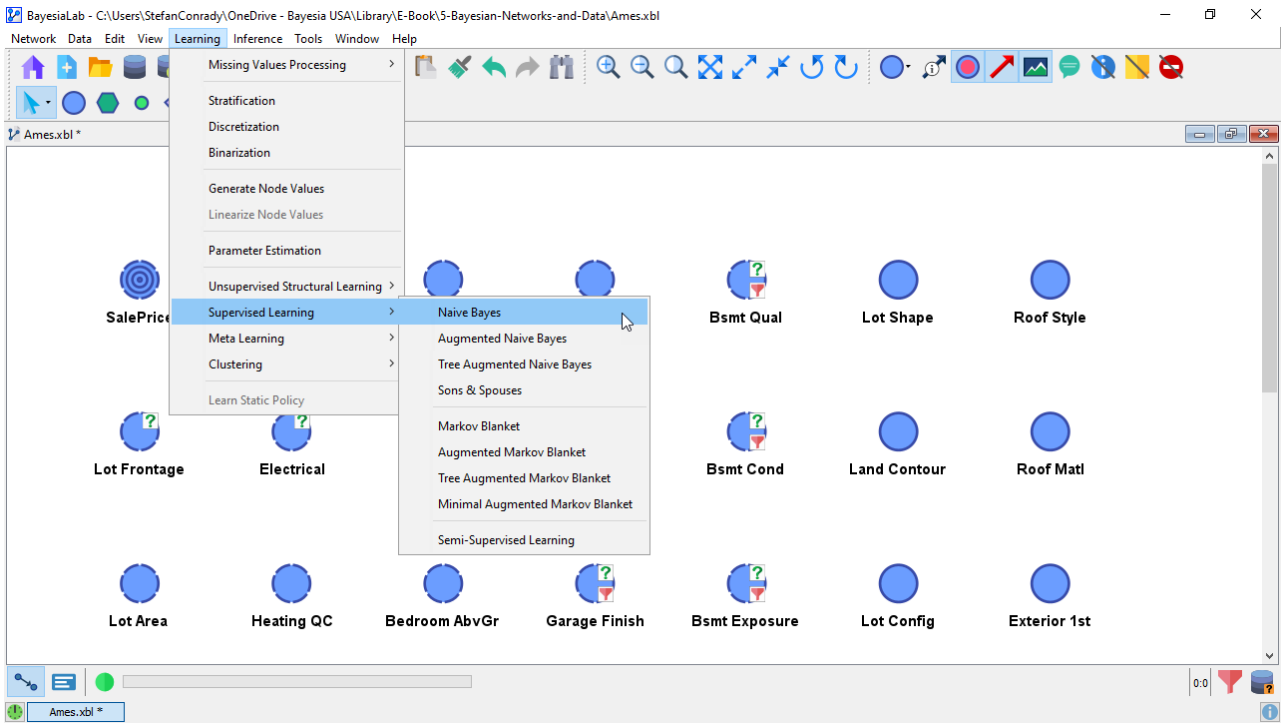
Naive Bayes Network

Rather than computing the relationships individually for each pair of nodes, we ask BayesiaLab to estimate a Naive Bayes network. A Naive Bayes structure is a network with only one parent, the Target Node, i.e., the only arcs in the graph are those directly connecting the **Target Node** to a set of nodes. By designating *SalePrice* as the Target Node, we can automatically compute its Mutual Information with all other available nodes.

For the node *SalePrice*, we select `Node Context Menu > Set as Target Node`. Alternatively, we can double-click the node while pressing `T`.



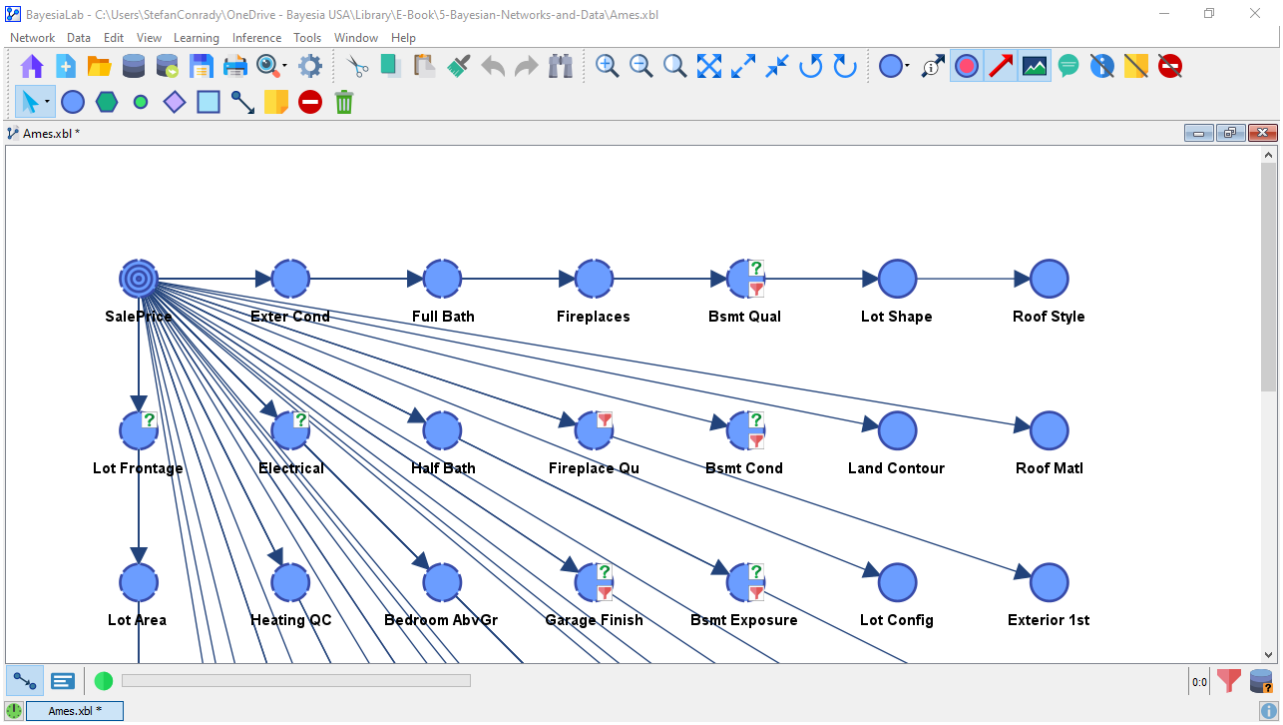
The special status of the Target Node is highlighted by the bullseye symbol . We can now proceed to learn the Naive Bayes network: `Select Menu > Learning > Supervised Learning > Naive Bayes`.



Strictly speaking, we are not learning a network in the true sense of machine learning. Rather, we are specifying a naive structure, i.e., arcs from the Target Node to all other nodes, and then estimating the parameters.

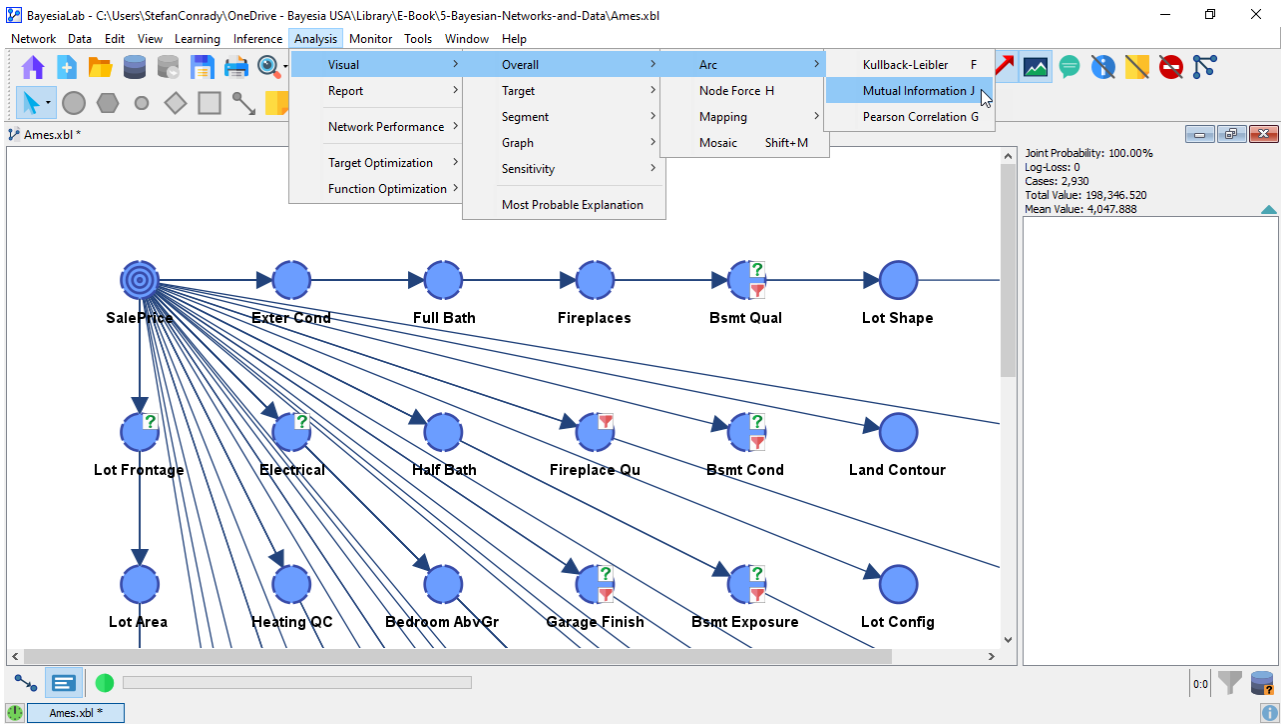
Due to its simplicity, the Naive Bayes network is presumably the most commonly used Bayesian network. As a result, we find it implemented in many software packages. For instance, the so-called Bayesian anti-spam systems are based on this model.

However, it is important to note that the Naive Bayes network is merely the first step towards embracing the Bayesian network paradigm.

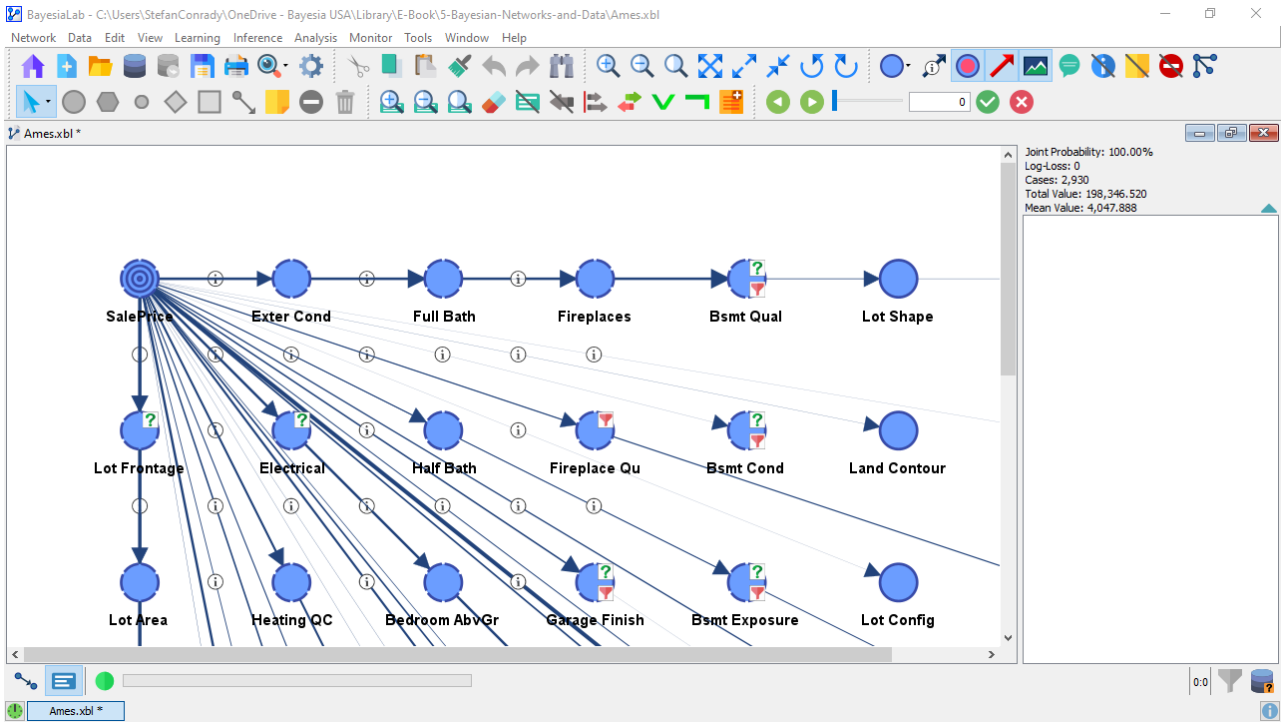


Now we have the network that allows computing the Mutual Information between all nodes.

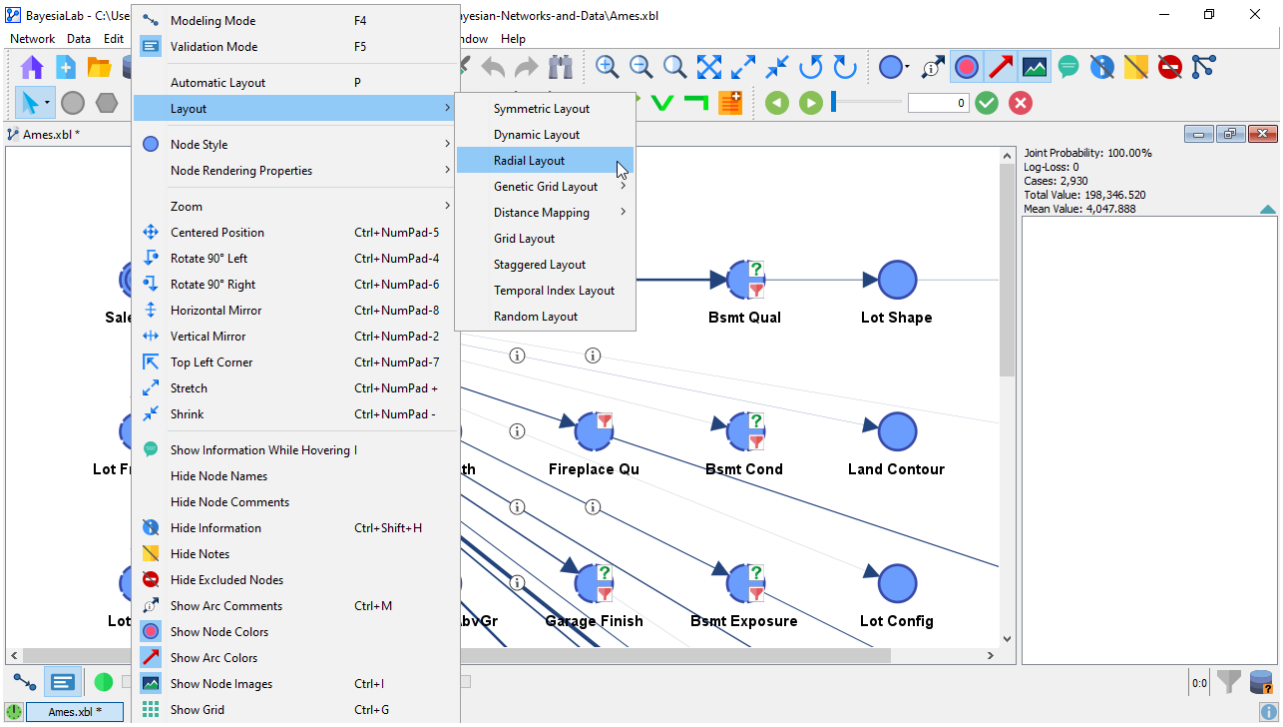
We switch to Validation Mode F5 and select `Menu > Analysis > Visual > Overall > Arc > Mutual Information`.



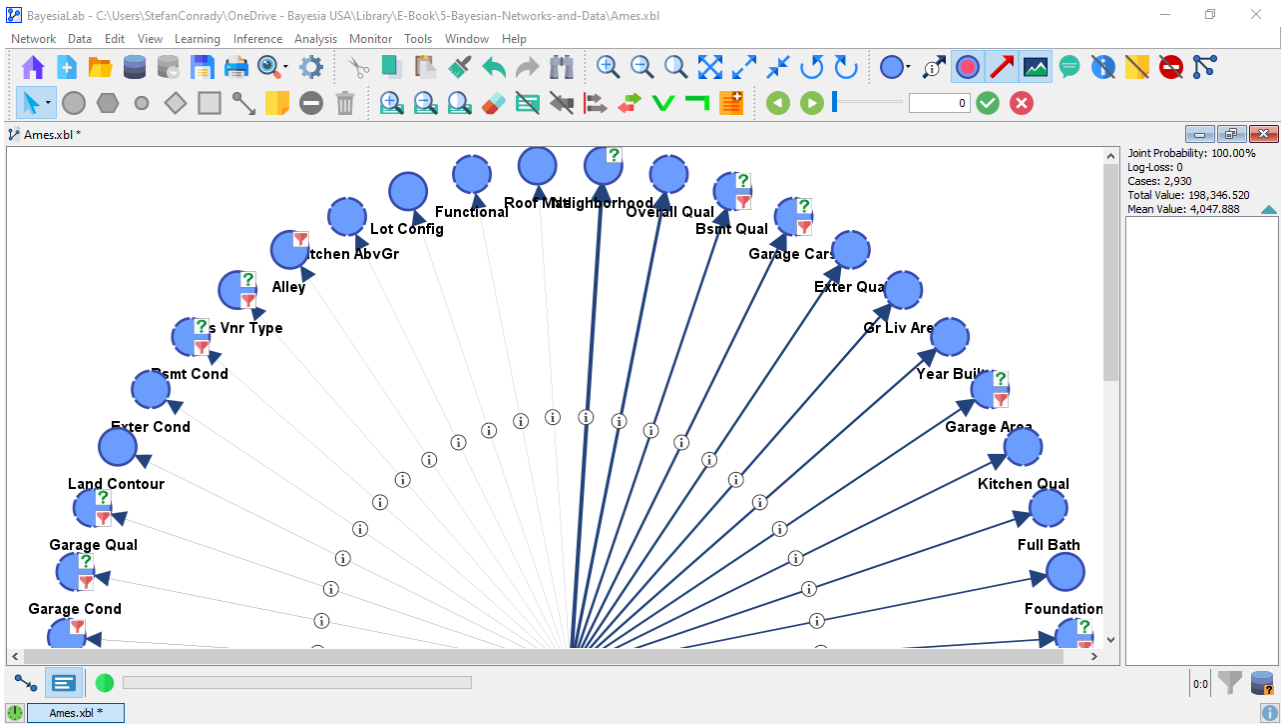
The different levels of Mutual Information are now reflected in the thickness of the arcs.



However, given the grid layout of the nodes and the overlapping arcs, it is difficult to establish a rank order of the nodes in terms of Mutual Information. To address this, we adjust the layout and select `Menu > View > Layout > Radial Layout`.



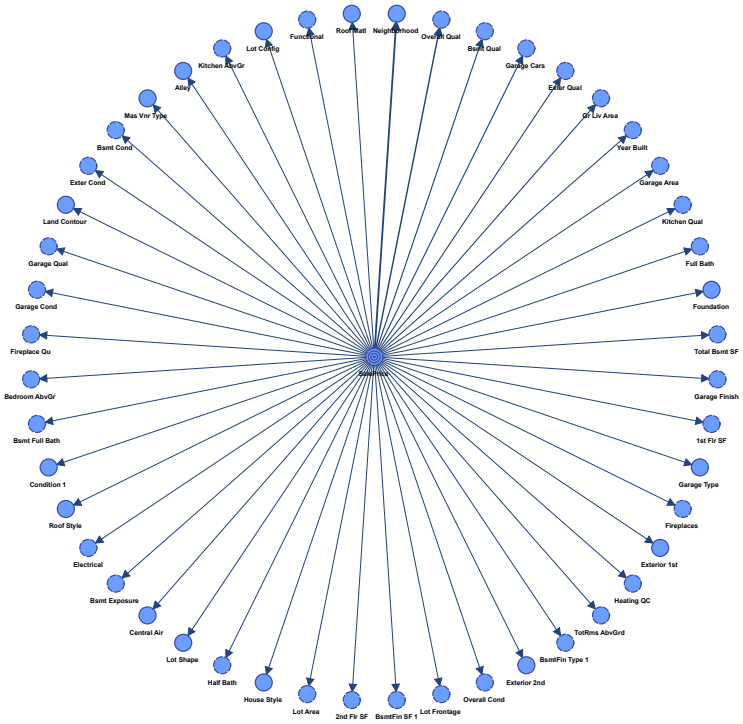
This generates a circular arrangement of all nodes with the Target Node, *SalePrice*, in the center. Clicking the Stretch icon ⤢ repeatedly, we expand the network to make it fit into the available screen space widthwise.



Also, having run the `Radial Layout` while the `Arc Mutual Information` function was still active, the arcs and nodes are ordered clockwise from strongest to weakest Mutual Information.

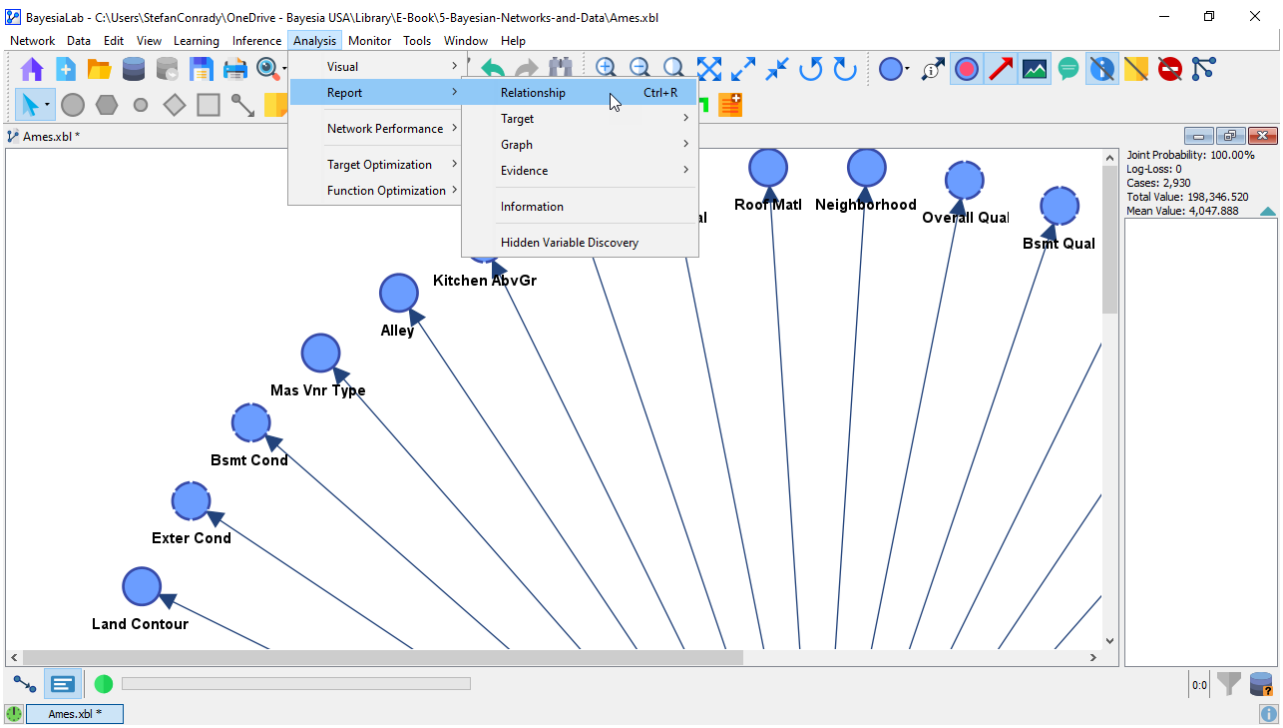
To improve the interpretability further, we select `Menu > View > Hide Information`. Alternatively, we click the Hide Information icon in the Toolbar. This removes the information icons ⓘ from the arcs. Their presence indicates that further information would be available to display, e.g., the numerical values of the Mutual Information of each arc.

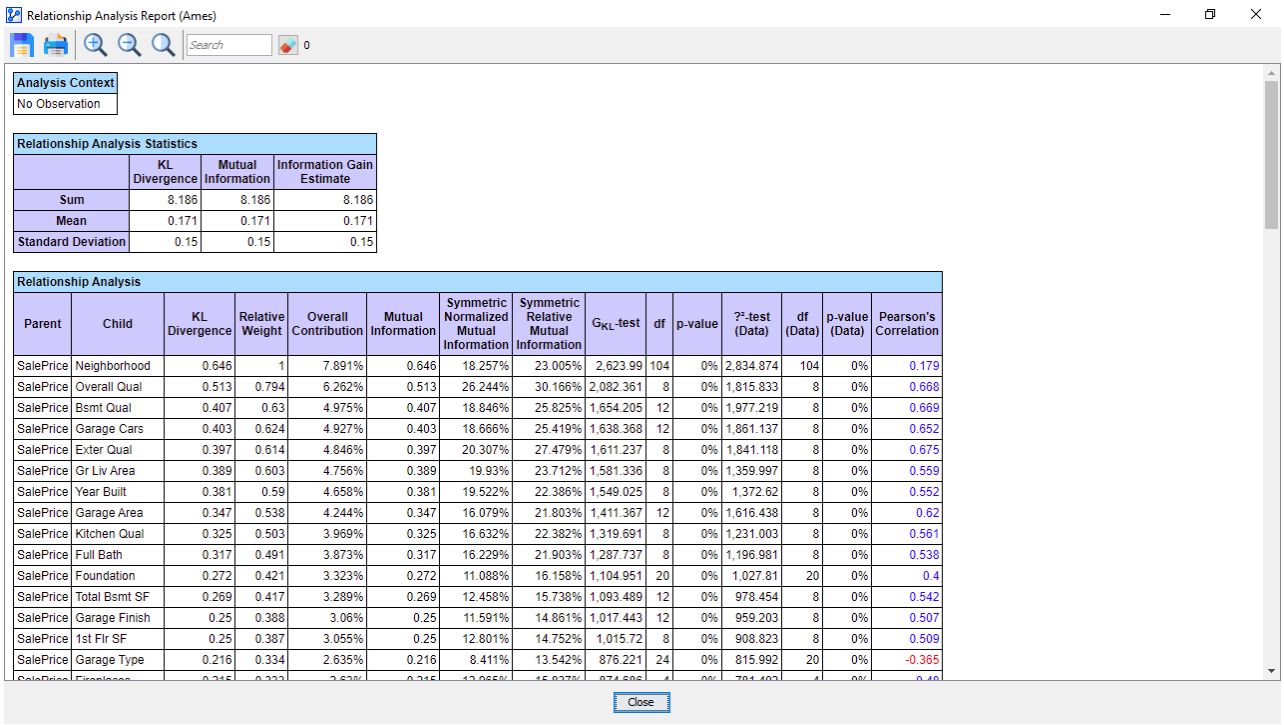
The following standalone graphic highlights the order of the arcs and nodes in this Naive Bayes network:



This illustration shows that *Neighborhood* provides the highest amount of Mutual Information and, at the opposite end of the range, RoofMtl (Roof Material) the least.

As an alternative to this visualization, we can run a report `Main Menu > Analysis > Report > Relationship`:




Relationship Analysis Report (Ames)

Search 0

Analysis Context
No Observation

Relationship Analysis Statistics	KL Divergence	Mutual Information	Information Gain Estimate
Sum	8.186	8.186	8.186
Mean	0.171	0.171	0.171
Standard Deviation	0.15	0.15	0.15

Relationship Analysis

Parent	Child	KL Divergence	Relative Weight	Overall Contribution	Mutual Information	Symmetric Normalized Mutual Information	Symmetric Relative Mutual Information	G_{KL}-test	df	p-value	?²-test (Data)	df (Data)	p-value (Data)	Pearson's Correlation
SalePrice	Neighborhood	0.646	1	7.891%	0.646	18.257%	23.005%	2,623.99	104	0%	2,834.874	104	0%	0.179
SalePrice	Overall Qual	0.513	0.794	6.262%	0.513	26.244%	30.166%	2,082.361	8	0%	1,815.833	8	0%	0.668
SalePrice	Bsmt Qual	0.407	0.63	4.975%	0.407	18.846%	25.825%	1,654.205	12	0%	1,977.219	8	0%	0.669
SalePrice	Garage Cars	0.403	0.624	4.927%	0.403	18.666%	25.419%	1,638.368	12	0%	1,861.137	8	0%	0.652
SalePrice	Exter Qual	0.397	0.614	4.846%	0.397	20.307%	27.479%	1,611.237	8	0%	1,841.118	8	0%	0.675
SalePrice	Gr Liv Area	0.389	0.603	4.756%	0.389	19.93%	23.712%	1,581.336	8	0%	1,359.997	8	0%	0.559
SalePrice	Year Built	0.381	0.59	4.658%	0.381	19.522%	22.386%	1,549.025	8	0%	1,372.62	8	0%	0.552
SalePrice	Garage Area	0.347	0.538	4.244%	0.347	16.079%	21.803%	1,411.367	12	0%	1,616.438	8	0%	0.62
SalePrice	Kitchen Qual	0.325	0.503	3.969%	0.325	16.632%	22.382%	1,319.691	8	0%	1,231.003	8	0%	0.561
SalePrice	Full Bath	0.317	0.491	3.873%	0.317	16.229%	21.903%	1,287.737	8	0%	1,196.981	8	0%	0.538
SalePrice	Foundation	0.272	0.421	3.323%	0.272	11.088%	16.158%	1,104.951	20	0%	1,027.81	20	0%	0.4
SalePrice	Total Bsmt SF	0.269	0.417	3.289%	0.269	12.458%	15.738%	1,093.489	12	0%	978.454	8	0%	0.542
SalePrice	Garage Finish	0.25	0.388	3.06%	0.25	11.591%	14.861%	1,017.443	12	0%	959.203	8	0%	0.507
SalePrice	1st Flr SF	0.25	0.387	3.055%	0.25	12.801%	14.752%	1,015.72	8	0%	908.823	8	0%	0.509
SalePrice	Garage Type	0.216	0.334	2.635%	0.216	8.411%	13.542%	876.221	24	0%	815.992	20	0%	-0.365

Close

Wouldn't this report look the same if computed based on correlation? In fact, the rightmost column in this `Relationship Analysis Report` shows Pearson's Correlation for reference. As we can see, the order would be different if we chose Pearson's Correlation as the main metric.

So, what have we gained over correlation? One of the key advantages of Mutual Information is that it can be computed–and interpreted–between numerical and categorical variables without any variable transformation. For instance, we can easily compute the Mutual Information, such as between the *Neighborhood* and *SalePrice*. The question regarding the most important predictive variable can now be answered. It is *Neighborhood*.

Now that we have established the central role of Entropy and Mutual Information, we can apply these concepts in the next chapters for machine learning and network analysis.

Node Names, Long Names, and Node Comments

Node Names, Long Names, and Node Comments

The Node Names displayed by default are taken directly from the column header of the imported dataset. To keep the Graph Panel uncluttered, we will keep these "short" names as the formal Node Names. On the other hand, we may want to have longer, more descriptive names available when interpreting the network or presenting it to an audience.

BayesiaLab offers three levels of "node names" for each node:

- The Node Name uniquely identifies a node and is displayed by default.

- A Long Name can be displayed instead of the Node Name on the Graph Panel, on the Monitors in the Monitor Panel, on reports, and in the context of many analysis functions.
- A Node Comment provides additional space for supplemental information about a node. For instance, if nodes represent survey responses, the Node Comment could accommodate the verbatim survey question.

Long Names

Long Names can be added to a network in two ways:

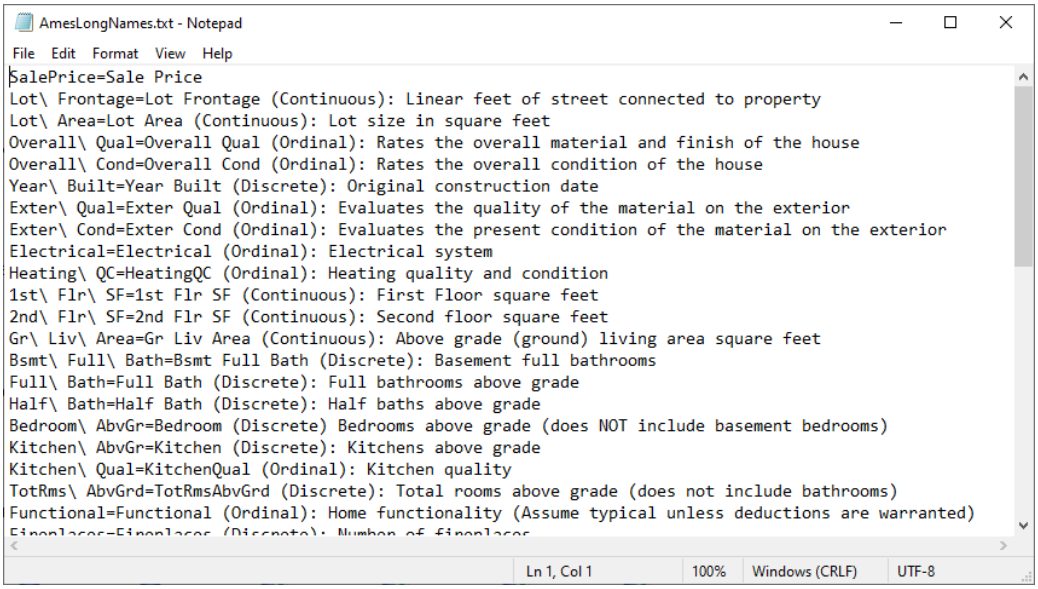
- One by one for each node via the `Properties` tab of the Node Editor (`Node Context Menu > Edit > Properties`).
- Using a Dictionary to provide Long Names for multiple nodes at once.

Dictionary

Given that we want to apply Long Names to 49 nodes, using a Dictionary will be much more convenient. The format of a Dictionary is rather straightforward:

- We define a plain text file that includes one Node Name per line. Spaces and special characters in the Node Name require backslash "\" as an escape character.
- Each Node Name is followed by a delimiter ("=", tab, or space) and then by the Long Name.

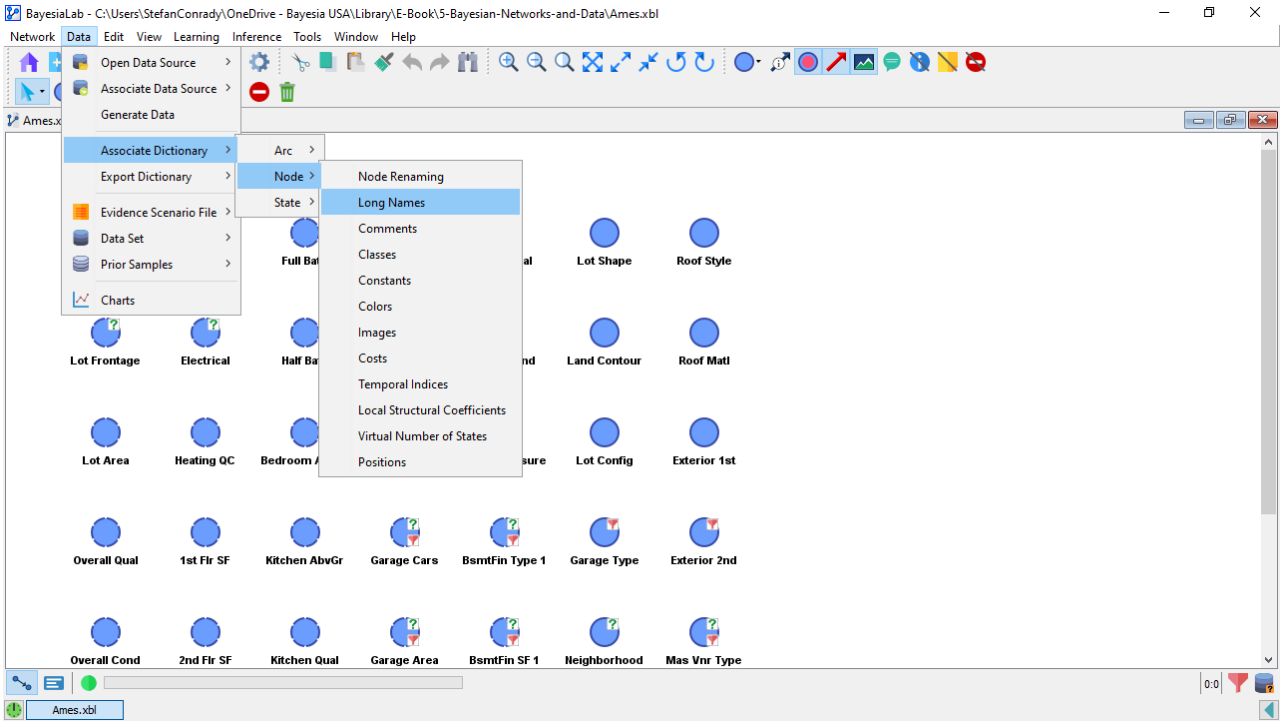
Here is a preview of the Dictionary:



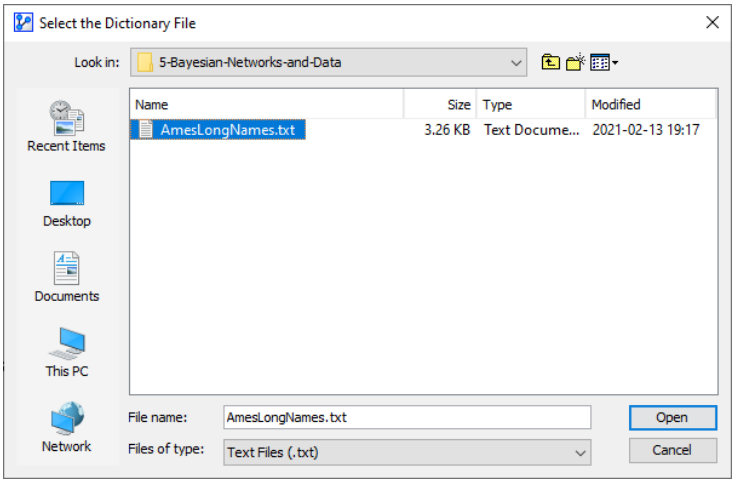
You can download the complete Dictionary file here:

Download: `AmesLongNames.txt`

To attach this Dictionary, select `Menu > Data > Associate Dictionary > Node > Long Names`.



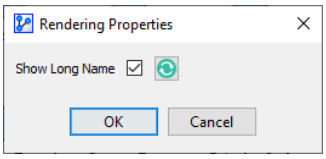
Next, we select the Dictionary file, "AmesLongNames.txt".



Upon loading the Dictionary file, the appearance of the network does not change. Only if an error occurred would a warning triangle appear in the lower right corner of the Graph Window. Also, any error details would be available in the Console.

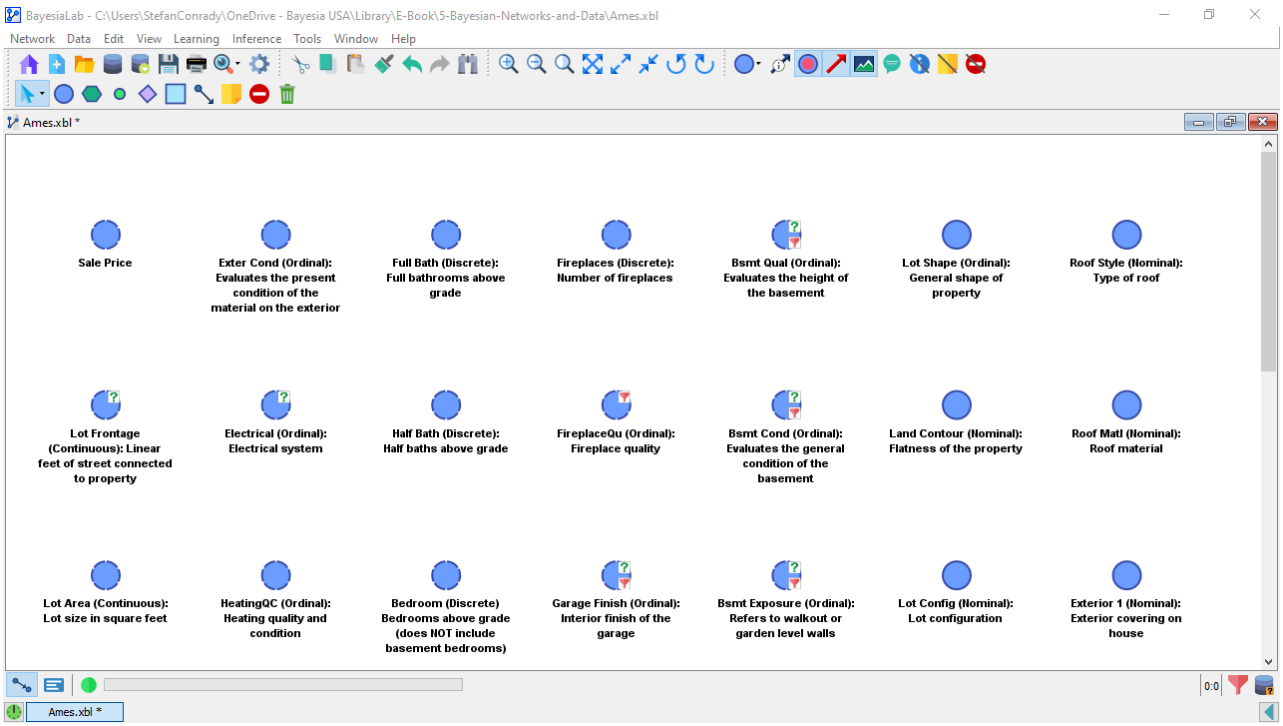
We now have the option of turning on the Long Names for individual nodes or all nodes. For our purposes, we want to see the Long Names on all nodes:

- Select all nodes, e.g., using Ctrl+A.
- `Node Context Menu > Properties > Rendering Properties > Show Long Name`.
- Check the `Show Long Name` box in the pop-up window:

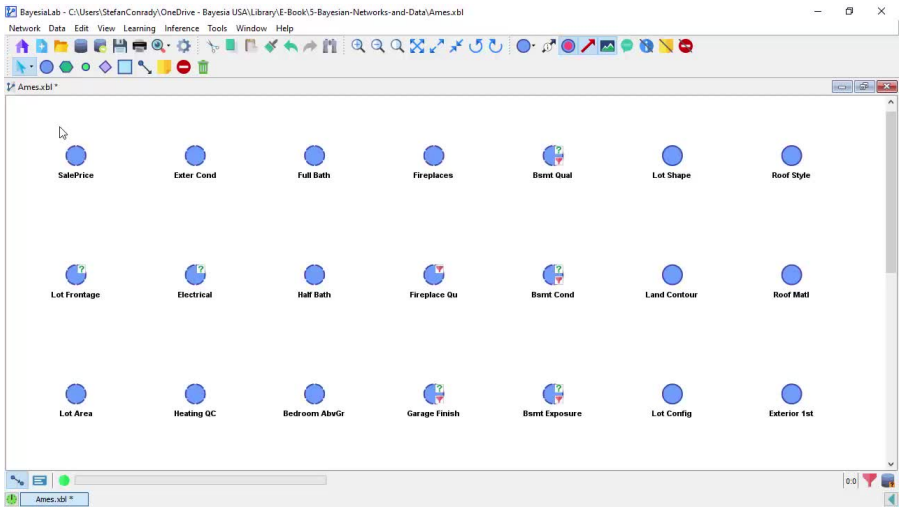


- Click `OK`.

Instead of the "short" Node Names, BayesiaLab now displays the Long Names for all nodes.



Workflow Animation

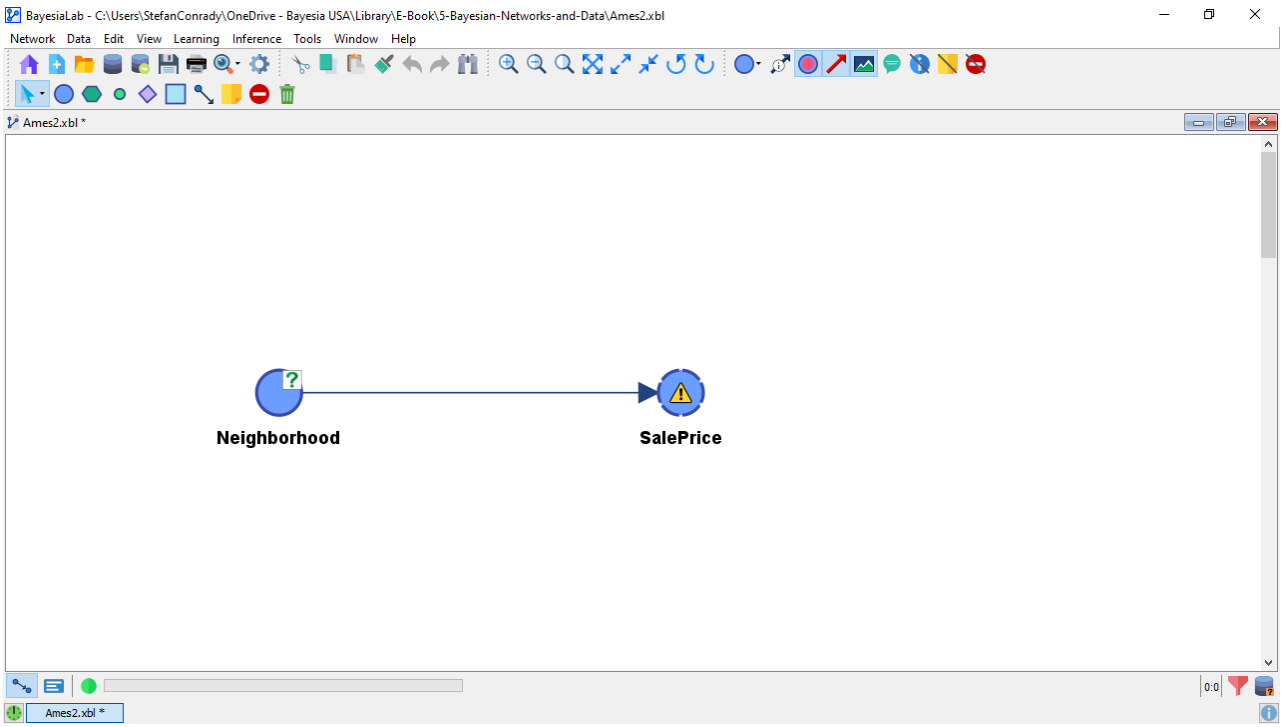


Click to view video

Parameter Estimation

Now we can see the real benefit of bringing all variables as nodes into BayesiaLab. To calculate Mutual Information, all the terms of the equation $I(X,Y) = H(X) - H(X \mid Y)$ can be easily computed with BayesiaLab once we have a fully specified network.

We start with a pair of nodes, namely *Neighborhood* and *SalePrice*. As opposed to *LotArea*, which is a discretized Continuous variable, *Neighborhood* is categorical, and, as such, it has been automatically treated as Discrete in BayesiaLab. This is the reason the node corresponding to *Neighborhood* has a solid border. We now add an arc between these two nodes to explicitly represent the dependency between them:



The yellow warning triangle ⚠ reminds us that the Conditional Probability Table (CPT) of *SalePrice* given *Neighborhood* has not been defined yet. In Chapter 4, we defined the CPT based on existing knowledge. On the other hand, as we have an associated database, BayesiaLab can use it to estimate the CPT by using Maximum Likelihood, i.e., BayesiaLab "counts" the (co-)occurrences of the states of the variables in our data. The table below shows the first 10 records of the variables *SalePrice* and *Neighborhood* from the Ames dataset.

SalePrice	Neighborhood
<=150,000	Gilbert
<=225,000	Northwest Ames
<=150,000	Old Town
>300,000	College Creek
<=225,000	Northridge Heights
<=300,000	Somerset
<=300,000	Gilbert
<=225,000	Sawyer
<=225,000	Sawyer West
<=150,000	Edwards

Counting all records, we obtain the marginal count of each state of *Neighborhood*.

Neighborhood	Count of Neighborhood
1 (Missing)	442
Bloomington Heights	28
Bluestem	10
Briardale	30
Brookside	108
Clear Creek	44
College Creek	267
Crawford	103
Edwards	194
Gilbert	165
Green Hills	2
Greens	8
Iowa DOT and Rail Road	93
Landmark	1
Meadow Village	37
Mitchell	114
Northpark Villa	23
Northridge	71
Northridge Heights	166
Northwest Ames	131
Old Town	239
Sawyer	151
Sawyer West	125
Somerset	182
South & West of Iowa State University	46
Stone Brook	51
Timberland	72
Veenker	24

Given that our Bayesian network structure says that *Neighborhood* is the parent node of *SalePrice*, we now count the states of *SalePrice* conditional on *Neighborhood*. This is simply a cross-tabulation.

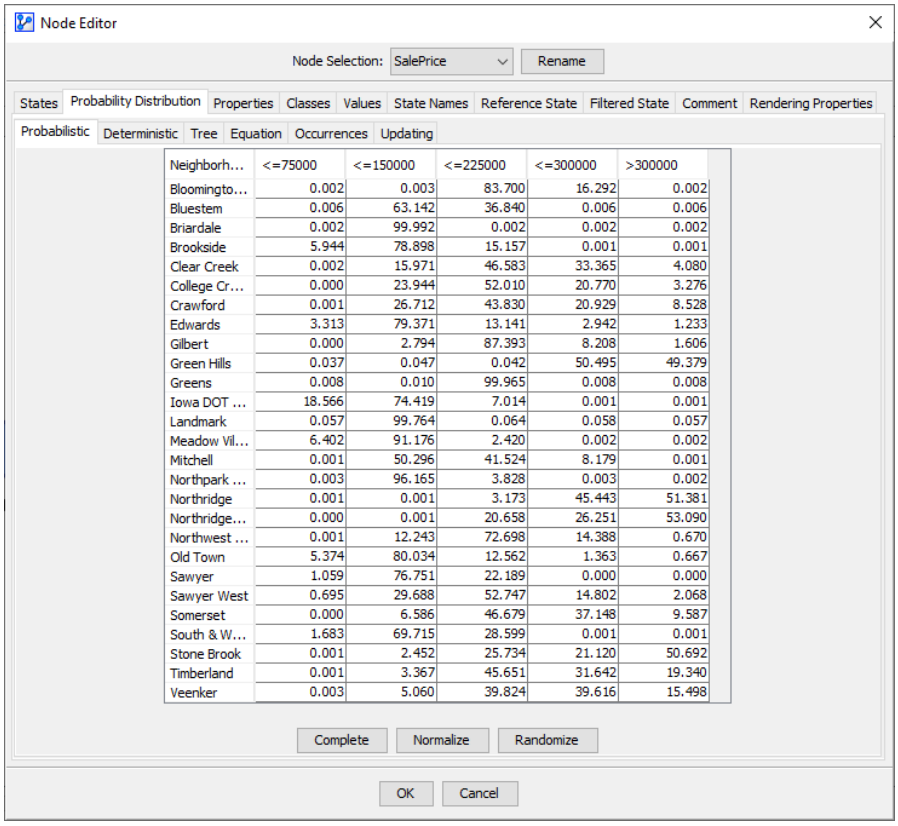
Neighborhood	<=75,000	<=150,000	<=225,000	<=300,000	>300,000
? (Missing)	1	291	137	12	2
Bloomington Heights	0	0	23	5	0
Bluestem	0	6	4	0	0
Briardale	0	30	0	0	0
Brookside	8	82	18	0	0
Clear Creek	0	6	20	16	2
College Creek	0	56	139	62	10
Crawford	0	24	45	24	10
Edwards	8	148	28	7	3
Gilbert	0	4	143	15	3
Green Hills	0	0	0	1	1
Greens	0	0	8	0	0
Iowa DOT and Rail Road	21	65	7	0	0
Landmark	0	1	0	0	0
Meadow Village	3	33	1	0	0
Mitchell	0	53	50	11	0
Northpark Villa	0	22	1	0	0
Northridge	0	0	2	32	37
Northridge Heights	0	0	31	44	91
Northwest Ames	0	14	95	21	1
Old Town	16	184	33	4	2
Sawyer	2	112	37	0	0
Sawyer West	1	33	67	21	3
Somerset	0	10	81	72	19
South & West of Iowa State University	1	32	15	0	0
Stone Brook	0	1	12	11	27
Timberland	0	2	31	24	15
Veenker	0	1	9	10	4

Once we translate these counts into probabilities (by normalizing by the total number of occurrences for each row in the table), this table becomes a CPT. Together, the network structure (qualitative) and the CPTs (quantitative) comprise the Bayesian network.

				Conditional Probability of SalePrice Given Neighborhood				
Neighborhood	Marginal Probability of Neighborhood		Neighborhood	<=75,000	<=150,000	<=225,000	<=300,000	>300,000
Bloomington Heights	1.1		Bloomington Heights	0	0	83.7	16.3	0
Bluestem	0.4		Bluestem	0	63.1	36.8	0	0
Briardale	1.3		Briardale	0	100	0	0	0
Brookside	4.7		Brookside	5.9	78.9	15.2	0	0
Clear Creek	1.7		Clear Creek	0	16	46.6	33.4	4.1
College Creek	10.5		College Creek	0	23.9	52	20.8	3.3
Crawford	4		Crawford	0	26.7	43.8	20.9	8.5
Edwards	8.4		Edwards	3.3	79.4	13.1	2.9	1.2
Gilbert	6.4		Gilbert	0	2.8	87.4	8.2	1.6
Green Hills	0.1		Green Hills	0	0	0	50.5	49.4
Greens	0.3		Greens	0	0	100	0	0
Iowa DOT and Rail Road	3.9		Iowa DOT and Rail Road	18.6	74.4	7	0	0
Landmark	0		Landmark	0.1	99.8	0.1	0.1	0.1
Meadow Village	1.6		Meadow Village	6.4	91.2	2.4	0	0
Mitchell	4.7		Mitchell	0	50.3	41.5	8.2	0
Northpark Villa	1		Northpark Villa	0	96.2	3.8	0	0
Northridge	2.5		Northridge	0	0	3.2	45.4	51.4
Northridge Heights	5.9		Northridge Heights	0	0	20.7	26.3	53.1
Northwest Ames	5.1		Northwest Ames	0	12.2	72.7	14.4	0.7
Old Town	10.3		Old Town	5.4	80	12.6	1.4	0.7
Sawyer	6.6		Sawyer	1.1	76.8	22.2	0	0
Sawyer West	5		Sawyer West	0.7	29.7	52.7	14.8	2.1
Somerset	6.8		Somerset	0	6.6	46.7	37.1	9.6
South & West of Iowa State University	2.1		South & West of Iowa State University	1.7	69.7	28.6	0	0
Stone Brook	1.8		Stone Brook	0	2.5	25.7	21.1	50.7
Timberland	2.7		Timberland	0	3.4	45.7	31.6	19.3
Veenker	0.9		Veenker	0	5.1	39.8	39.6	15.5

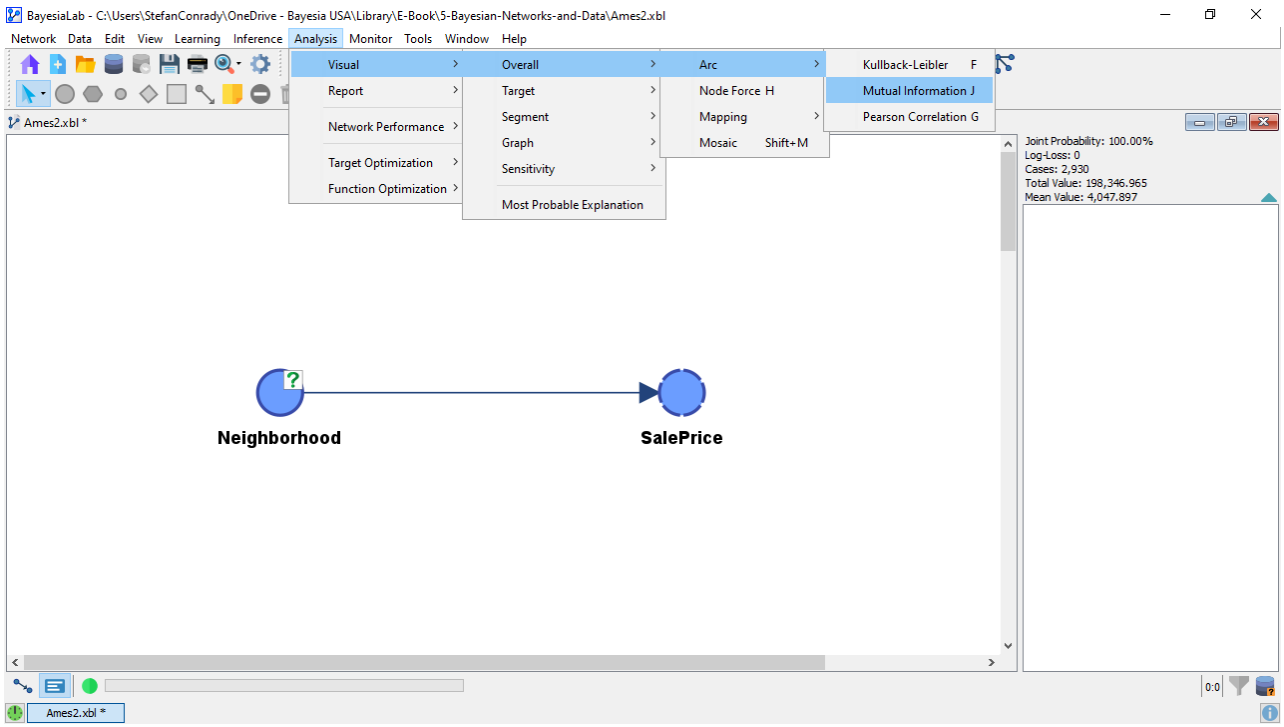
In practice, however, we do not need to bother with these individual steps. Rather, BayesiaLab can automatically learn all marginal and conditional probabilities from the associated database. We select `Menu > Learning > Parameter Estimation` to perform this task.

Upon completing the Parameter Estimation, the warning triangle ⚠ has disappeared, and we can verify the results by double-clicking *SalePrice* to open the Node Editor. Under the tab `Probability Distribution > Probabilistic` we can see the probabilities of the states of *SalePrice* given *Neighborhood*. The CPT presented in the Node Editor is indeed identical to the table shown above.

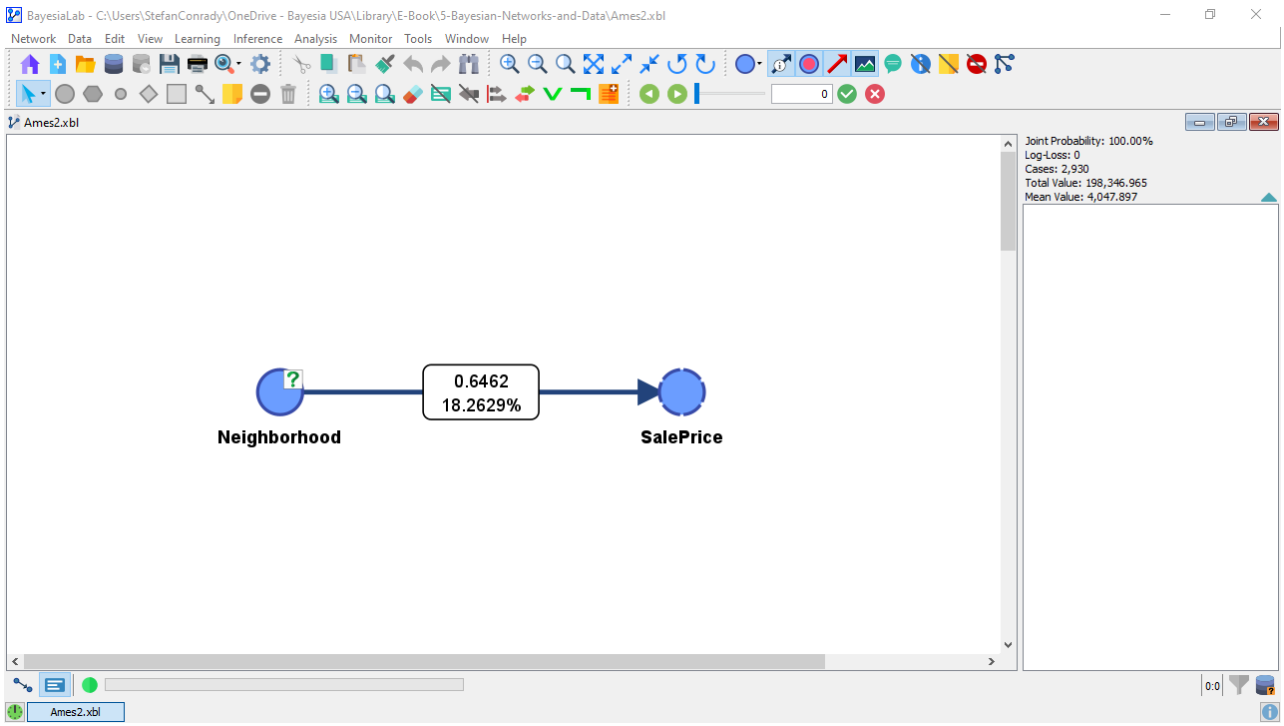


Neighborh...	<=75000	<=150000	<=225000	<=300000	>300000
Bloomingto...	0.002	0.003	83.700	16.292	0.002
Bluestem	0.006	63.142	36.840	0.006	0.006
Briardale	0.002	99.992	0.002	0.002	0.002
Brookside	5.944	78.898	15.157	0.001	0.001
Clear Creek	0.002	15.971	46.583	33.365	4.080
College Cr...	0.000	23.944	52.010	20.770	3.276
Crawford	0.001	26.712	43.830	20.929	8.528
Edwards	3.313	79.371	13.141	2.942	1.233
Gilbert	0.000	2.794	87.393	8.208	1.606
Green Hills	0.037	0.047	0.042	50.495	49.379
Greens	0.008	0.010	99.965	0.008	0.008
Iowa DOT ...	18.566	74.419	7.014	0.001	0.001
Landmark	0.057	99.764	0.064	0.058	0.057
Meadow Vil...	6.402	91.176	2.420	0.002	0.002
Mitchell	0.001	50.296	41.524	8.179	0.001
Northpark ...	0.003	96.165	3.828	0.003	0.002
Northridge	0.001	0.001	3.173	45.443	51.381
Northridge...	0.000	0.001	20.658	26.251	53.090
Northwest ...	0.001	12.243	72.698	14.388	0.670
Old Town	5.374	80.034	12.562	1.363	0.667
Sawyer	1.059	76.751	22.189	0.000	0.000
Sawyer West	0.695	29.688	52.747	14.802	2.068
Somerset	0.000	6.586	46.679	37.148	9.587
South & W...	1.683	69.715	28.599	0.001	0.001
Stone Brook	0.001	2.452	25.734	21.120	50.692
Timberland	0.001	3.367	45.651	31.642	19.340
Veenker	0.003	5.060	39.824	39.616	15.498

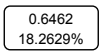
This model now provides the basis for computing the Mutual Information between *Neighborhood* and *SalePrice*. BayesiaLab computes Mutual Information on demand and can display its value in numerous ways. For instance, in Validation Mode F5, we can select `Menu > Analysis > Visual > Arc > Overall > Mutual Information`.



The value of Mutual Information is now represented graphically in the thickness of the arc. This does not give us much insight because we only have a single arc in this network. So, we click the Show Arc Comments icon in the Toolbar to show the numerical values.



The top number in the Arc Comment box shows the actual Mutual Information value, i.e., 0.6462 bits. We should also point out that Mutual Information is a symmetric measure. As such, the amount of Mutual Information that *Neighborhood* provides on *SalePrice* is the same as the amount of MI that *SalePrice* provides with regard to *Neighborhood*. This means that knowing the *SalePrice* reduces the uncertainty with regard to *Neighborhood*, even though that may not be of interest.



Without context, however, the value of Mutual Information is not meaningful. Hence, BayesiaLab provides an additional measure, i.e., the Symmetric Normalized Mutual Information, which gives us a sense of how much the entropy of *SalePrice* was reduced. Previously, we computed the marginal entropy of *SalePrice* to be 1.85. Dividing the Mutual Information by the Marginal Entropy of *SalePrice* gives us a sense of how much our uncertainty is reduced:

$$\frac{0.5999}{1.85} = 0.3243 = 32.43\%$$

Conversely, the red number shows the Relative Mutual Information with regard to the parent node, *Neighborhood*. Here, we divide the Mutual Information, which is the same in both directions, by the Marginal Entropy of *Neighborhood*:

$$\frac{0.5999}{4.1839} = 0.1434 = 14.34\%$$

This means that by knowing *Neighborhood*, we reduce our uncertainty regarding *SalePrice* by 32% on average. By knowing *SalePrice*, we reduce our uncertainty regarding *Neighborhood* by 14% on average. These values are readily interpretable. However, we need to know this for all nodes to determine which node is most important.

Uncertainty, Entropy, and Mutual Information

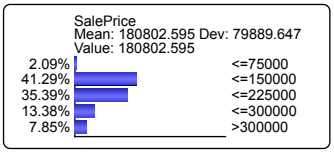
Now that we have the Ames dataset represented internally in BayesiaLab, we need to become familiar with how BayesiaLab can quantify the probabilistic properties of these nodes and their relationships.

In traditional statistical analysis, we would presumably examine correlation and covariance between the variables to establish their relative importance, especially regarding the target variable *SalePrice*. In this chapter, we take an alternative approach based on information theory. Instead of computing the correlation coefficient, we consider how the uncertainty of the states of a to-be-predicted variable is affected by observing a predictor variable.

Beyond our common-sense understanding of uncertainty, there is a more formal quantification of uncertainty in information theory: Entropy. More specifically, we use Entropy to quantify the uncertainty manifested in the probability distribution of a variable or of a set of variables. In the context of our example, the uncertainty relates to the to-be-predicted home price.

It is fair to say that we would need detailed information about a property to predict its value reasonably. However, in the absence of any specific information, would we be entirely uncertain about its value? Probably not. Even if we did not know anything about a particular house, we would have

some contextual knowledge, i.e., that the house is in Ames, Iowa, rather than in midtown-Manhattan, and that the property is a private home rather than a shopping mall. That knowledge significantly reduces the range of possible values. True uncertainty would mean that a value of $0.01 is as probable as a value of $1 million or $1 billion. That is clearly not the case here. So, how uncertain are we about the value of a random home in Ames prior to learning anything about that particular home? The answer is that we can compute the entropy from the marginal probability distribution of home values in Ames. Since we have the Ames dataset already imported into BayesiaLab, we can display a histogram of *SalePrice* by bringing up its Monitor.



This Monitor reflects the discretization intervals that we defined during the data import. It is now easy to see the frequency of prices in each price interval, i.e., the marginal distribution of *SalePrice*. For instance, only about 2% of homes sold had a price of $75,000 or less. On the basis of this probability distribution, we can now compute the Entropy. The definition of Entropy for a discrete distribution is:

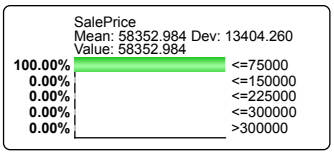
$$H(X) = -\sum_{x \in X} P(x) \log_2 P(x)$$

Entering the values displayed in the Monitor, we obtain:

$$H(SalePrice) = -(0.0208 \times \log_2(0.0208) + 0.413 \times \log_2(0.413) + 0.3539 \times \log_2(0.3539) + 0.1338 \times \log_2(0.1338) + 0.0785 \times \log_2(0.0785)) = 1.85$$

In information theory, the unit of information is "bit", which is why we use base 2 of the logarithm. On its own, the calculated Entropy value of 1.85 bits may not be a meaningful measure. To understand how much or how little uncertainty this value represents, we compare it to two easily-interpretable measures, i.e., "no uncertainty" and "complete uncertainty."

No uncertainty means that the probability of one bin (or state) of *SalePrice* is 100%. This could be, for instance, $P(SalePrice < 75000) = 1$.



We now compute the entropy of this distribution once again:

$$H(SalePrice_{<75{,}000}) = -(1 \times \log_2(1) + 0 \times \log_2(0) + 0 \times \log_2(0) + 0 \times \log_2(0) + 0 \times \log_2(0)) = \log_2(1) = 0$$

Here,

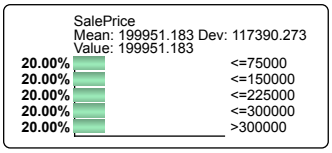
$0 \times \log_2(0)$

is taken as 0, given the limit

$\lim_{p \to 0+} p\log_2(p) = 0$

This means that "no uncertainty" has zero Entropy.

What about the opposite end of the spectrum, i.e., complete uncertainty? Maximum uncertainty exists when all possible states of a distribution are equally probable when we have a uniform distribution:



Once again, we calculate the entropy:

$H(SalePrice_{uniform}) = -(0.2 \times \log_2(0.2) + 0.2 \times \log_2(0.2) + 0.2 \times \log_2(0.2) + 0.2 \times \log_2(0.2) + 0.2 \times \log_2(0.2)) = \log_2(5) = 2.3219$

The value 5 in the logarithm of the simplified equation reflects the number of states. This means that the Entropy is a function of the variable discretization. In addition to the previously computed Marginal Entropy of 1.85, we now have the values 0 and 2.3219 for "no uncertainty" and "complete uncertainty," respectively.

Entropy and Predictive Importance

How do such entropy values help us to establish the importance of predictive variables? If there is no uncertainty regarding a variable, one state of this variable has to have a 100% probability, and predicting that particular state must be correct. This would be like predicting the presence of clouds during rain. On the other hand, if the probability distribution of the target variable is uniform, e.g., the outcome of a fair coin toss, a random prediction has to be correct with a probability of 50%.

In the context of house prices, knowing the marginal distribution of *SalePrice* and assuming this distribution is still true when we make the prediction, predicting $SalePrice \geq 150{,}000$ would have a 41.28% probability of being correct, even if we knew nothing else. However, we would expect that observing an attribute of a specific home would reduce our uncertainty concerning its *SalePrice* and increase our probability of making a correct prediction for this particular home. In other words, conditional upon learning an attribute of a home, i.e., by observing a predictive variable, we expect a lower uncertainty for the target variable, *SalePrice*.

For instance, the moment we learn of a particular home that $LotArea = 200{,}000$ (measured in square feet. 200,000 ft2 ≈ 4.6 acres ≈ 18,851 m2 ≈ 1.86 ha), and assuming, again, that the estimated marginal distribution is still true when we are making the prediction, we can be certain that $SalePrice > 300{,}000$. This means that upon learning the value of this home's *LotArea*, the entropy of *SalePrice* goes from 1.85 to 0. Learning the size reduces our entropy by 1.85 bits. Alternatively, we can say that we gain information amounting to 1.85 bits.

The information gain or entropy reduction from learning about *LotArea* of this house is obvious. Observing a different home with a more common lot size, e.g., *LotArea* = 10,000, would presumably provide less information and, thus, have less predictive value for that home.

However, we wish to know how much information we would gain on average–considering all values of *LotArea* along with their probabilities–by generally observing it as a predictive variable for *SalePrice*. Knowing this "average information gain" would reflect the predictive importance of observing the variable *LotArea*.

To compute this, we need two quantities. First, the marginal entropy of the target variable $H(SalePrice)$, and second, the conditional entropy of the target variable given the predictive variable:

$$H(SalePrice \mid LotArea) = \sum_i P(LotArea_i)\, H(SalePrice \mid LotArea_i)$$

The difference between the marginal entropy of the target variable and the conditional entropy of the target given the predictive variable is formally known as Mutual Information, denoted by I. In our example, the Mutual Information I between *SalePrice* and *LotArea* is the marginal entropy of *SalePrice* minus the conditional entropy of *SalePrice* given *LotArea*:

$$I(SalePrice, LotArea) = H(SalePrice) - H(SalePrice \mid LotArea)$$

More generally, the Mutual Information I between variables X and Y is defined by:

$$I(X, Y) = H(X) - H(X \mid Y)$$

which is equivalent to:

$$I(X, Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 \frac{p(x,y)}{p(x)p(y)}$$

and furthermore also equivalent to:

$$I(X, Y) = \sum_{y \in Y} p(y) \sum_{x \in X} p(x \mid y) \log_2 \frac{p(x|y)}{p(x)}$$

This allows us to compute the Mutual Information between a target variable and any possible predictors. As a result, we can find out which predictor provides the maximum information gain and, thus, has the greatest predictive importance.

Chapter 6: Supervised Learning

In Chapter 4 we defined the qualitative and quantitative parts of a Bayesian network from existing (human) knowledge. Chapter 5 described how we can define the qualitative part of a Bayesian network manually and then use data to estimate the quantitative part. In this chapter, we use BayesiaLab to generate both the structure and the parameters of a network automatically from data. This means we introduce machine learning for building Bayesian networks. The only guidance (or constraint) we provide is defining the variable of interest, i.e., the target of the machine-learning process. Hence, we speak of Supervised Learning (in Chapter 7, we will remove that constraint as well and perform Unsupervised Learning).

The objective of what we call Supervised Learning is no different from that of predictive modeling. We wish to find regularities (a model) between the target variable and potential predictors from observations (e.g., historical data). Such a model will allow us to infer a distribution of the target variable from new observations. If the target variable is Continuous, the predicted distribution produces an expected value. For a Discrete target variable, we perform classification. The latter will be the objective of the example in this chapter.

Example: Tumor Classification

Given the sheer amount of medical knowledge in existence today, plus advances in artificial intelligence, so-called medical expert systems have emerged, which are meant to support physicians in performing medical diagnoses. In this context, several papers by Wolberg, Street, Heisey, and Managasarian became much-cited examples. For instance, Street, Wolberg, Mangasarian (1993) proposed an automated method for the classification of Fine-Needle Aspirates (FNA) through imaging processing and machine learning with the objective of achieving greater accuracy in distinguishing between malignant and benign cells for the diagnosis of breast cancer. At the time of their study, the practice of visual inspection of FNA yielded inconsistent diagnostic accuracy. The proposed new approach would increase this accuracy reliably to over 95%. This research was quickly translated into clinical practice and has since been applied with continued success.

As part of their studies in the late 1980s and 1990s, the research team generated what became known as the Wisconsin Breast Cancer Database, which contains measurements of hundreds of FNA samples and the associated diagnoses. Several versions of this database have been extensively studied, even outside the medical field. Statisticians and computer scientists have proposed a wide range of techniques for this classification problem and have continuously raised the benchmark for predictive performance.

The objective of this chapter is to show how Bayesian networks, in conjunction with machine learning, can be used for classification. Furthermore, we wish to illustrate how Bayesian networks can help researchers generate a deeper understanding of the underlying problem domain. Beyond merely producing predictions, we can use Bayesian networks to precisely quantify the importance of individual variables and employ BayesiaLab to help identify the most efficient path towards diagnosis.

To provide further background regarding this example, we quote Mangasarian et al. (1994):

"Most breast cancers are detected by the patient as a lump in the breast. The majority of breast lumps are benign, so it is the physician's responsibility to diagnose breast cancer, that is, to distinguish benign lumps from malignant ones. There are three available methods for diagnosing breast cancer: mammography, FNA with visual interpretation, and surgical biopsy. The reported sensitivity (i.e., ability to correctly diagnose cancer when the disease is present) of mammography varies from 68% to 79%, of FNA with visual interpretation from 65% to 98%, and of surgical biopsy close to 100%.

Therefore mammography lacks sensitivity, FNA sensitivity varies widely, and surgical biopsy, although accurate, is invasive, time-consuming, and costly. The goal of the diagnostic aspect of our research is to develop a relatively objective system that diagnoses FNAs with an accuracy that approaches the best achieved visually."

Data

The Wisconsin Breast Cancer Database was created through the clinical work of Dr. William H. Wolberg at the University of Wisconsin Hospitals in Madison.

The dataset we are using for this tutorial contains 569 patient records, which contain a diagnosis plus features that were computed from digital images of fine-needle aspirates (FNA) of breast masses. More specifically, these features characterize the cell nuclei contained in the tissue samples.

- ID number
- Diagnosis (M=malignant, B=benign)
- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

For each feature, the mean, standard error, and "worst" or largest (mean of the three largest values) were computed. For this tutorial, however, we only use the mean values as variables.

The diagnosis variable was established via subsequent biopsies or long-term monitoring of the tumor. It consists of two classes: 357 benign cases (62.7%) and 212 malignant cases (37.2%).

You can download this dataset in CSV format via the link below or from data.world.

Download: `WBCD2.csv`

Note that this dataset from the Wisconsin Breast Cancer Database is different from the one we used in the original, printed edition of this book.

Tutorial

The following topics explain each step of the Supervised Learning workflow on the basis of this example.

- Data Import and Discretization
- Supervised Learning: Markov Blanket
- Supervised Learning: Structural Coefficient Analysis
- Inference: Automatic Evidence-Setting
- Inference: Adaptive Questionnaire

Data Import and Discretization

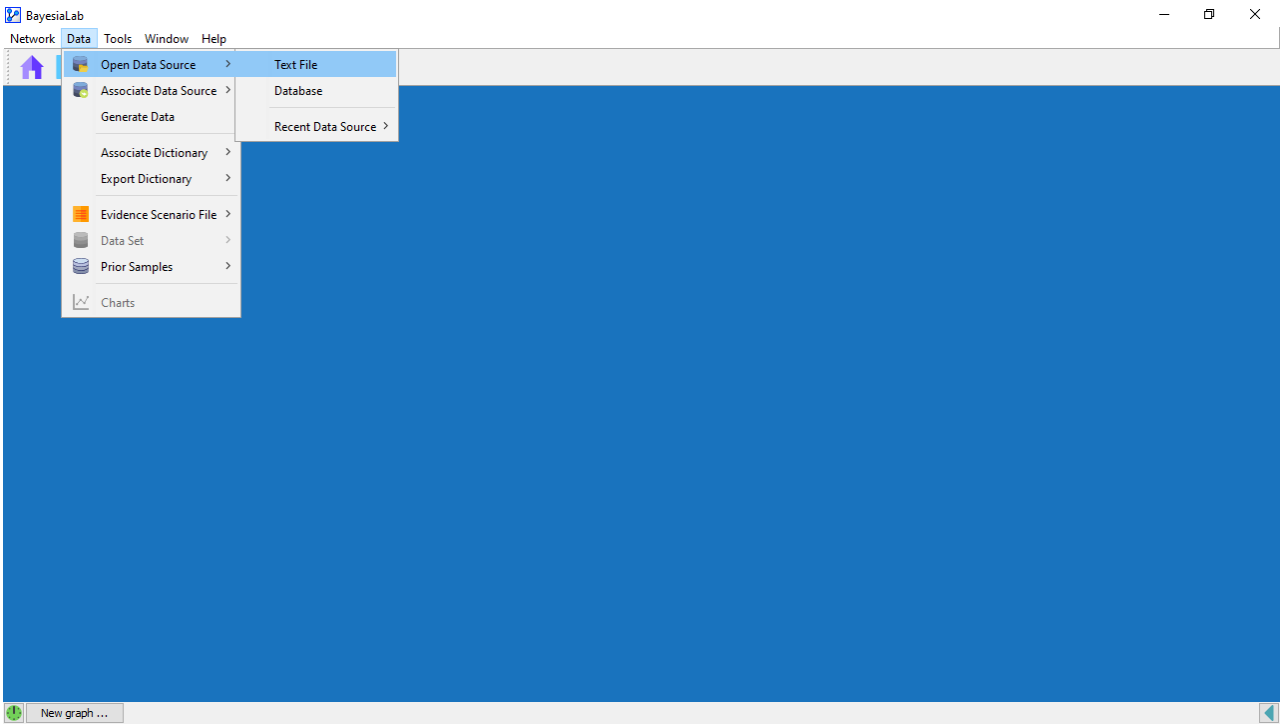
Our modeling process begins with importing the dataset. You can download this dataset in CSV format via the link below or from data.world.

Download: `WBCD2.csv`

Note that this dataset from the Wisconsin Breast Cancer Database differs from the one we used in the original, printed edition of this book.

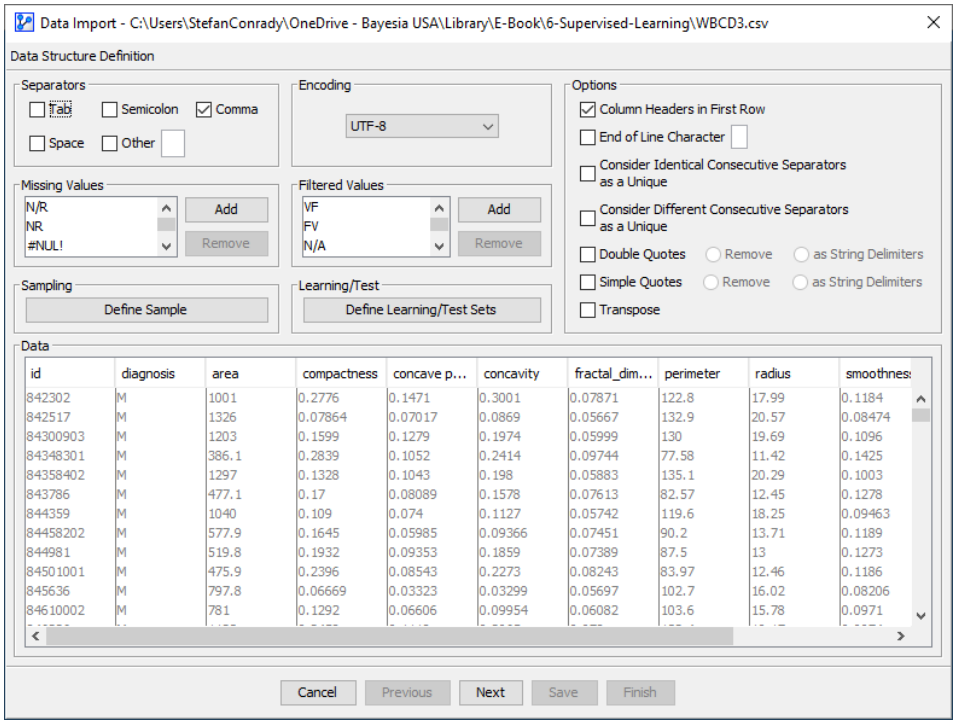
Data Import Wizard

We start the Data Import Wizard with `Menu > Data > Open Data Source > Text File`.



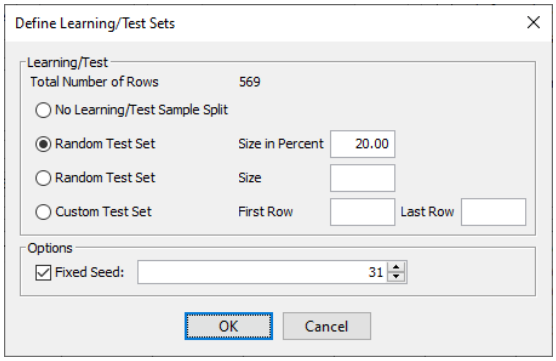
Next, we select the file WBCD2.CSV. Then, the Data Import Wizard guides us through the required steps.

In Step 1 of the Data Import Wizard, we click on `Define Learning/Test Sets` and specify to set aside a Test Set from the to-be-imported dataset.



We specify a random sample of 20% of the entire dataset to serve as a Test Set. The remaining 80% will serve as the Learning Set.

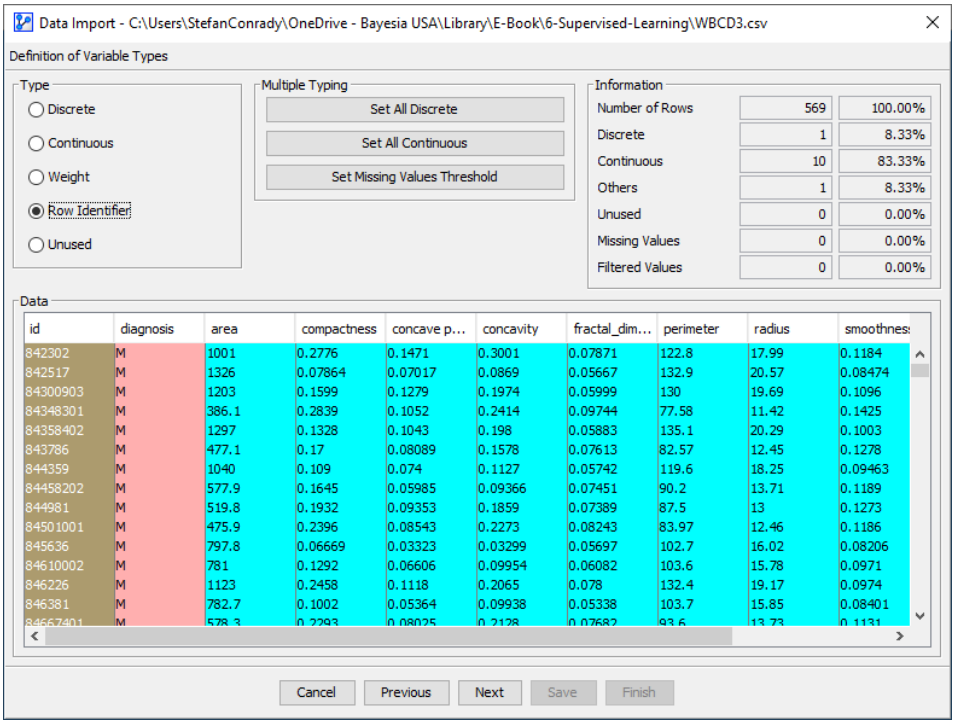
> If you follow this tutorial and want to replicate the exact numerical values we present here, please check `Fixed Seed` under Options and set its value to 31. This ensures that the random number generator produces the same Learning Set and Test Set split that we use here.



In Step 2 of the Data Import Wizard BayesiaLab suggests a data type for each variable.

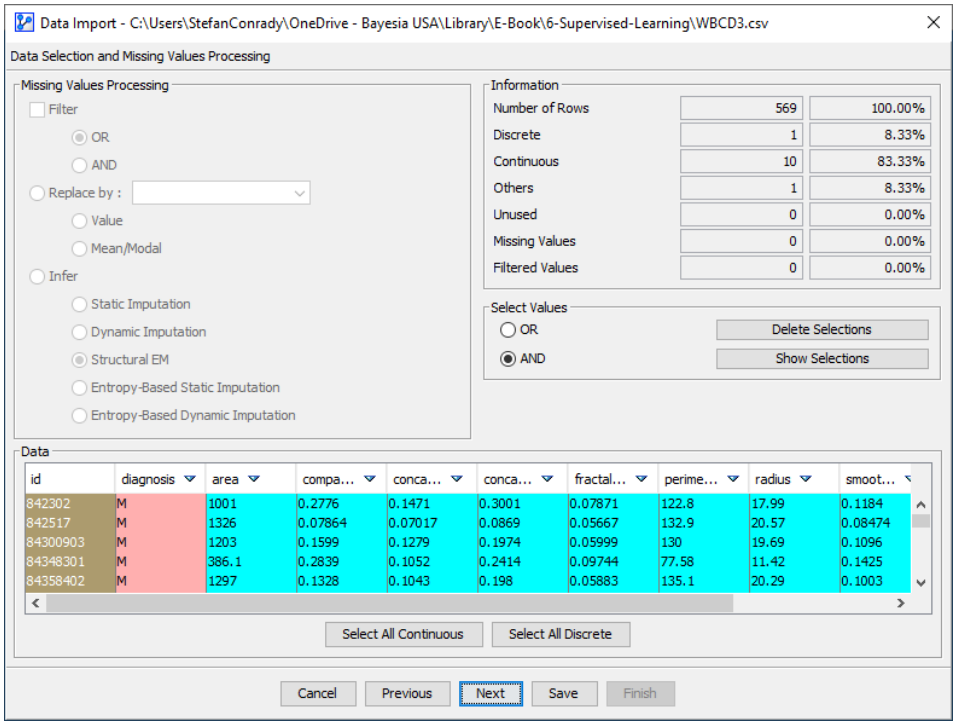
It identifies the diagnosis variable as Discrete, and all the feature variables are interpreted as Continuous. These default assignments are all correct.

We only need to correct the variable id, which BayesiaLab initially considers Continuous. However, id is a code to identify each patient, so we must specify it as a Row Identifier.



In Step 3 of the Data Import Wizard, no action is required. Our dataset has no missing values, so applying any Missing Values Processing is unnecessary.

However, given that many datasets do contain missing values, we devoted an entire chapter to dealing with that problem. Please see Chapter 9: Missing Values Processing.



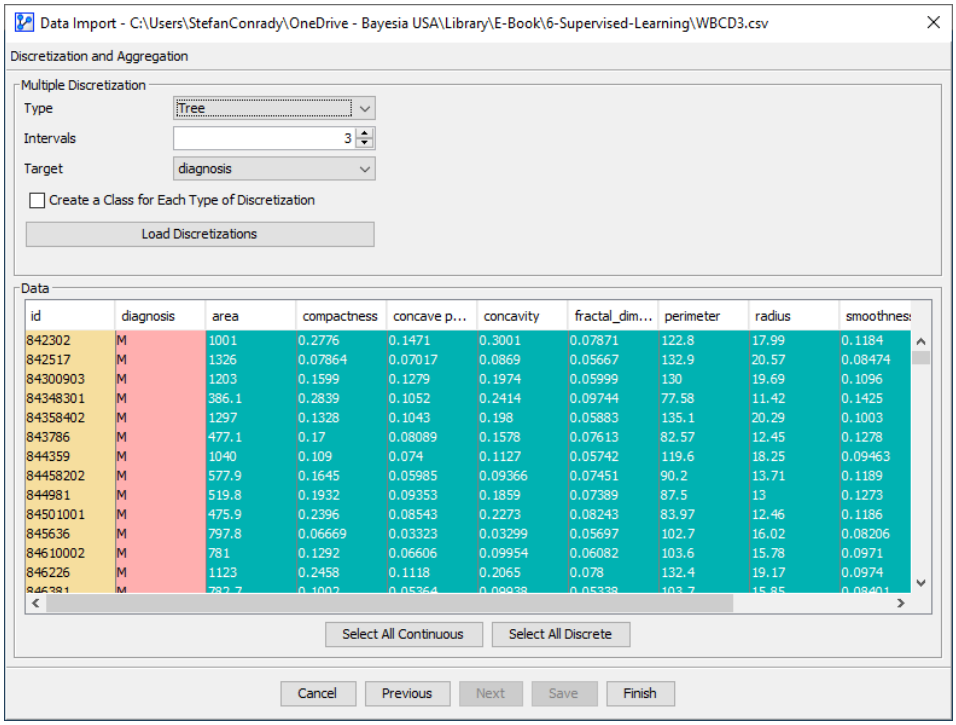
In Step 4 of the Data Import Wizard, we need to discretize the Continuous variables in the dataset. Even though we could specify a discretization method for each Continuous variable separately, we want to apply the same algorithm to all.

So, we click `Select All Continuous`, and all Continuous variables are highlighted in the data table. The Discretization Type and all related options will now apply to all the selected nodes.

Given that we are building a model to predict the target variable diagnosis, it makes perfect sense to discretize all the continuous feature variables with that objective in mind.

Thus, we choose the `Tree` algorithm from the drop-down menu in the `Multiple Discretization` panel. The `Tree` algorithm attempts to find discretization thresholds so that each feature variable's information gain with regard to the target variable is maximized.

Note that the `Tree` algorithm requires a Target, which has to be a Discrete variable or a Continuous variable that has already been discretized. In our context, diagnosis is Discrete and, therefore, available from the Target dropdown menu.



Note that the discretization algorithm only uses the records from the Learning Set for creating the discretization threshold. If a Learning/Test Set split is specified in Step 1 of the Data Import Wizard, BayesiaLab automatically restricts the discretization algorithms to the Learning Set.

> While you can also create a Learning/Test Set split after completing the Data Import Wizard, it would compromise the Test Set. Such a Test Set would no longer be properly out-of-sample as it would have contributed to the discretization.

Discretization Intervals

Bayesian networks are non-parametric probabilistic models. Therefore, there is no hypothesis with regard to the form of the relationships between variables (e.g., linear, quadratic, exponential, etc.). However, this flexibility has a cost. The number of observations necessary to quantify probabilistic relationships is higher than those required in parametric models. We use the heuristic of five observations per probability cell, which implies that the bigger the size of the probability tables, the larger must be the number of observations.

Two parameters affect the size of a probability table: the number of parents and the number of states of the parent and child nodes. A machine-learning algorithm usually determines the number of parents based on the strength of the relationships and the number of available observations. The number of states, however, is our choice, which we can set by means of Discretization (for Continuous variables) and Aggregation (for Discrete variables).

We can use our heuristic of five observations per probability cell to help us with the selection of the number of discretization Intervals:

$$StateCount^{ParentCount+1} \times 5 \leq ObservationCount$$

We usually look for an odd number of states to be able to capture non-linear relationships. Given that we have a relatively small learning set of only 456 observations, we should estimate how many parents would be allowed based on this heuristic and a discretization with 3 states:

- No parent: 3×5=15
- One parent: 3×3×5=45
- Two parents: 3×3×3×5=135
- Three parents: 3×3×3×3×5=405
- Four parents: 3×3×3×3×3×5=1,215

Considering a discretization with 5 states, we would obtain the following:

- No parent: 5×5=25
- One parent: 5×5×5=125
- Two parents: 5×5×5×5=625

By using this heuristic, we hypothesize about the size of the biggest CPT of the to-be-learned Bayesian network and multiply this value by 5. Experience tells us that this is a rather practical heuristic, which typically helps us find a structure. However, this is by no means a guarantee that we will find a precise quantification of the probabilistic relationships.

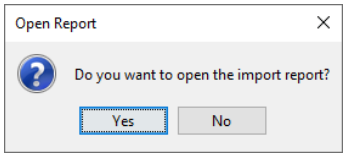
Indeed, our heuristic is based on the hypothesis that all the cells of the CPT are equally likely to be sampled. Of course, such an assumption cannot hold as the sampling probability of a cell depends on its probability, i.e., either a marginal probability if the node does not have parents, or if it does have parents, a joint probability defined by the parent states and the child state.

Given our 456 observations and the scenarios listed above, we select a discretization scheme with a maximum of 3 states. This is a maximum in the sense that the Tree discretization algorithm could return 2 states if 3 were not needed. This happens to be the case for *fractal_dimension_se* and *texture_se*.

Import Report

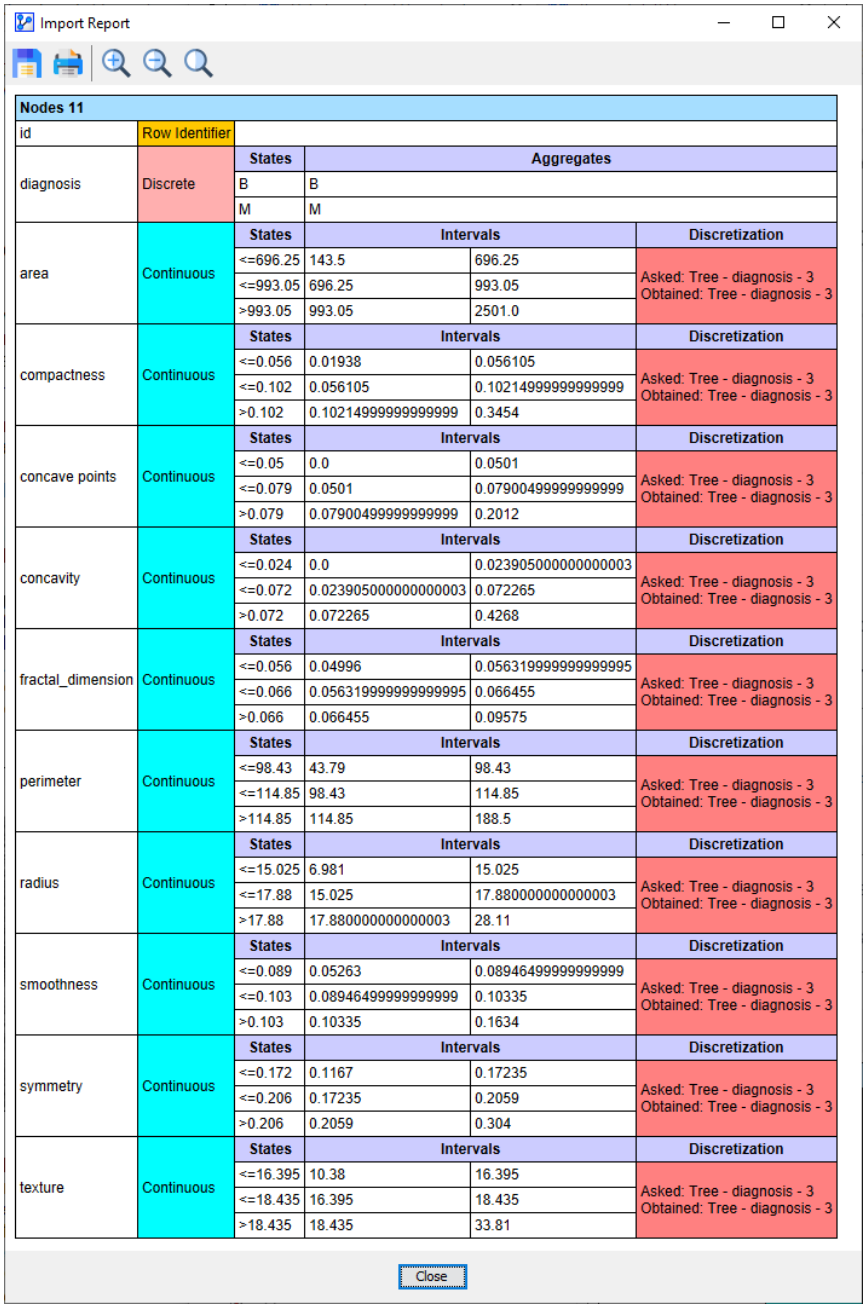
Upon clicking `Finish`, BayesiaLab imports and discretizes the entire dataset and concludes with Step 5 of the Data Import Wizard by offering an Import Report.

Clicking `Yes` brings up the Import Report.



It is interesting to see that all the variables have indeed been discretized with the Tree algorithm and that all Discretization Intervals are variable-specific.

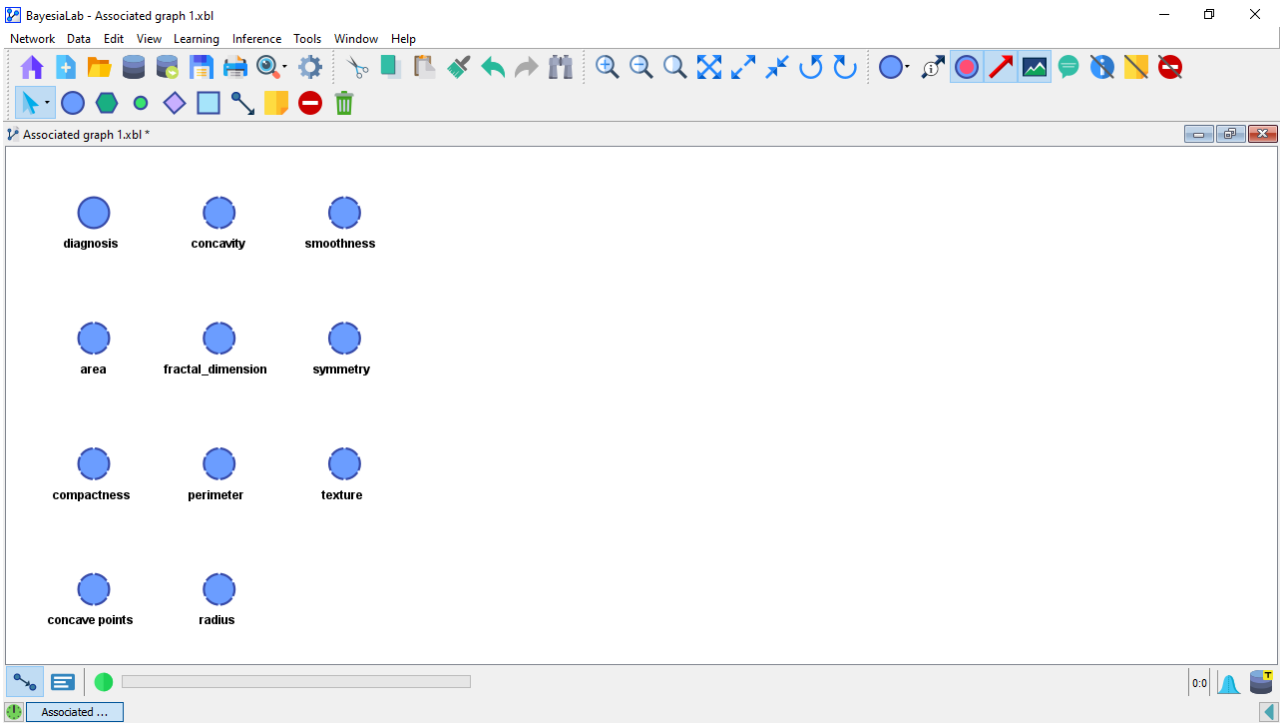
This means that all the variables are marginally dependent on the Target Node (and vice versa). This is promising: the more dependent variables we have, the easier it should be to learn a good model for predicting the Target Node.



Nodes 11					
id	Row Identifier				
diagnosis	Discrete	States	Aggregates		
		B	B		
		M	M		
area	Continuous	States	Intervals		Discretization
		<=696.25	143.5	696.25	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=993.05	696.25	993.05	
		>993.05	993.05	2501.0	
compactness	Continuous	States	Intervals		Discretization
		<=0.056	0.01938	0.056105	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.102	0.056105	0.10214999999999999	
		>0.102	0.10214999999999999	0.3454	
concave points	Continuous	States	Intervals		Discretization
		<=0.05	0.0	0.0501	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.079	0.0501	0.07904999999999999	
		>0.079	0.07904999999999999	0.2012	
concavity	Continuous	States	Intervals		Discretization
		<=0.024	0.0	0.023905000000000003	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.072	0.023905000000000003	0.072265	
		>0.072	0.072265	0.4268	
fractal_dimension	Continuous	States	Intervals		Discretization
		<=0.056	0.04996	0.056319999999999995	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.066	0.056319999999999995	0.066455	
		>0.066	0.066455	0.09575	
perimeter	Continuous	States	Intervals		Discretization
		<=98.43	43.79	98.43	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=114.85	98.43	114.85	
		>114.85	114.85	188.5	
radius	Continuous	States	Intervals		Discretization
		<=15.025	6.981	15.025	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=17.88	15.025	17.880000000000003	
		>17.88	17.880000000000003	28.11	
smoothness	Continuous	States	Intervals		Discretization
		<=0.089	0.05263	0.08946499999999999	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.103	0.08946499999999999	0.10335	
		>0.103	0.10335	0.1634	
symmetry	Continuous	States	Intervals		Discretization
		<=0.172	0.1167	0.17235	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=0.206	0.17235	0.2059	
		>0.206	0.2059	0.304	
texture	Continuous	States	Intervals		Discretization
		<=16.395	10.38	16.395	Asked: Tree - diagnosis - 3 Obtained: Tree - diagnosis - 3
		<=18.435	16.395	18.435	
		>18.435	18.435	33.81	

Graph Window

Upon closing the Import Report, we see a representation of the newly imported database as a fully unconnected Bayesian network in the Graph Window.



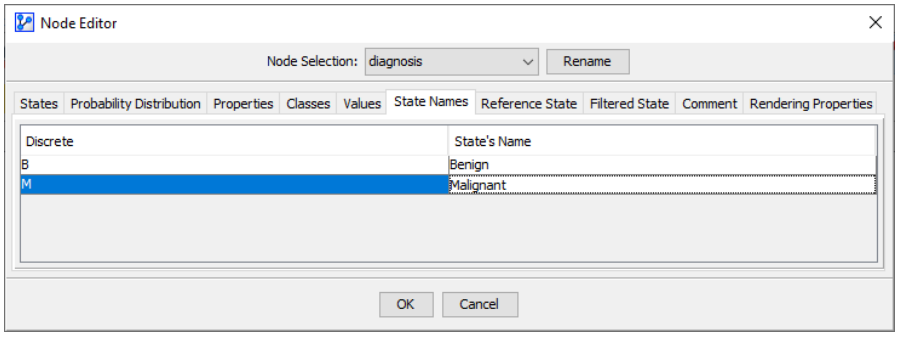
Additionally, there is a tag on the database icon in the lower right corner of the Graph Window: The T icon confirms that we have a Learning/Test Set split in place.

State Names

In the dataset, the variable *diagnosis* contained the codes B and M, representing Benign and Malignant, respectively. For reading the analysis reports, however, it will be easier to work with a proper State Name instead of an abbreviation.

By double-clicking the node *diagnosis*, we open the Node Editor and then go to the `State Names` tab. There, we associate States B and M with new State Names:

- B → Benign
- M → Malignant



With all variables represented as nodes in the Graph Window, we are ready to proceed to Supervised Learning in this tutorial.

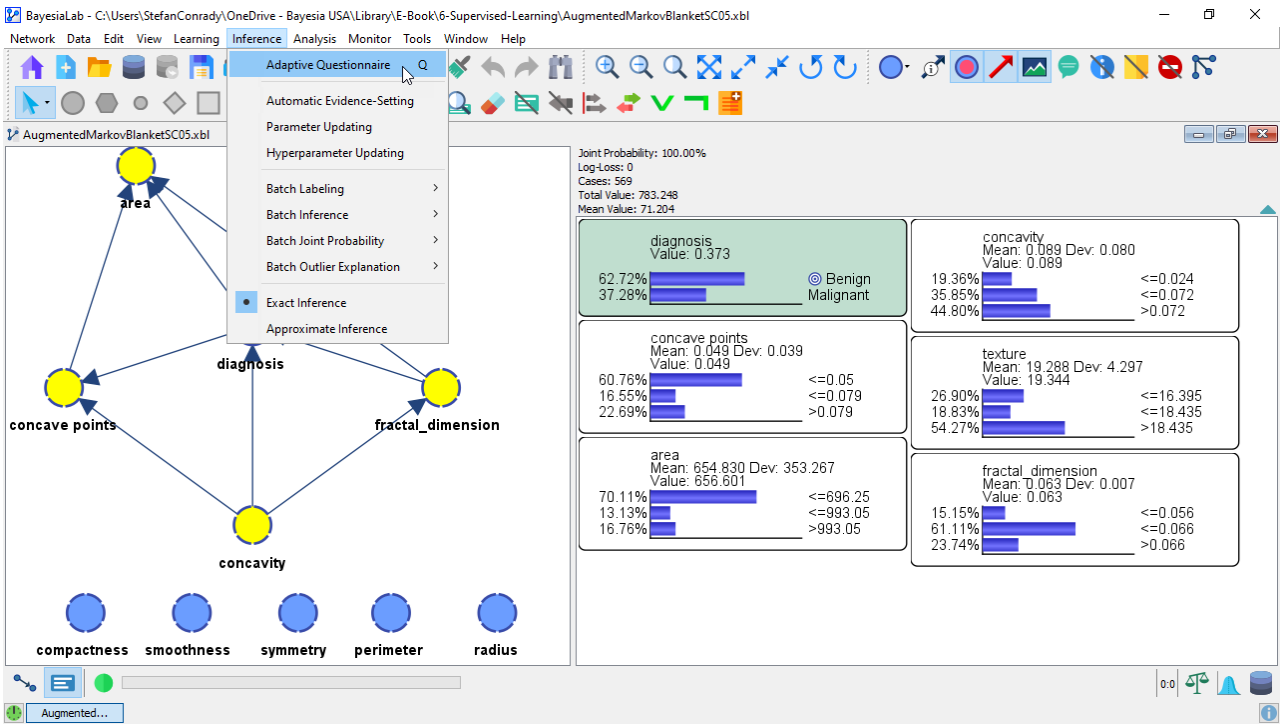
Inference: Adaptive Questionnaire

Context

- In situations in which individual cases are under review, e.g., when diagnosing a patient, BayesiaLab can provide diagnostic support by means of the Adaptive Questionnaire.
- This approach helps prioritize what variable to investigate or what pieces of evidence to collect in order to reduce the uncertainty regarding a target variable of interest.
- Whenever you have a Bayesian network with a Target Node, regardless of whether the network was machine-learned or created from expert knowledge, you can launch the Adaptive Questionnaire.
- Importantly, the Adaptive Questionnaire seeks the optimal sequencing of evidence for a specific case or instance rather than creating a set of rules that apply in general.
- For creating a generalized set of priorities, please see Target Interpretation Tree in this chapter.

Usage

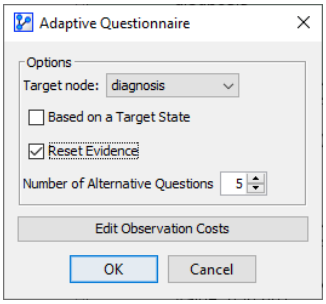
The Adaptive Questionnaire can be started via `Menu > Inference > Adaptive Questionnaire`.



For a Target Node with more than two states, it can be helpful to specify a Target State for the Adaptive Questionnaire.

Setting a Target Node allows BayesiaLab to compute the Binary Mutual Information and then focus on the designated Target State.

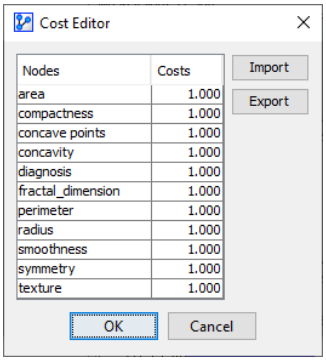
However, as the Target Node in our example is binary by default, setting a Target State is superfluous.



Costs

Furthermore, we can set the cost of collecting observations via the Cost Editor, which can be started by clicking the `Edit Observations Costs` button.

This is helpful if certain variables are more costly to observe or require more effort to obtain than others. So, Costs do not necessarily have to represent a financial cost. For instance, we could make Costs proportional to the difficulty of collecting observations.



Adaptive Evidence-Seeking

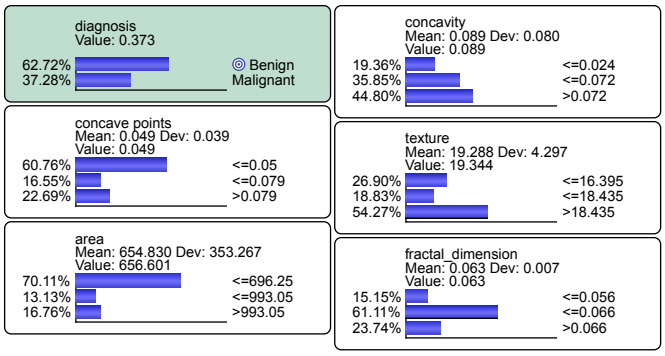
> In analyzing Fine Needle Aspirates, all image attributes are obtained simultaneously. As a result, this particular domain is not ideal for demonstrating the Adaptive Questionnaire. A better example would be a diagnostic process, in which a clinician collects observations from a patient in a targeted way. We can imagine that a physician starts the diagnosis process by collecting easy-to-obtain data, such as blood pressure, before proceeding to more elaborate (and more expensive) diagnostic techniques, such as performing an MRI.

Here, we simulate using the Adaptive Questionnaire as if we could choose the order of collecting evidence.

After starting the Adaptive Questionnaire, BayesiaLab presents the Monitor of the Target Node and displays its marginal probability. That Monitor is highlighted in green.

Furthermore, the Monitors are automatically sorted in descending order with regard to the Target Node by taking into account the Mutual Information (or Binary Mutual Information, if applicable) and the Cost of obtaining the evidence:

1. *concave points*
2. *area*
3. *concavity*
4. *texture*
5. *fractal_dimension*



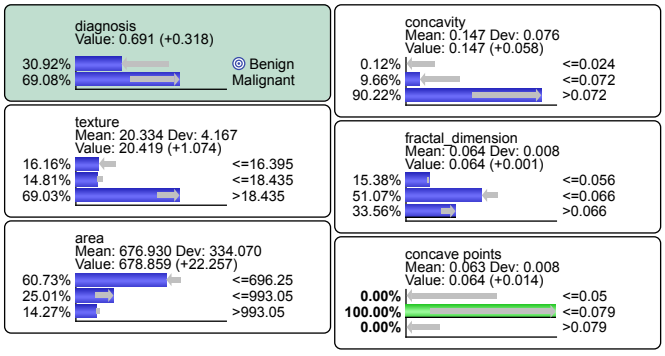
Given this order, it would be ideal to collect the value of *concave points* as the first observation.

Observation #1

Let us suppose that we can do that and obtain *concave points* ≤ 0.079 as the first piece of evidence.

Upon setting that state in the Monitor of *concave points*, the Monitor Panel is updated as follows:

- In the Monitor of the Target Node, we see that the probability of *diagnosis* = *malignant* has increased to 69.08%.
- The order of Monitors is resorted according to Mutual Information and Cost:
  1. *texture*
  2. *area*
  3. *concavity*
  4. *fractal_dimension*
- The Monitor of the node we just observed drops to the bottom of the list. Given that we already know its value, no further information can be gained from it.
- Also, the distributions of all the not-yet-observed nodes have changed, with *concavity* increasing substantially.
- The small gray arrows inside the Monitors indicate how much the probabilities have changed.



Note that we are not merely seeing the next-in-line Monitor "moving up." Rather, the entire list is recomputed, given the most recent piece of evidence.

For instance, when we started the Adaptive Questionnaire, *area* was two spots ahead of *texture*. Given the last observation, however, *texture* has become more important than area.

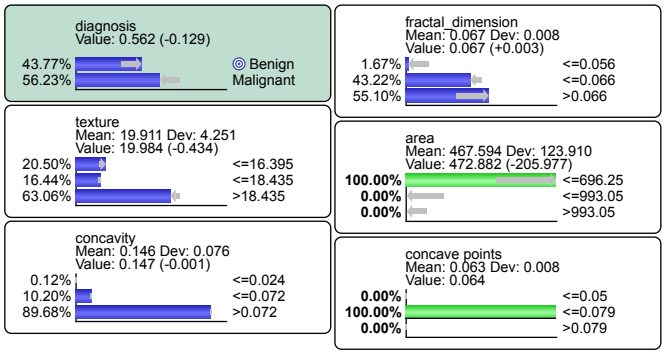
Observation #2

So, given the above ordering, *texture* would be the next best evidence to obtain.

In real-world applications, it is possible that the ideal evidence is not available and, therefore, must be skipped. We simulate such a situation by observing *area* instead of *texture*.

We find $area \leq 696.25$ and enter that evidence in the corresponding Monitor:

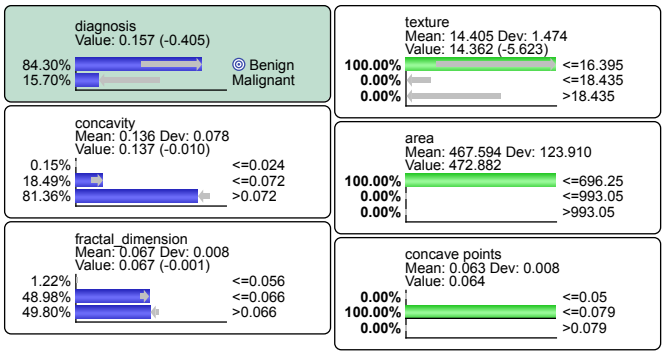
- The probability of $diagnosis = malignant$ decreases to 56.23%
- The order of the remaining unobserved nodes is now:
  1. *texture*
  2. *concavity*
  3. *fractal_dimension*



Observation #3

For this iteration, we follow the top recommendation, i.e., *texture*, and observe $texture \leq 16.395$.

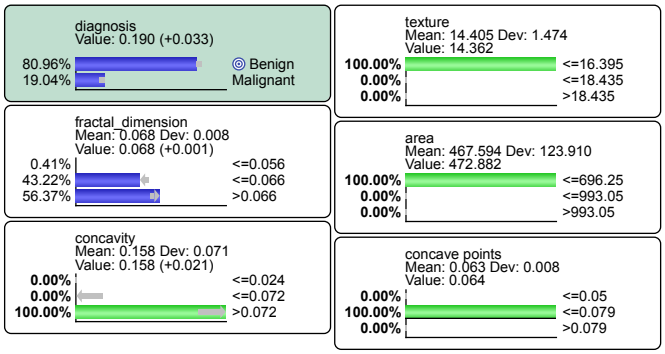
- The probability of $diagnosis = malignant$ decreases further to 15.70%
- The order of the remaining unobserved nodes is now:
  1. *concavity*
  2. *fractal_dimension*



Observation #4

For this iteration, we follow the recommendation and observe *concavity* > 0.072.

- The probability of $diagnosis = malignant$ increased to 19.04%
- At this point, only *fractal_dimension* remains unobserved.

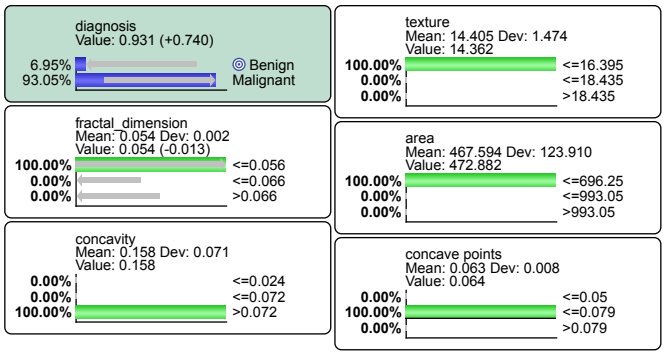


Observation #5

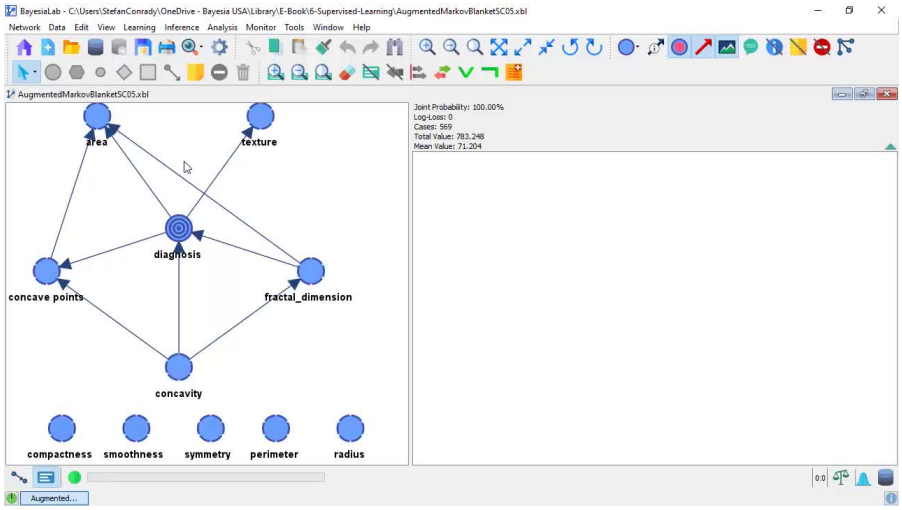
For the last node, *fractal_dimension*, we obtain $fractal\_dimension \leq 0.056$

In this hypothetical example, the last observation appears to have a rather substantial impact on the diagnosis.

The Monitor of the Target Node now reports that $diagnosis = malignant$ has a probability of 93.05%



Workflow Animation



Click to view video

Summary & Next Steps

The Adaptive Questionnaire is a highly practical tool for seeking the optimal next piece of evidence when trying to determine the state of a Target Node.

We used the Adaptive Questionnaire via the Graphical User Interface in BayesiaLab in this example. For situations when end-users do not have access to the BayesiaLab software, you can publish an Adaptive Questionnaire via the WebSimulator. This allows anyone to interact with an Adaptive Questionnaire through a web browser.

Finally, BayesiaLab can produce a static version of the Adaptive Questionnaire, which can be used entirely offline. This tool is the Target Interpretation Tree, which we discuss in the next section.

Inference: Automatic Evidence-Setting

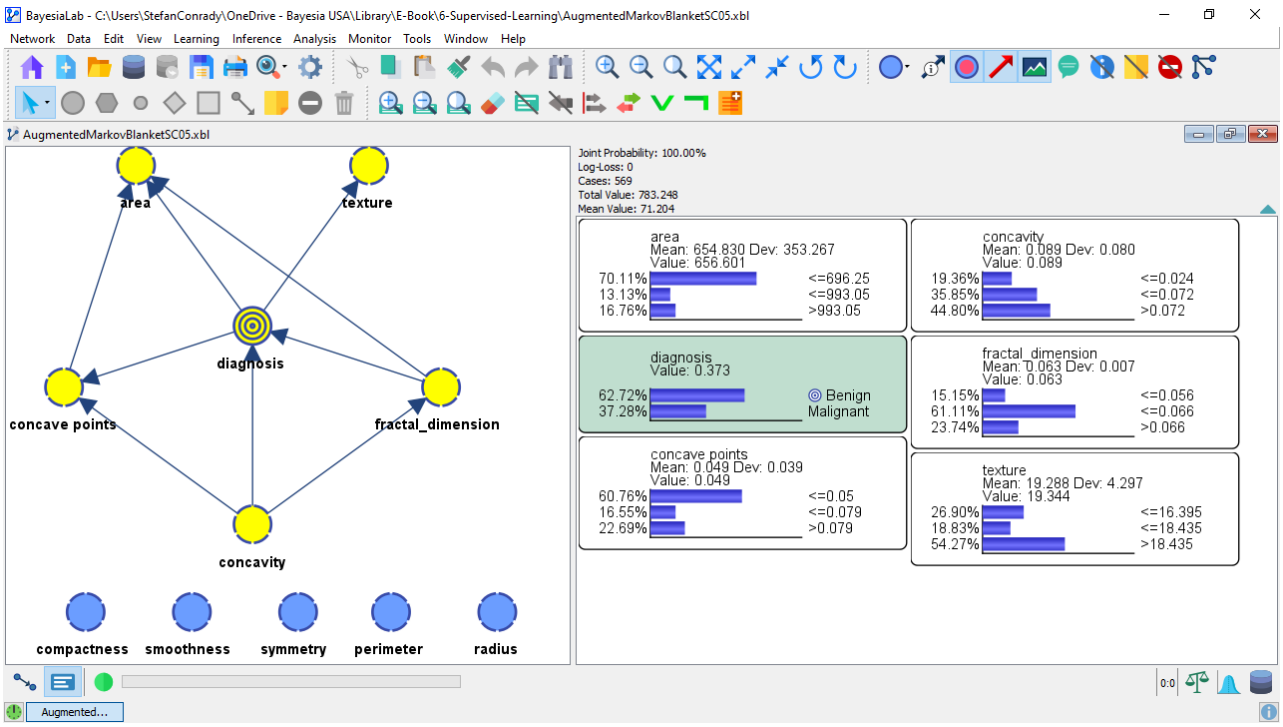
Context

- Early in this chapter, we used the Augmented Markov Blanket algorithm to machine-learn a predictive model for classifying cell samples.
- Subsequently, we optimized the model with the Structural Coefficient Analysis workflow.
- We can now use the validated model for analysis and inference.
- In this and the next two sections of this chapter, we look at different ways of performing inference:
  - Automatic-Evidence Setting
  - Adaptive Questionnaire
  - Target Interpretation Tree

Validation Mode

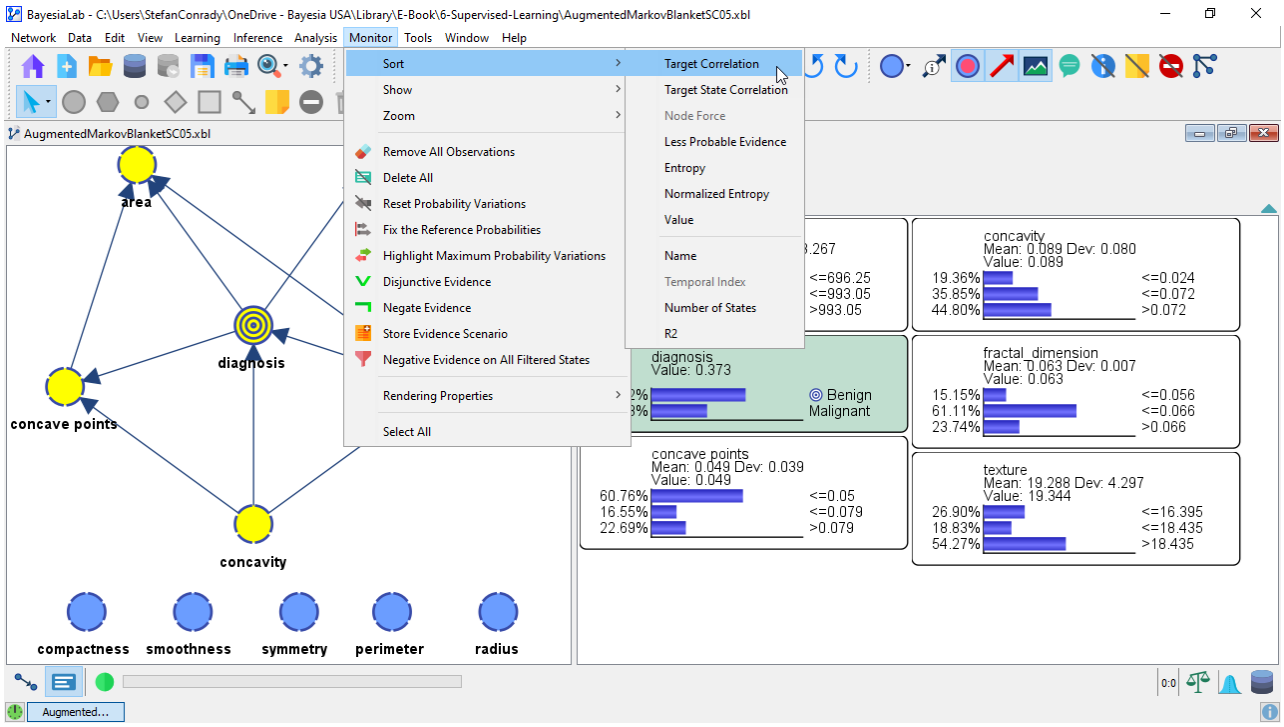
For all types of inference with a Bayesian network model, we need to switch to Validation Mode `F5`.

Before proceeding to Automatic Evidence-Setting, we bring up all the Monitors connected to the Target Node in the Monitor Panel.



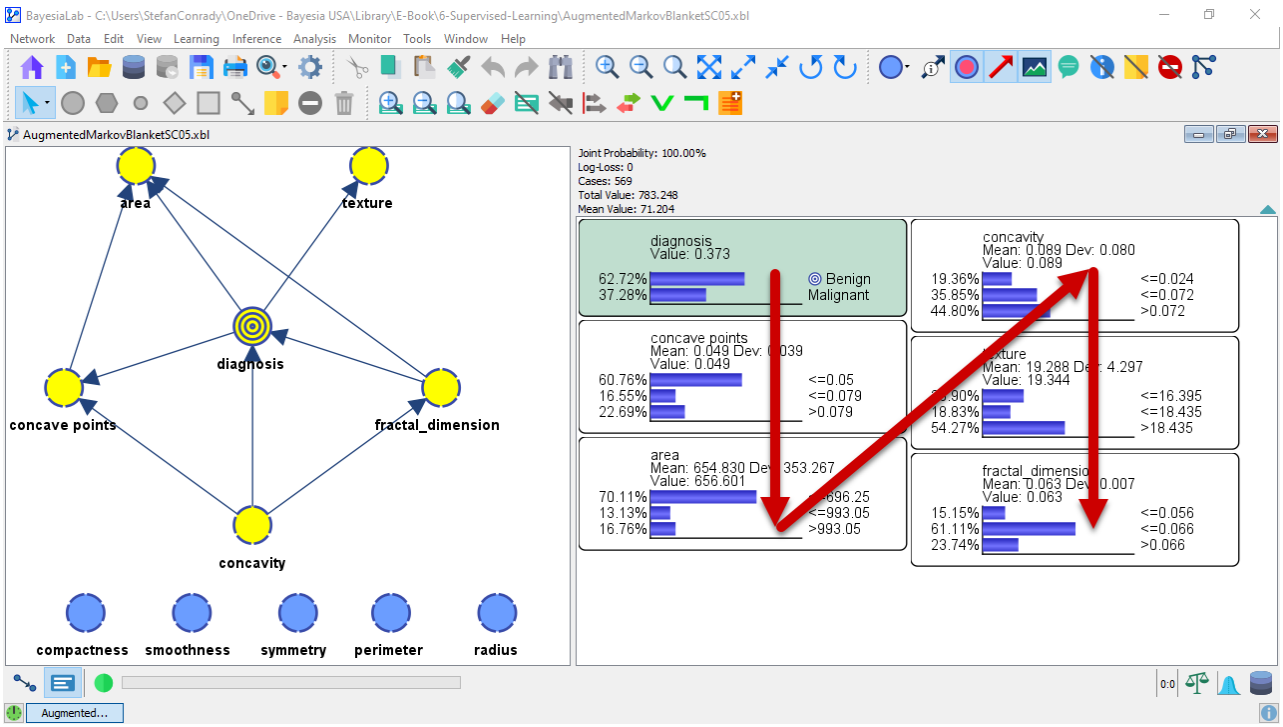
Since we have a Target Node, we can right-click inside the Monitor Panel to select `Sort > Target Correlation` from the Monitor Panel Context Menu.

Alternatively, we can do the same via `Menu > Monitor > Sort > Target Correlation`.



The Monitor of the Target Node is placed first in the Monitor Panel, followed by the other Monitors according to their "correlation" with the Target Node, from highest to lowest.

Note that we use "correlation" not literally in this context. Rather, the sort order of the Monitors is determined by Mutual Information.



Automatic Evidence Setting (Interactive Inference)

Given that we have a predictive model in place, we can use BayesiaLab to review its individual predictions record by record.

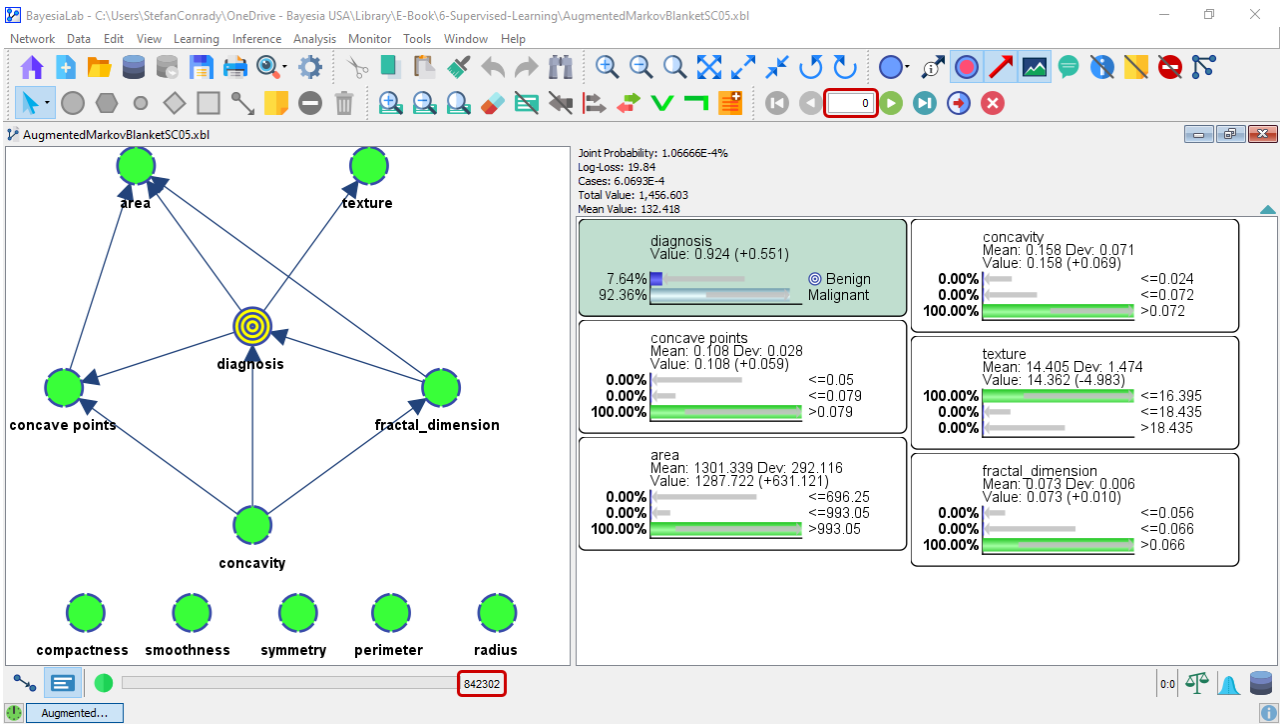
This feature is called Automatic Evidence-Setting, which can be accessed here: `Menu > Inference > Automatic Evidence-Setting`.

> In earlier releases of BayesiaLab, this function was called Interactive Inference.

As an extension of the Main Menu, a Navigation Bar and its record selectors allow us to scroll through all records in the dataset.

Record #0

The first record in the dataset is displayed in the screenshot below as record #0.



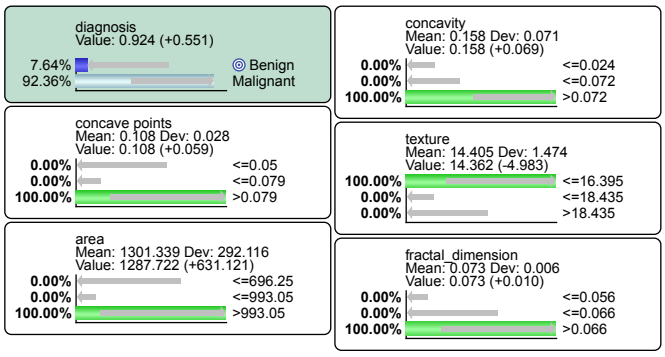
Additionally, the Row Identifier, 842302, is displayed in the Status Bar to the right of the Progress Bar at the bottom of the Graph Window.

The Monitors display the values of the variables in that record, i.e., the set of evidence or observations.

Given these observations, the model predicts a 92.36% probability that the diagnosis is malignant (the Monitor of the Target Node features a green background).

With such a high probability, the prediction diagnosis=malignant is the rational prediction.

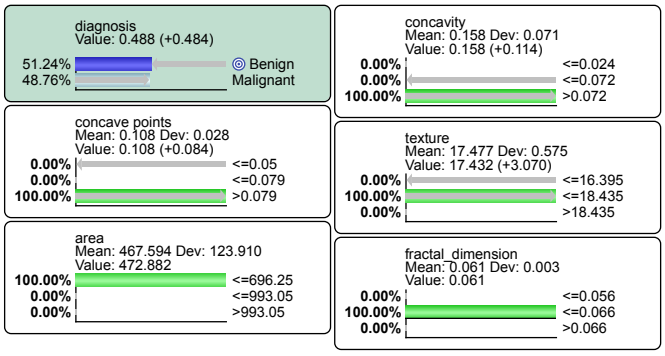
As it turns out, it is indeed the correct prediction for record #0 (Row Identifier 842302). The actual value recorded in the dataset is represented by a light blue bar, meaning diagnosis=malignant was the ground truth in this case.



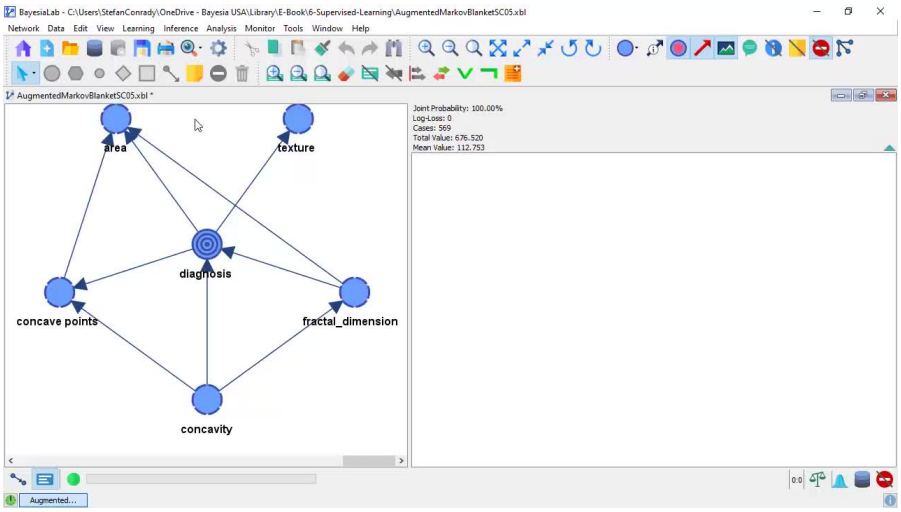
As we know from all the validation steps, the model performs well with an Overall Precision above 90%. Hence, most predictions are clear, just like this one.

Record #138

However, exceptions exist, such as record #138 (Row Identifier 868826). Here, the probability of diagnosis=benign is approximately 51%. Given this probability, the model predicts diagnosis=benign. However, this turns out to be incorrect. Here, the actual observation is diagnosis=malignant, which is again highlighted by a light blue bar.



Workflow Animation



Click to view video

Supervised Learning: Augmented Markov Blanket

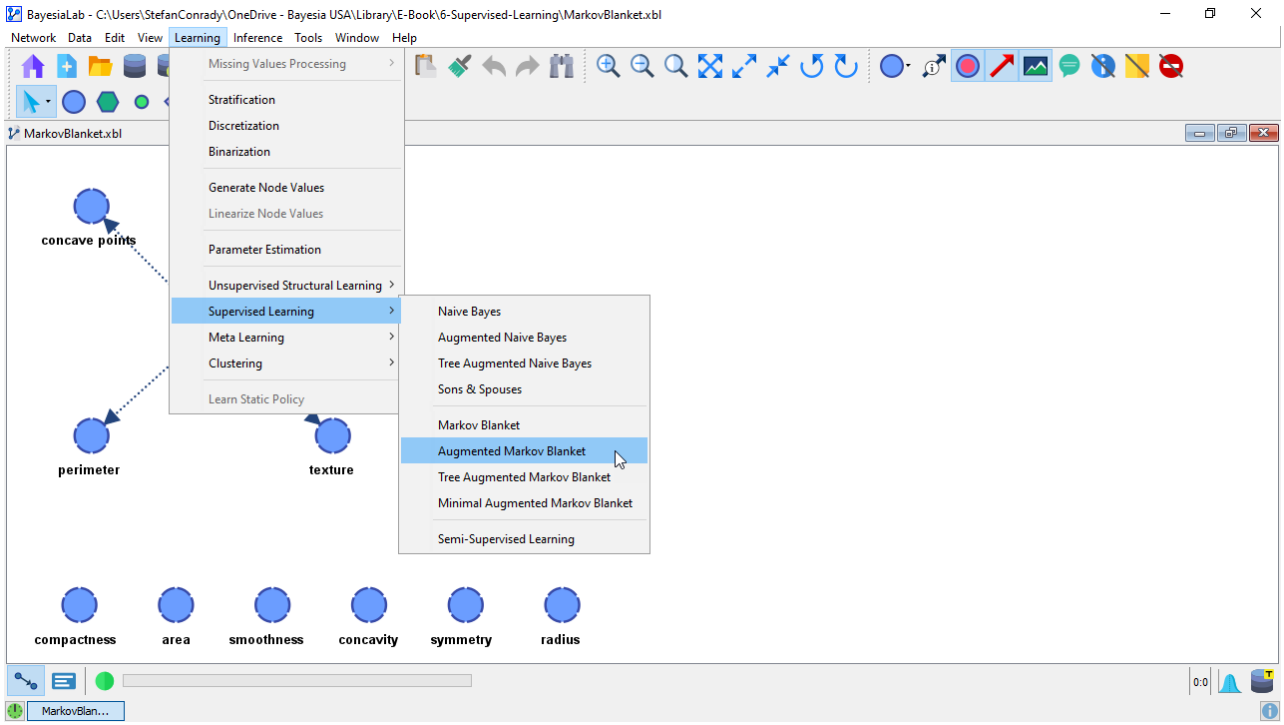
Augmented Markov Blanket

Given the performance of the Markov Blanket algorithm in the previous section (Supervised Learning: Markov Blanket), we are now looking for improvements by considering alternatives within the group of Supervised Learning algorithms.

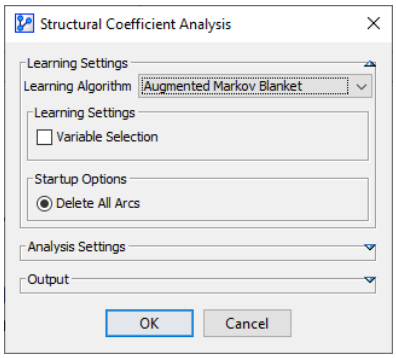
BayesiaLab offers an extension to the Markov Blanket algorithm, namely the Augmented Markov Blanket, which performs an Unsupervised Learning algorithm on the nodes in the Markov Blanket. This relaxes the constraint of requiring orthogonal child nodes. Thus, it helps identify any influence paths between the predictor variables and potentially improves the predictive performance. Adding such arcs would be similar to automatically creating interaction terms in a regression analysis.

After returning to the Modeling Mode F4 we start this learning algorithm via `Menu > Learning > Supervised Learning > Augmented Markov Blanket`.

Note that we are using the original Learning/Test Sets split again. The T symbol tagged onto the database icon reminds us that the Learning Set and Test Set split is in place.



As expected, the resulting network is slightly more complex than the Markov Blanket.

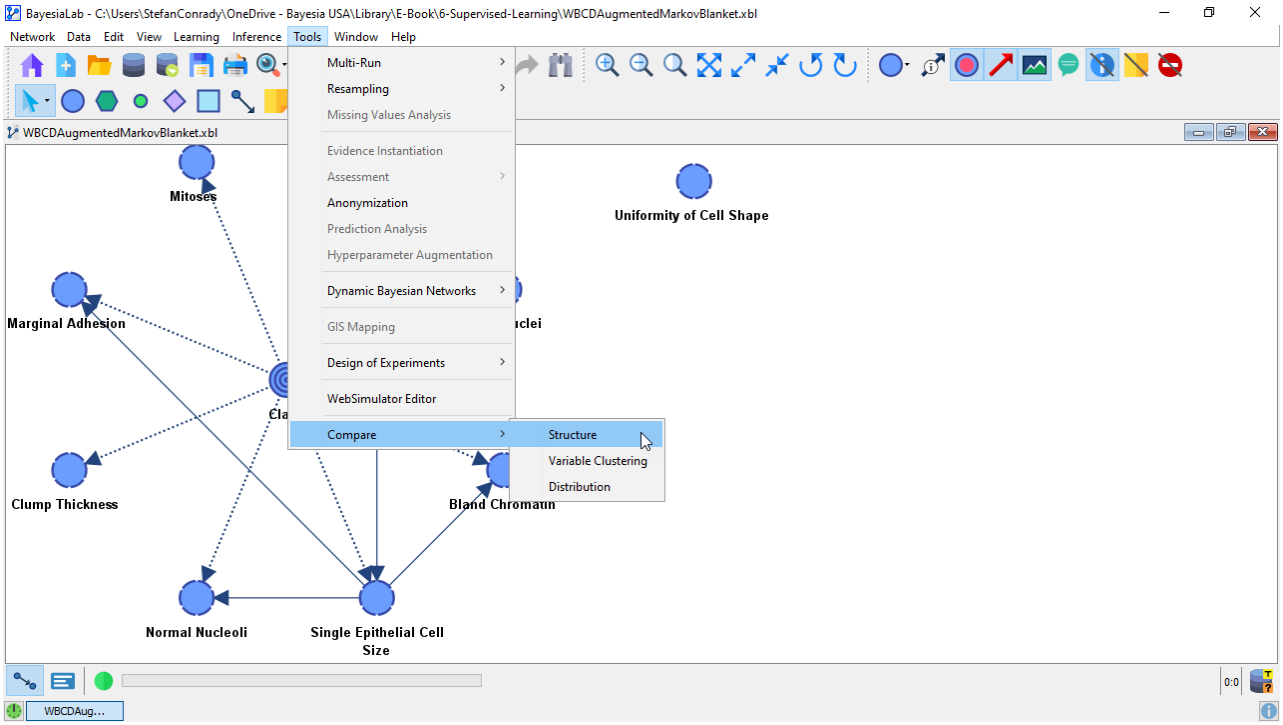


Structure Comparison

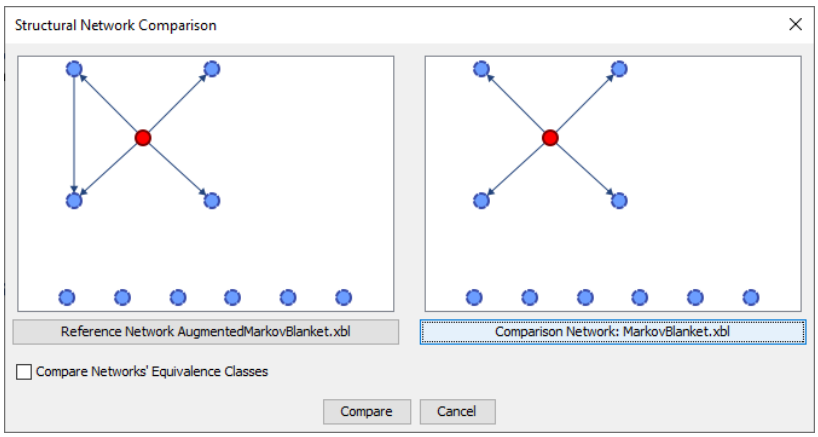
BayesiaLab offers a tool for formally comparing network structures, which we can apply to the Augmented Markov Blanket we just learned and the previously learned Markov Blanket.

We can use `Menu > Tools > Compare > Structure` to highlight the differences between both networks.

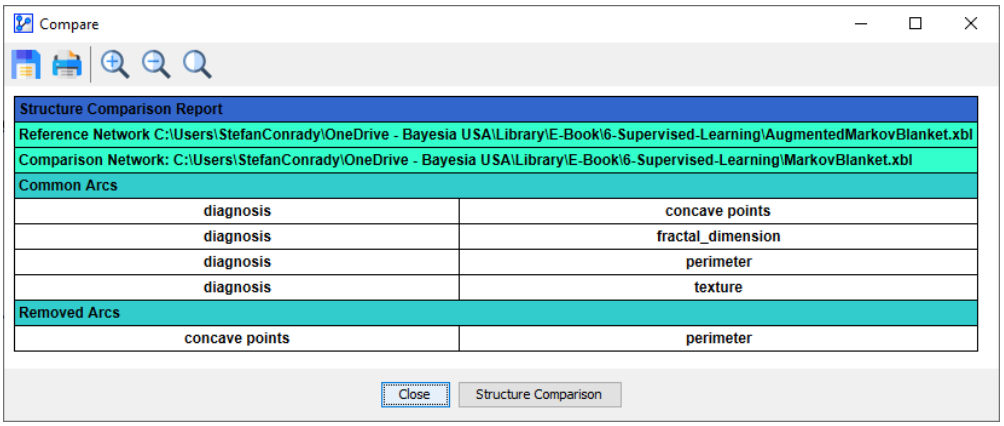
Given that the addition of two arcs is immediately visible, this function may appear overkill for our example. However, in more complex situations, Structure Comparison can be rather helpful.



By default, the current network appears as the Reference Network, and for the Comparison Network, we select the previously learned Markov Blanket.

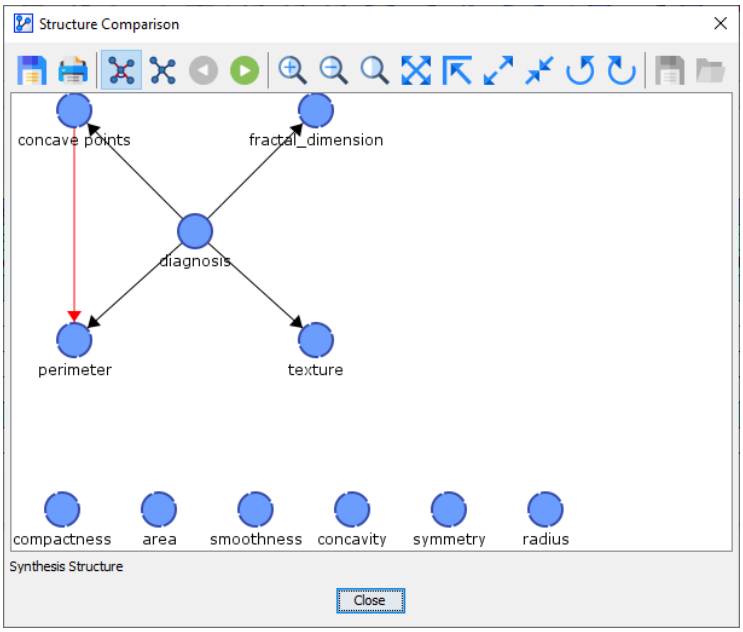


Clicking `Compare` brings up the Structure Comparison Report.



This report provides a list of arcs common to both networks and another list of those removed in the Comparison Network.

Clicking `Structure Comparison` shows a Synthesis Structure that visualizes these differences.



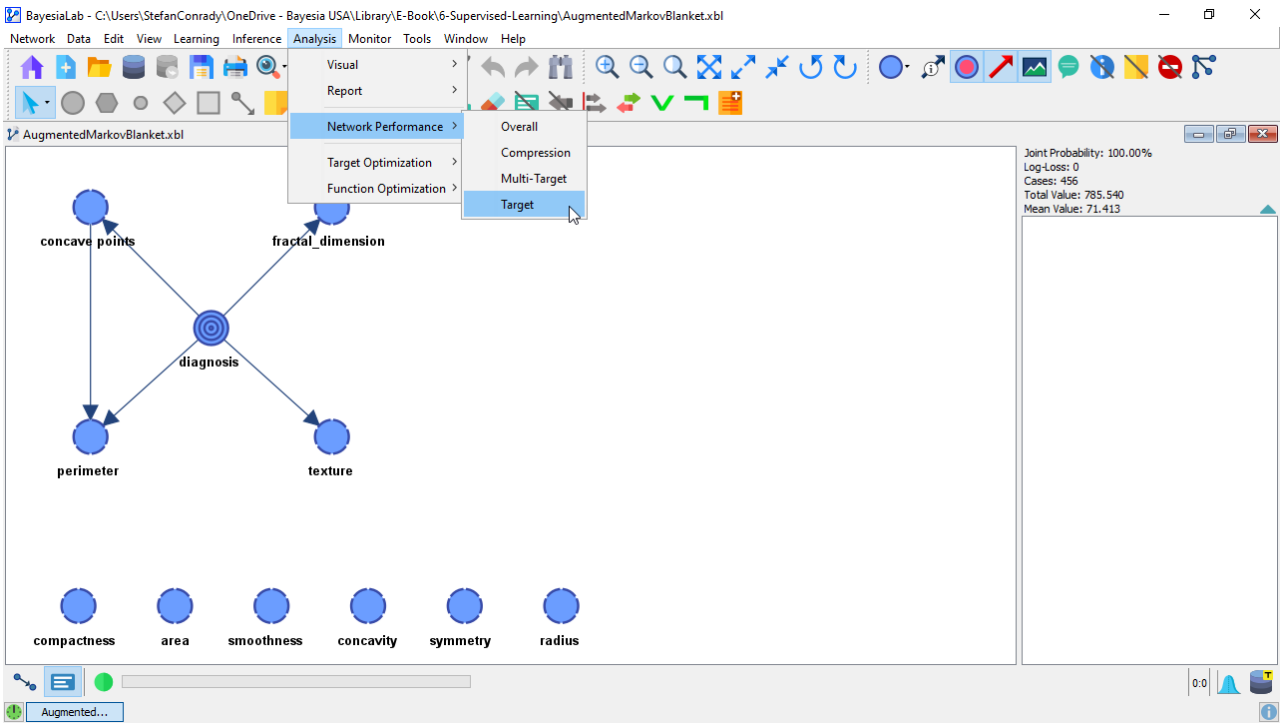
The arcs that exist in the Reference Structure, i.e., Augmented Markov Blanket, but do not exist in the Comparison Structure, i.e., the Markov Blanket, are highlighted in red.

Please see Direct Structural Network Comparison for a much more detailed explanation of this tool.

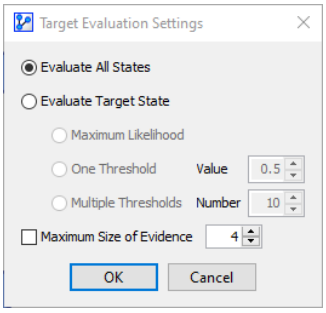
Performance Analysis

Given that the Augmented Markov Blanket algorithm has only added a single additional arc to the network, compared to what the Markov Blanket algorithm produced, we may not expect a dramatic difference in predictive performance.

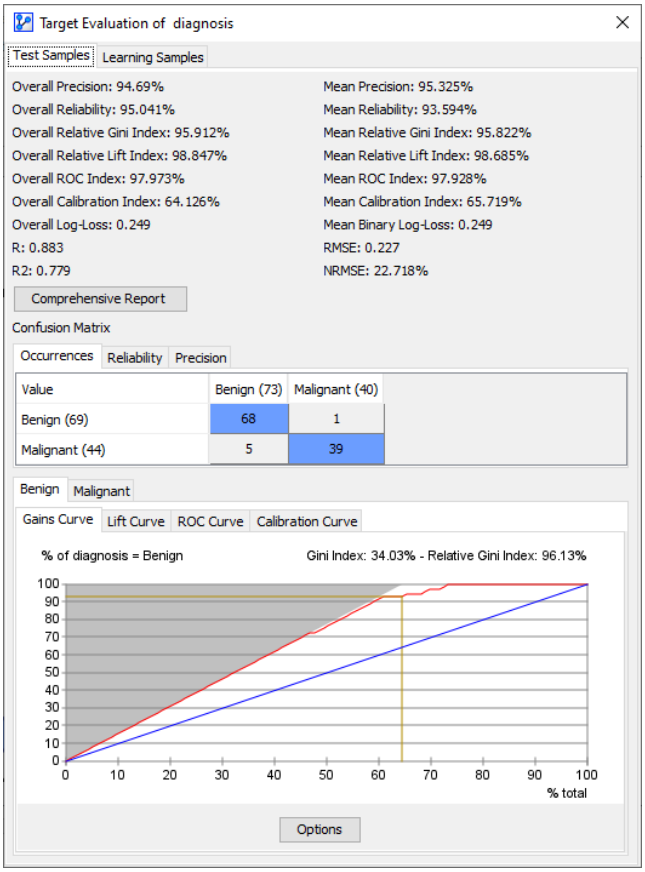
However, any improvements in terms of reducing the False Negative would be welcome. So, we run the Network Performance Analysis again: `Menu > Analysis > Network Performance > Target`.



As the analysis starts, BayesiaLab prompts us to specify the Target Evaluation Setting. Again, we select `Evaluate All States` and proceed.



Using the previously defined Test Set for evaluating our model, we obtain the performance report:



Now, we can compare the new performance metrics of the Augmented Markov Blanket with the ones previously obtained with the Markov Blanket.

Evaluation Against Test Set

Markov Blanket	Augmented Markov Blanket
Overall Precision: 90.224%	Overall Precision: 94.669%
False Negative Rate: 15%	False Negative Rate: 2.5%

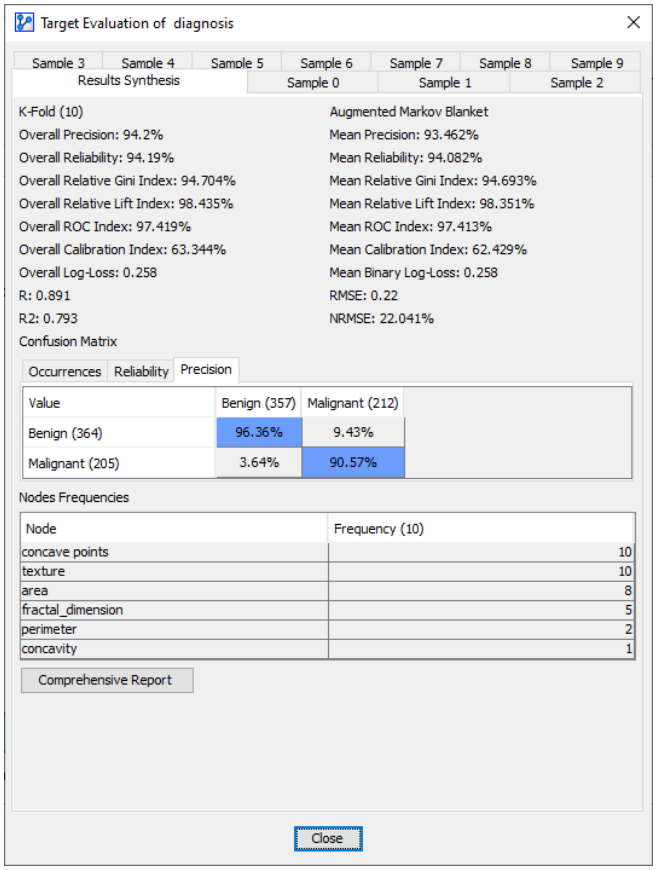
Most notable is the change in False Negatives, which drops from 5 to 1, i.e., a reduction from 15% to 2.5% in the False Negative Rate.

If this performance were to hold, it could turn this model from moderately useful to a valuable diagnostic tool.

Cross-Validation

Recognizing the potential of the Augmented Markov Blanket algorithm, we proceed to the K-Folds Cross-Validation: `Menu > Tools > Cross-Validation > Targeted Evaluation > K-Fold`.

The steps are identical to what we did for the Markov Blanket, so we move straight to the report.



As it turns out, the Cross-Validation Report does not confirm the excellent False Negative Rate that the evaluation with regard to the Test Set suggested. Nevertheless, comparing the Cross-Validation results, the Augmented Markov Blanket algorithm delivers an improvement over the Markov Blanket.

Cross-Validation Result Synthesis

Markov Blanket	Augmented Markov Blanket
Overall Precision: 92.97%	Overall Precision: 94.2%
False Negative Rate: 11.32%	False Negative Rate: 9.43%

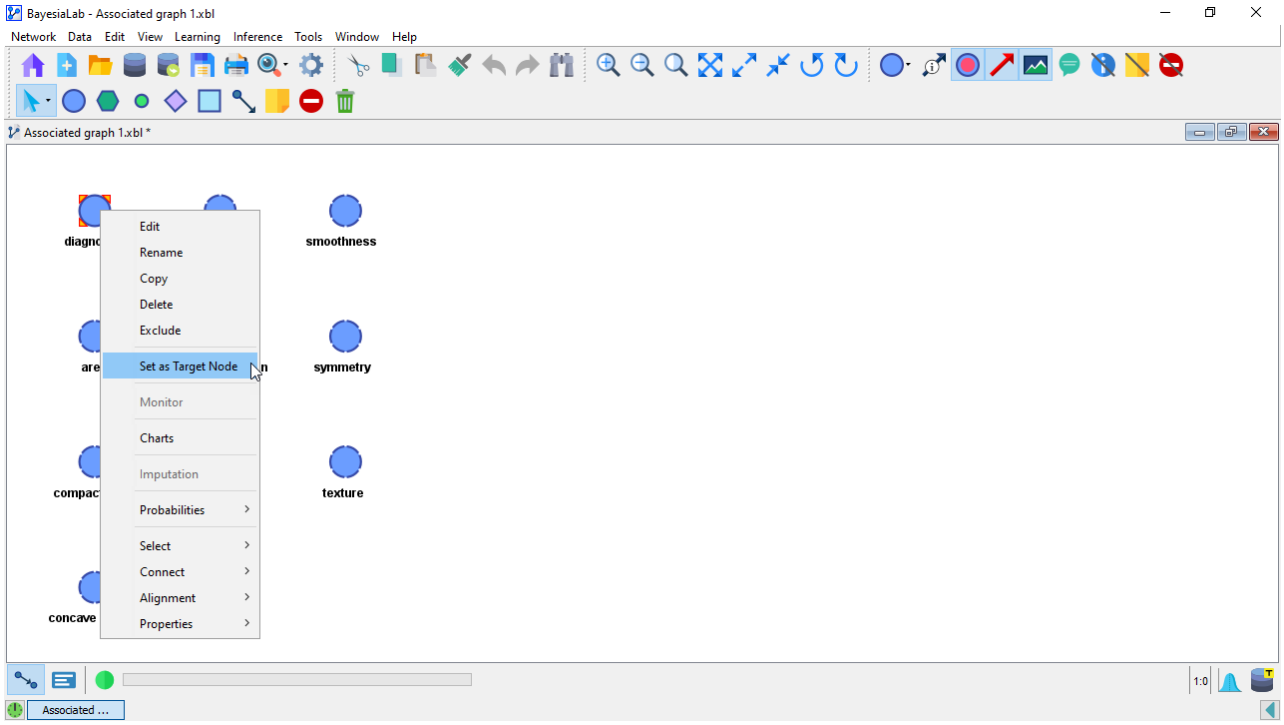
With the apparent advantage of the Augmented Markov Blanket model, we will now try to fine-tune this model further in pursuit of a performance gain. In the next section, Structural Coefficient Analysis, we explore how adjusting the Structural Coefficient can bring us closer to the performance limits of the model.

Supervised Learning: Markov Blanket

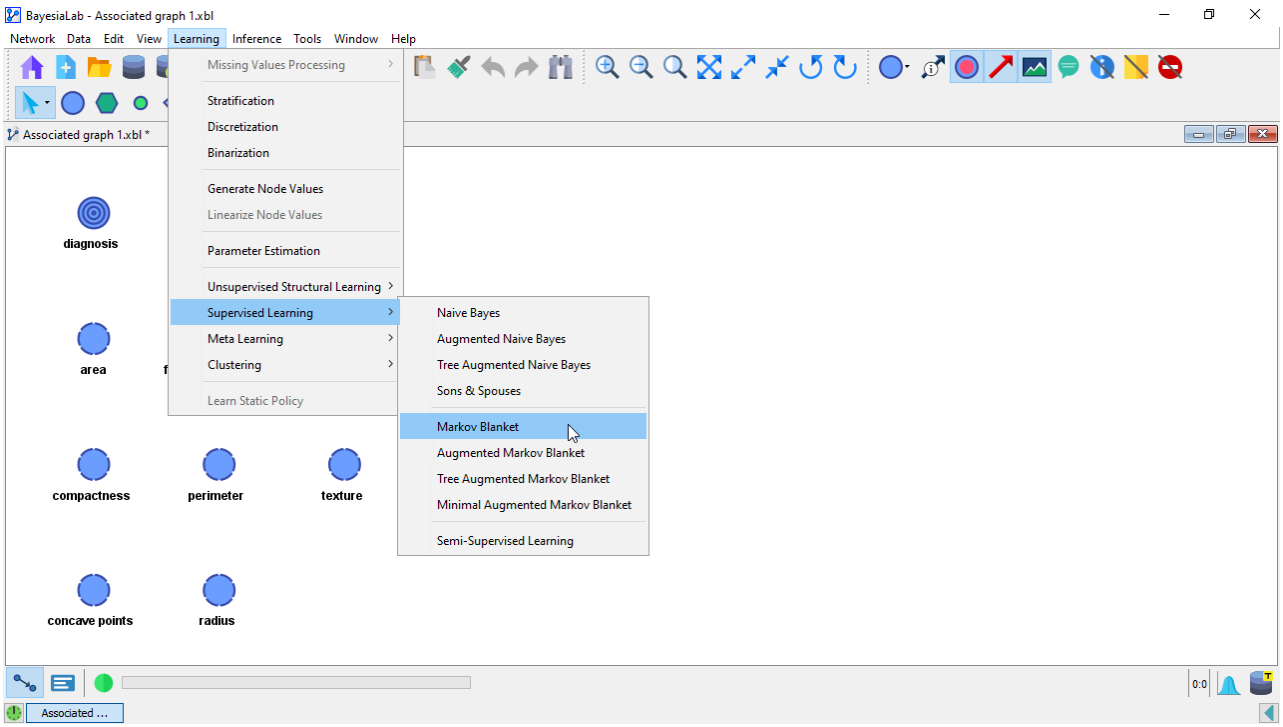
Markov Blanket

Given our objective of predicting the state of the variable diagnosis, i.e., Benign versus Malignant, we define diagnosis as the Target Node.

We need to specify this explicitly so that the Supervised Learning algorithm can focus on the characterization of the Target Node rather than on a representation of the entire Joint Probability Distribution (JPD) of the learning set.

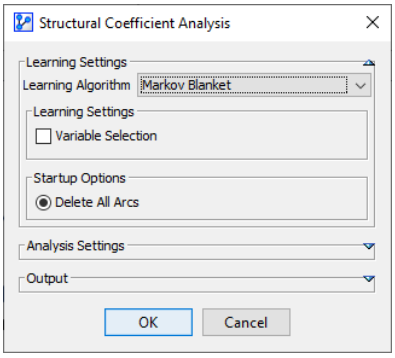


Upon defining the Target Node, all Supervised Learning algorithms become available under `Menu > Learning > Supervised Learning`.



Markov Blanket Definition

Upon learning the Markov Blanket for diagnosis and after having applied the Automatic Layout (shortcut P), the resulting Bayesian network appears as follows:



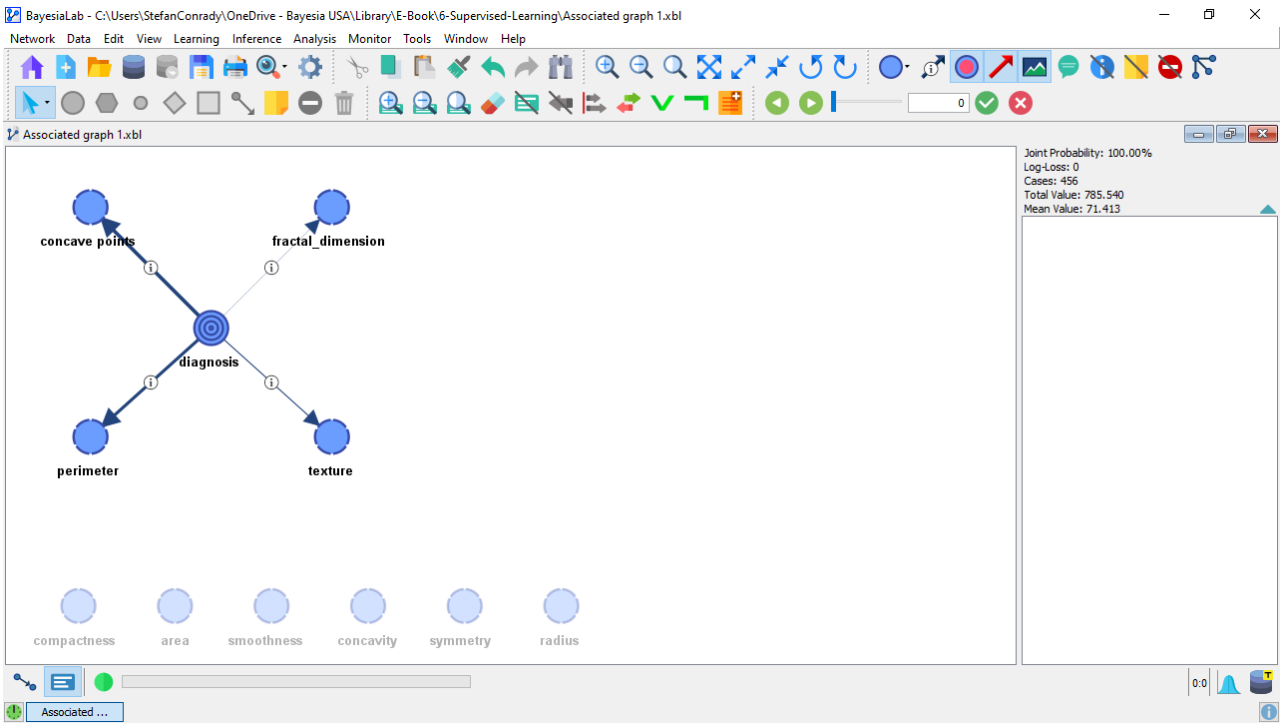
We can see that the obtained network is a Naive structure on a subset of nodes.

This means that *diagnosis* has a direct probabilistic relationship with concave_points, fractal_dimension, texture, and perimeter.

All other nodes remain unconnected. The lack of their connections with the Target Node implies that these nodes are independent of the Target Node, given the nodes in the Markov Blanket.

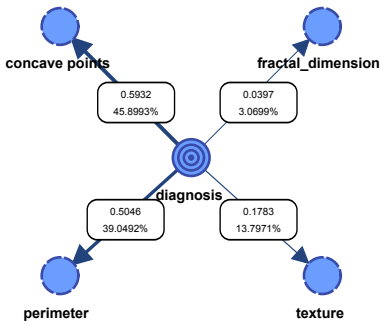
Beyond distinguishing between predictors (connected nodes) and non-predictors (disconnected nodes), we can further examine the relationship versus the Target Node *diagnosis* by highlighting the Mutual Information of the arcs connecting the nodes.

This function is accessible in Validation Mode F5 by selecting `Menu > Analysis > Visual > Overall > Arc > Mutual Information`.



Each arc's thickness is now proportional to the Mutual Information of the nodes it connects. Furthermore, the ⓘ icon indicates that additional information, i.e., the Arc Comments, is available to be displayed.

So, we select `Menu > View > Show Arc Comments`. Alternatively, clicking the `Show Arc Comment` button in the Toolbar achieves the same . This allows us to examine the Mutual Information between all nodes and the Target Node diagnosis, which enables us to gauge the relative importance of the nodes.

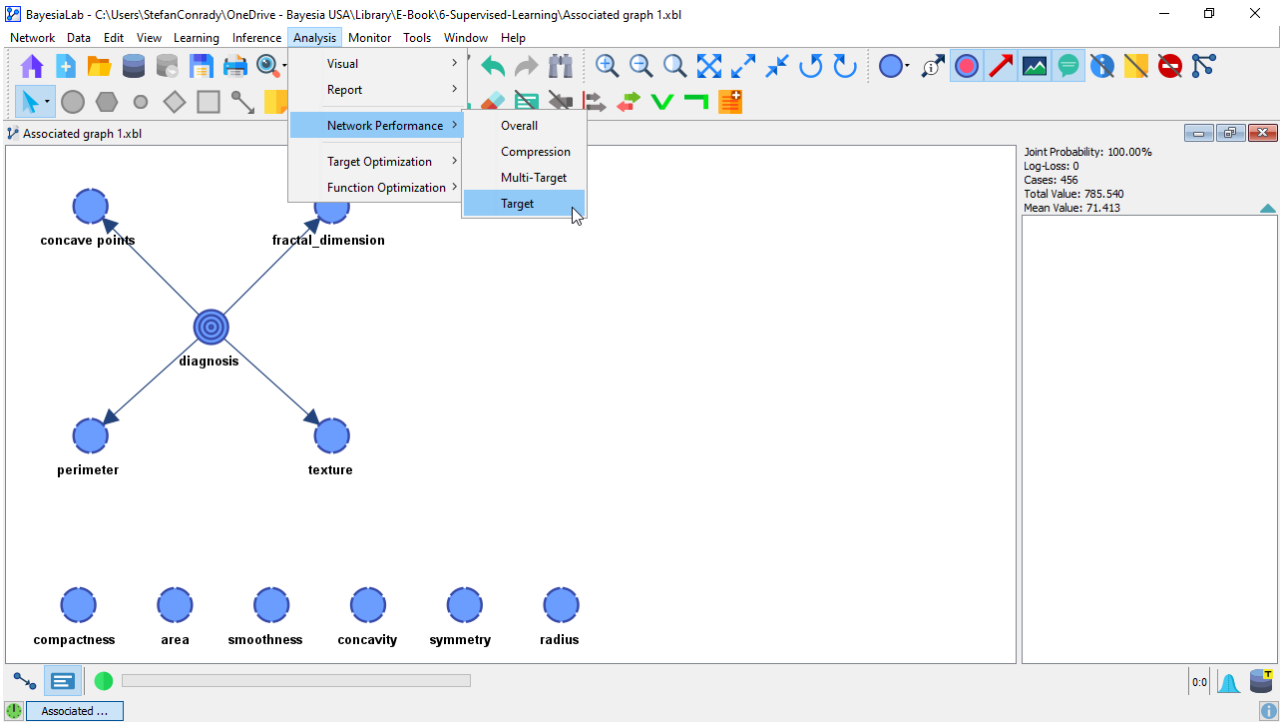


The top value shown in the box attached to each arc is the absolute value of the Mutual Information.

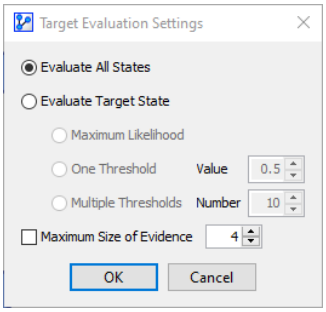
Below, the percentage refers to the Symmetric Normalized Mutual Information.

Performance Analysis

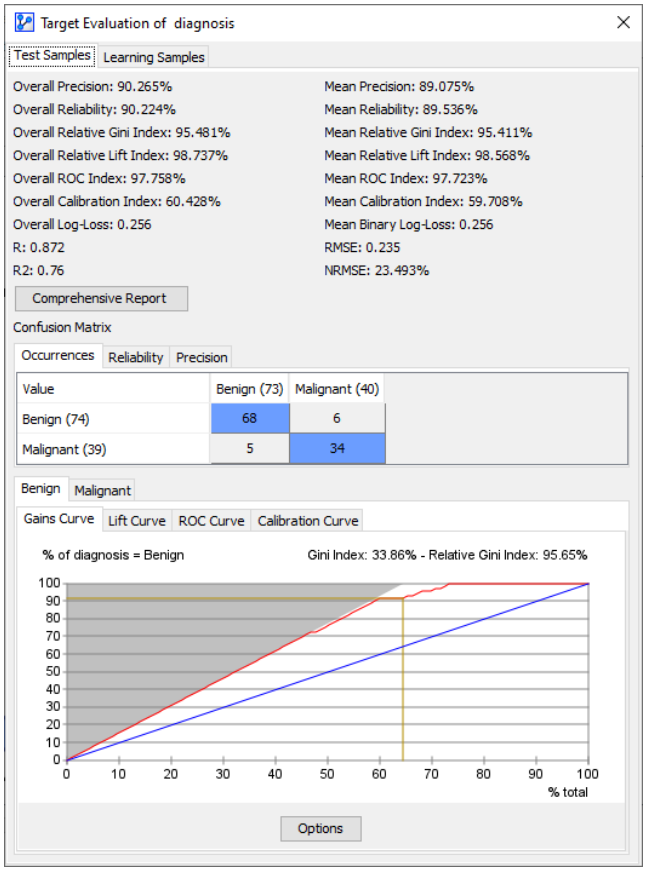
As we are not equipped with specific domain knowledge about the nodes, we will not further interpret these relationships but rather run an initial test regarding the Network Performance. We want to know how well this Markov Blanket model can predict the states of the diagnosis variable, i.e., Benign versus Malignant. This test is available via `Menu > Analysis > Network Performance > Target`.



As the analysis starts, BayesiaLab prompts us to specify the Target Evaluation Setting. In the given context, we select `Evaluate All States` and proceed.

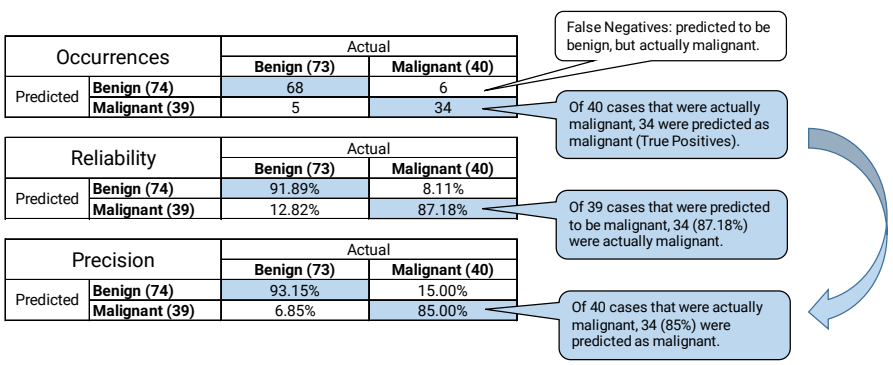


Using the previously defined Test Set for evaluating our model, we obtain the initial performance results, including metrics such as Total Precision, R, R2, etc.



In the context of this example, the table in the center of the report, the so-called Confusion Matrix, is of special interest.

The Confusion Matrix features three tabs, Occurrences, Reliability, and Precision, which are illustrated below:



Of the 40 Malignant cases in the Test Set, 34 were identified correctly (True Positive Rate: 85%), and 6 were incorrectly predicted (False Negative Rate: 15%).

Of the 73 Benign cases in the Test Set, 68 were correctly identified as Benign (True Negative Rate: 93.15%), and 5 were incorrectly identified as Malignant (False Positive Rate: 6.85%).

The Overall Precision, which is reported at the top of the report window, is computed as the total number of correct predictions (True Positives + True Negatives) divided by the total number of cases in the Test Set, i.e., (68+34)÷113=90.265%.

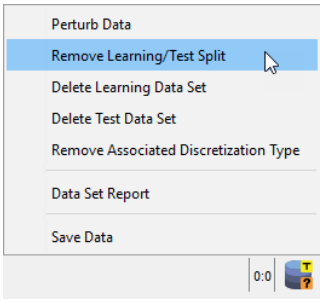
K-Folds Cross-Validation

An Overall Precision of around 90% is encouraging, but we must remember that we randomly selected the Test Set.

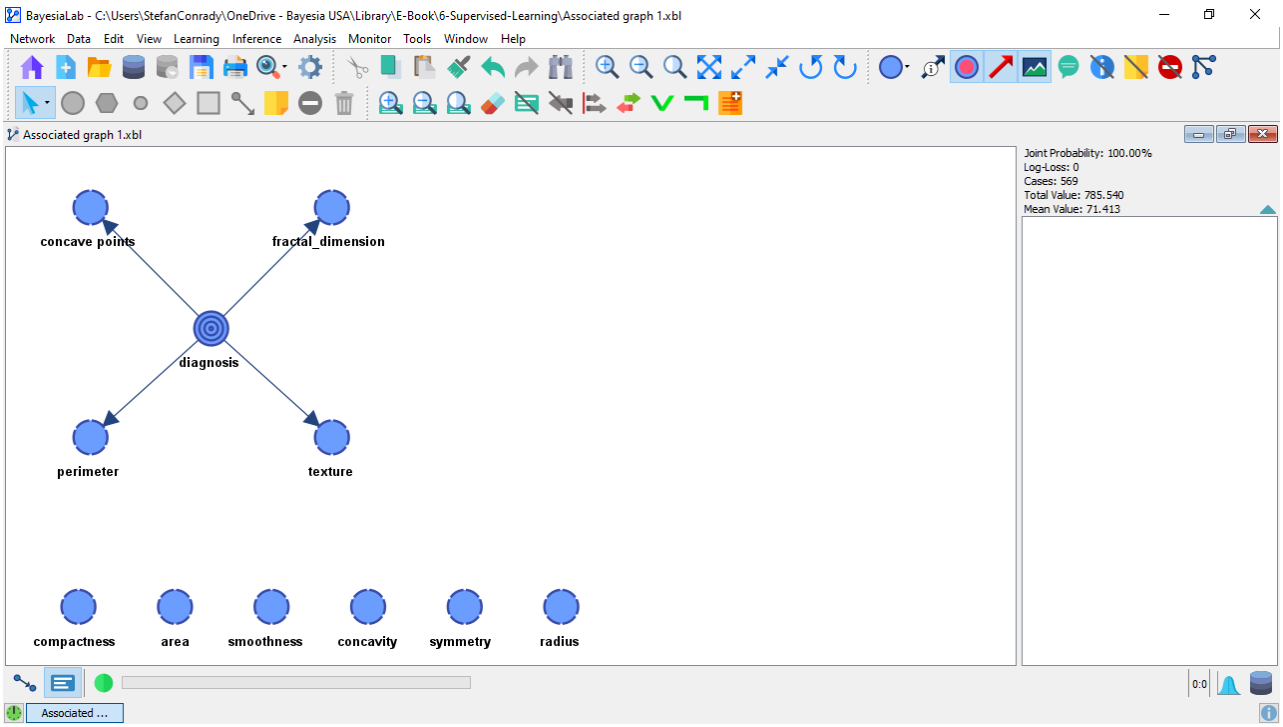
To mitigate any sampling artifacts that may occur in such a one-off Test Set, we can systematically learn networks on a series of different subsets and then aggregate the test results.

For this purpose, we perform a K-Folds Cross-Validation, which will iteratively select K different Learning Sets and Test Sets and then learn the corresponding networks and test their respective performance.

With this approach, we need to remove the original Learning Set and Test Set split. Right-clicking on the database icon in the lower right corner of the Graph Window brings up a menu. Here, we select `Remove Learning/Test Split`.

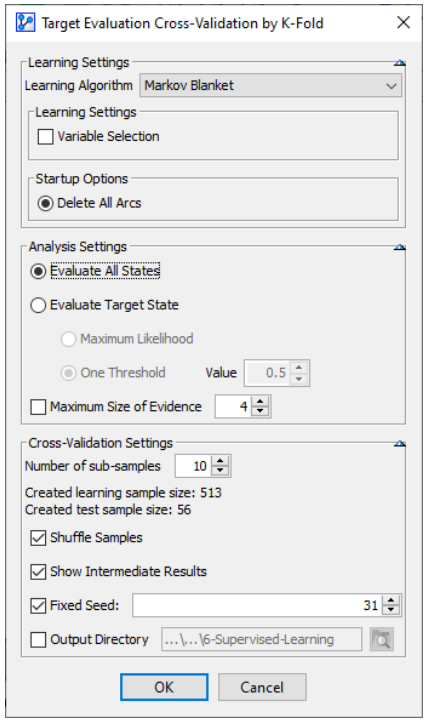


Then, K-Folds Cross-Validation can be started via `Menu > Tools > Resampling > Target Evaluation > K-Fold`:



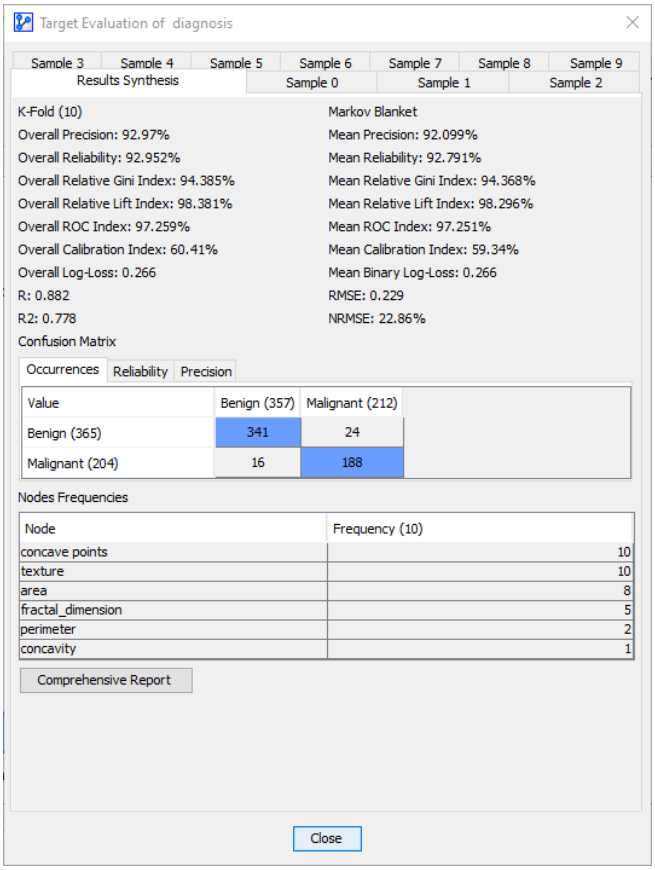
We use the same learning algorithm as before, i.e., the Markov Blanket, and choose K=10 as the number of sub-samples to be analyzed.

Of the total dataset of 569 cases, each of the ten iterations (folds) will create a Test Set of 56 randomly drawn samples and use the remaining 630 as the Learning Set. This means that BayesiaLab learns one network per Learning Set and then tests the performance on the respective Test Set. It is important to ensure that the Shuffle Samples option is checked.

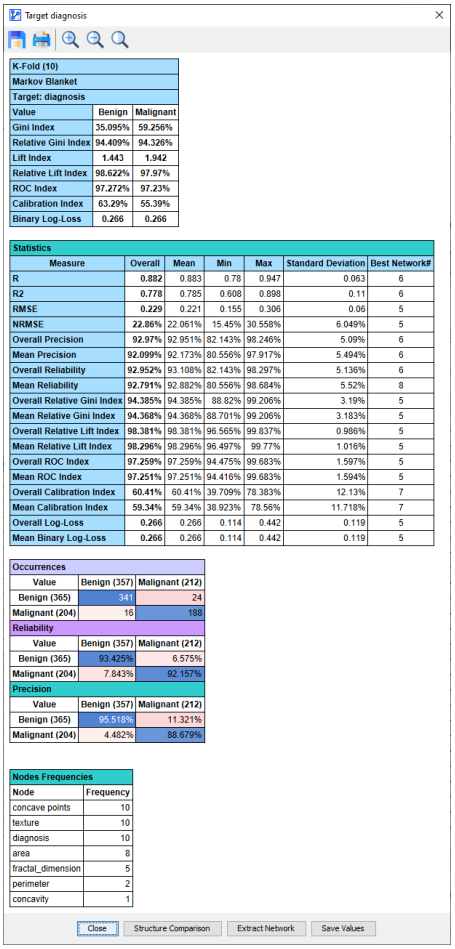


The summary, including the synthesized results, is shown below. These results confirm the good performance of this model.

The Total Precision is 92.97%, with a False Negative Rate of 11.32%. This means that 24 of the 212 Malignant cases were incorrectly predicted as Benign.



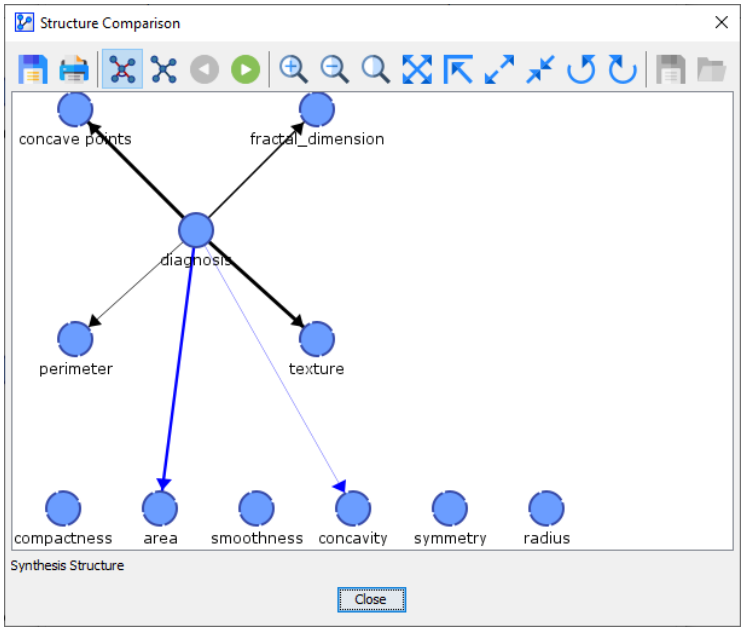
Clicking `Comprehensive Report` produces a summary with additional analysis options.



Interpreting the Cross-Validation Results

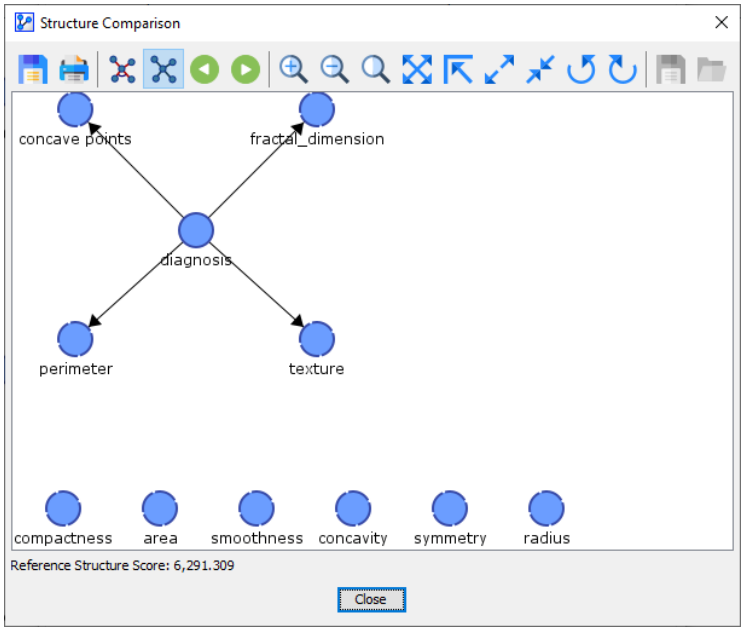
It is helpful to click the `Network Comparison` button to understand what exactly is happening during the K-Folds Cross-Validation. It brings up a Synthesis Structure of all the networks learned during the K-Folds Cross-Validation.

Black arcs in the Synthesis Structure above indicate that these arcs were present in the Reference Structure (below), i.e., the network that was learned on the basis of the original Learning Set.

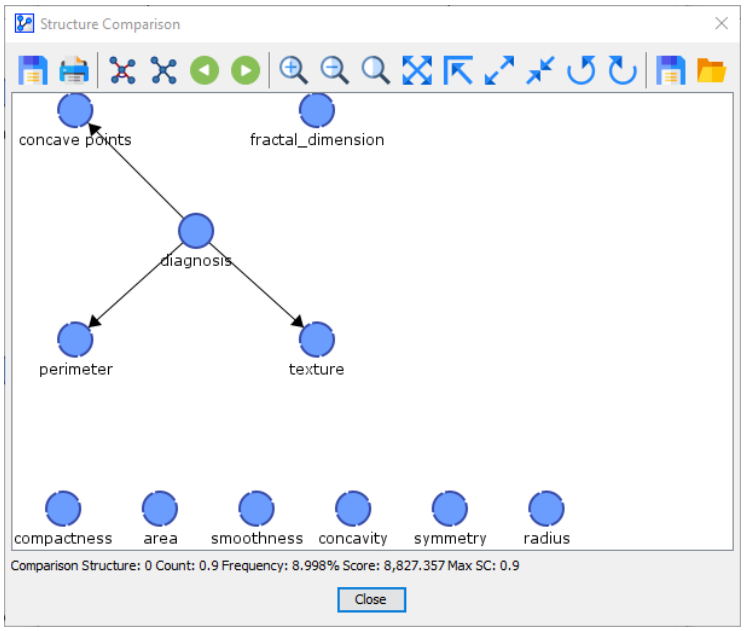


The thickness of the arcs in the Synthesis Structure reflects how often these links were found in the course of the K-Folds Cross-Validation. The blue-colored arc indicates that the link was only found in some folds but that it was not part of the Reference Structure. The thickness of the blue arc is also proportional to the number of folds in which that arc was added.

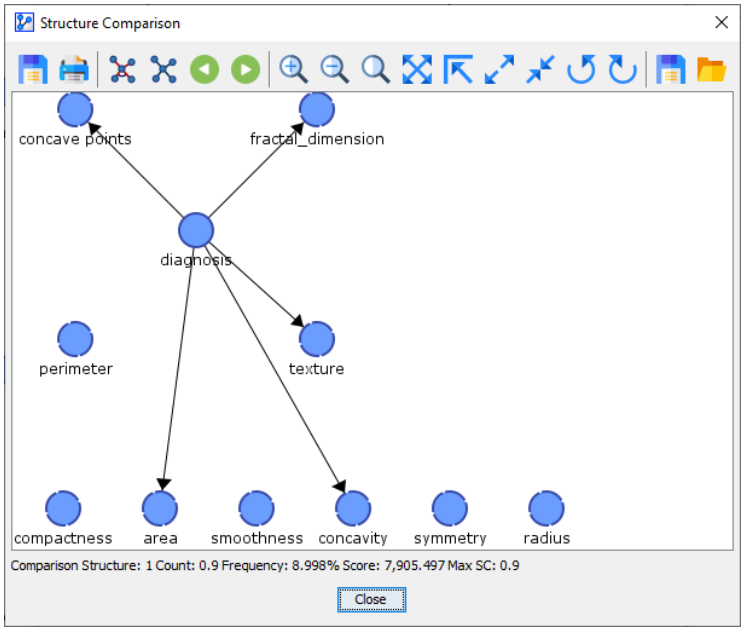
We can scroll through all the networks discovered during the K-Folds Cross-Validation using the record selector icons . The first structure after the Synthesis Structure is the Reference Structure, which was the current network when we started the K-Folds Cross-Validation.



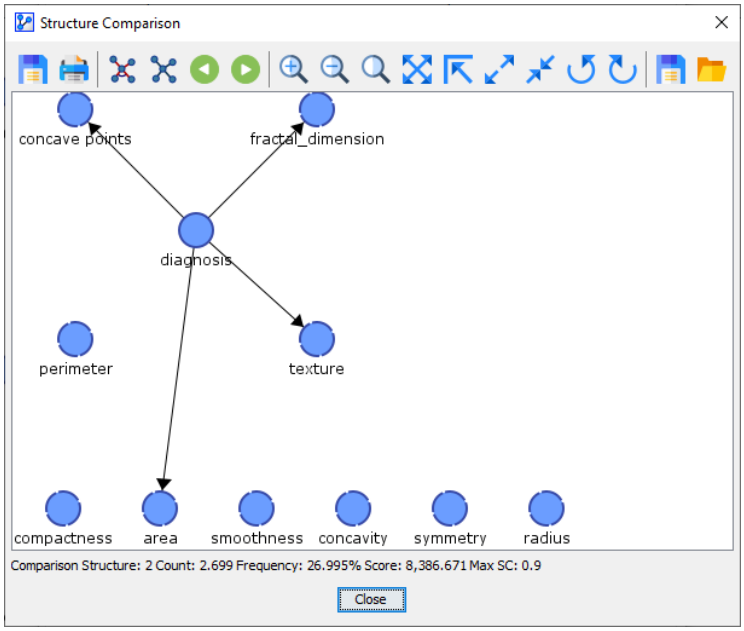
After the Reference Network, we arrive at Comparison Structure 0. This network structure was learned in 1 out of 10 folds.



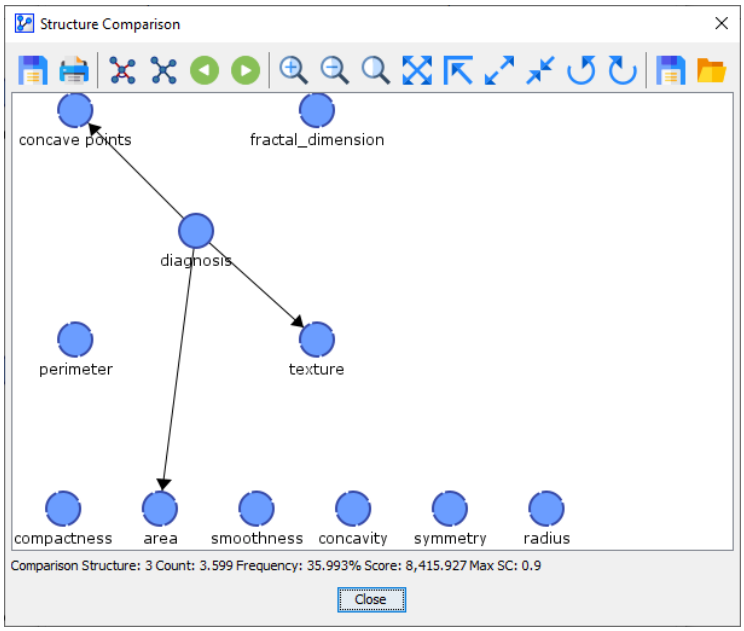
Comparison Network 1 was found 1 out of 10 times.



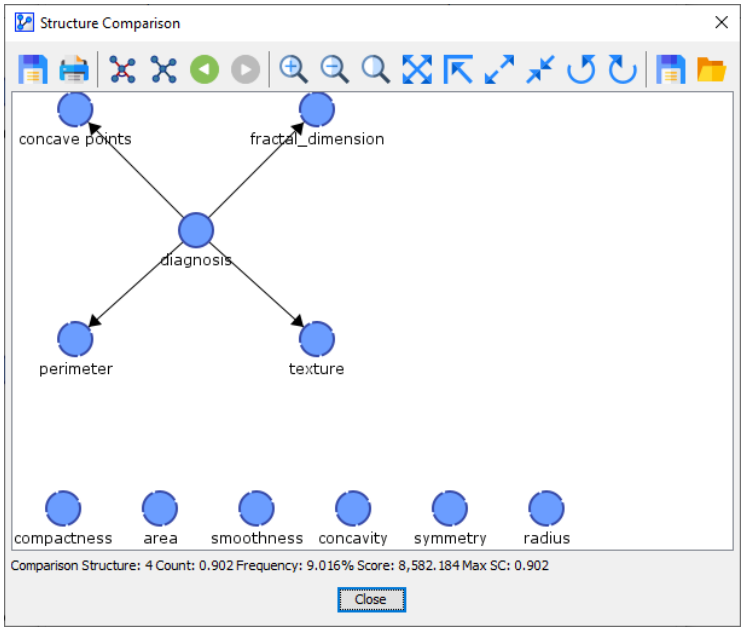
Comparison Network 2 was found 3 out of 10 times.



Comparison Network 3 was found 4 out of 10 times.



Comparison Network 4 was found 1 out of 10 times.



So, the first network we learned from the original Learning Set, the Reference Structure, was only found in 1 of the 10 networks learned during the 10-Fold Cross-Validation.

Given the relatively small sample size of the original Learning Set (431), it is unsurprising that larger sample sizes, i.e., 512 records in each fold of the 10-Fold Cross-Validation, would lead to alternative structures.

Markov Blanket Performance Summary

Performing the K-Fold Cross-Validation shows that the Overall Precision of a Markov Blanket model can approach 93%. However, a False Negative Rate of over 10% may prevent such a model from being useful for clinical purposes. In the context of diagnosing cancer, a False Negative means missing a malignant case.

As a result, we proceed to another algorithm to evaluate its potential for improved diagnostic performance: Supervised Learning: Augmented Markov Blanket.

Supervised Learning: Structural Coefficient Analysis

Up to this point, the differences in model structure and the corresponding performance were a result of the learning algorithm, i.e., Markov Blanket vs. Augmented Markov Blanket.

Now we explore how different levels of network complexity could potentially improve the Augmented Markov Blanket model. In other words, could a more complex network provide better performance without risking over-fitting?

Structural Coefficient

To modify a network's complexity, we now introduce the Structural Coefficient α.

Throughout this chapter, we abbreviate "Structural Coefficient α" with "SC."

This parameter allows changing the internal number of observations and, thus, determines a kind of "significance threshold" for network learning. Consequently, it influences the degree of complexity of the induced networks. The internal number of observations is defined as:

$$N' = \frac{N}{SC}$$

where N is the number of samples in the dataset.

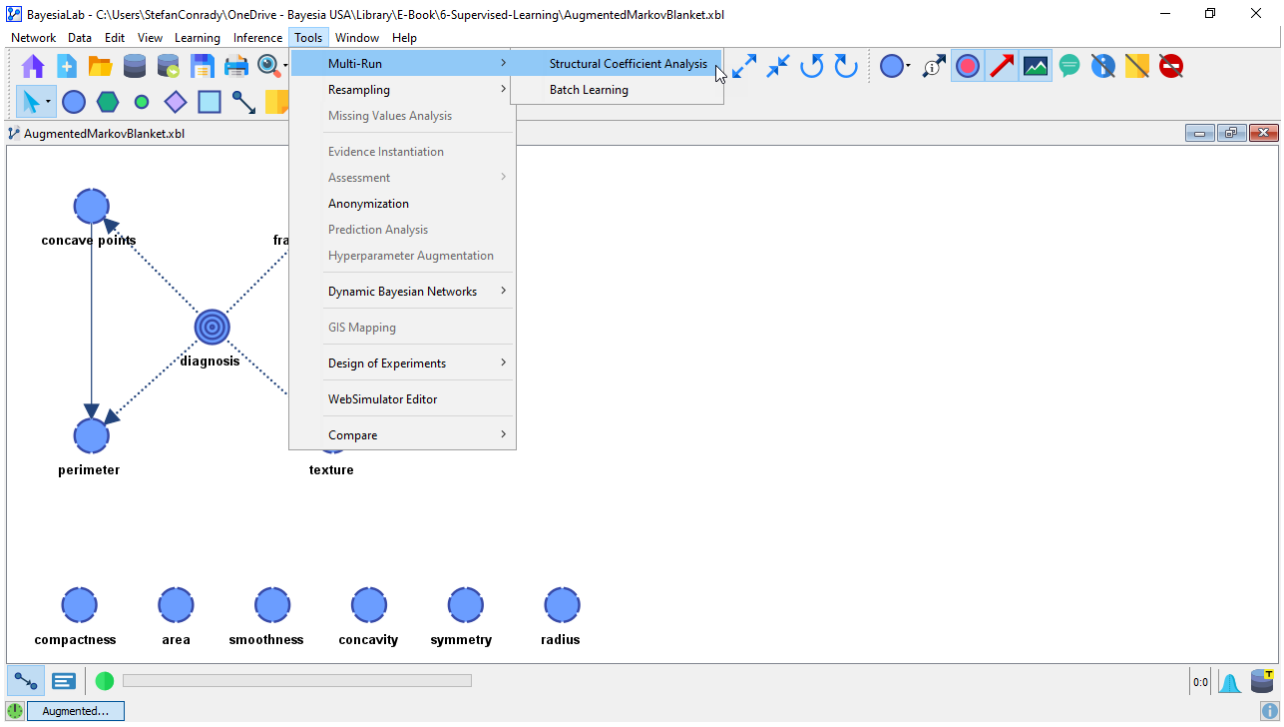
By default, SC is set to 1, which reliably prevents the learning algorithms from overfitting the model to the data. However, in studies with relatively few observations, the analyst's judgment is needed regarding whether a downward adjustment of this parameter can be justified. Reducing SC means increasing N', which is like increasing the number of observations in the dataset via resampling.

On the other hand, increasing SC beyond 1 means reducing N', which can help manage the complexity of networks learned from large datasets. Conceptually, reducing N' is equivalent to sampling the dataset.

Structural Coefficient Analysis

We now perform a Structural Coefficient Analysis on the basis of the Augmented Markov Blanket, which generates several metrics that help to trade off between complexity and fit: `Menu > Tools > Multi-Run > Structural Coefficient Analysis`.

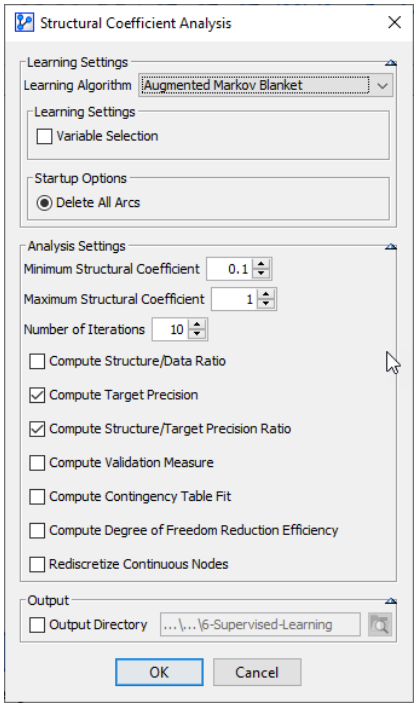
Note that we use the original Learning/Test Set split again, which allows us to directly compare the in-sample and out-of-sample predictive performance as a function of varying SC levels.



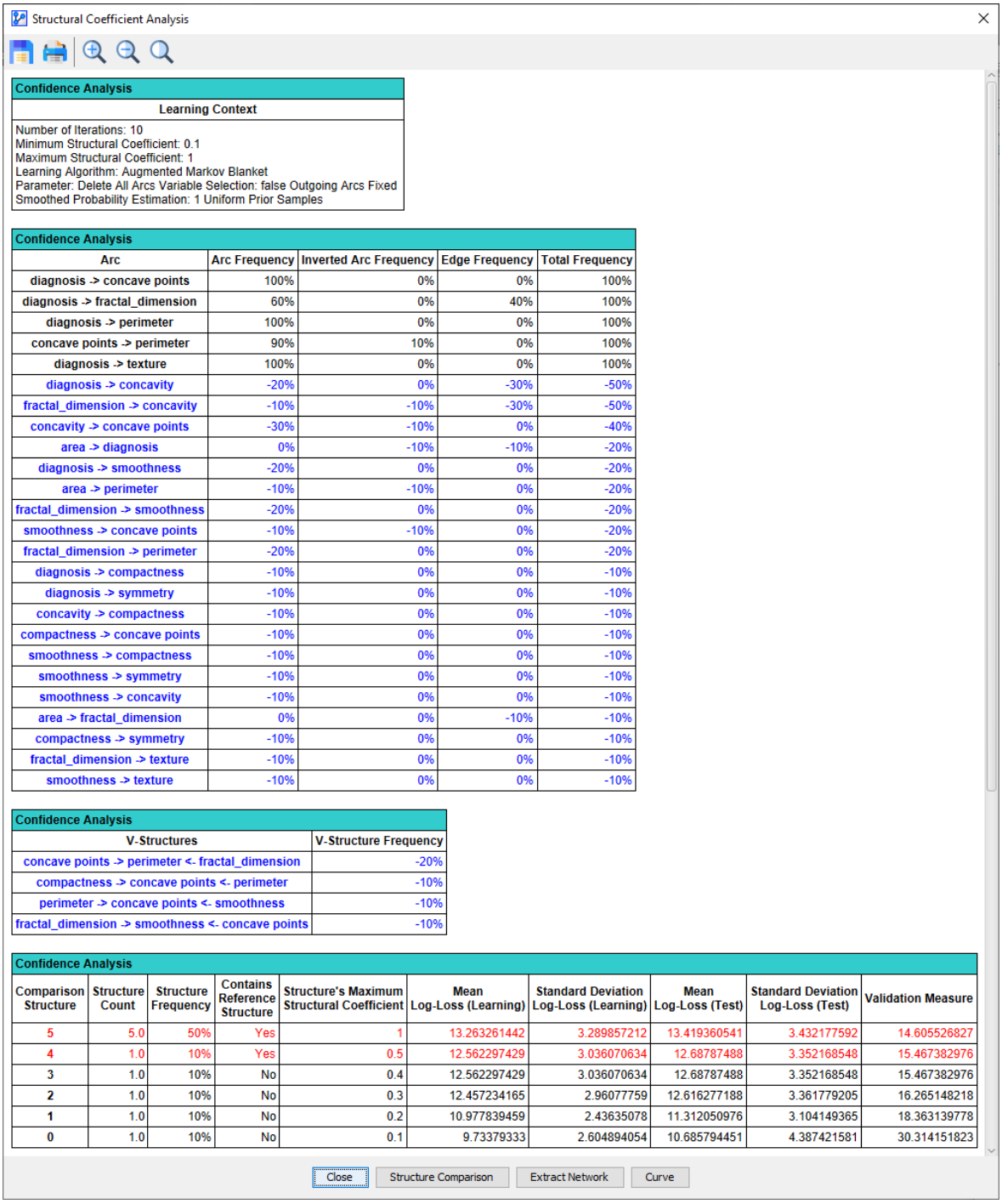
BayesiaLab prompts us to specify the range of SC values to be examined and the number of iterations to be performed. It is worth noting that the minimum SC value should not be set to 0, or even close to 0, without careful consideration.

An SC value of 0 would create a fully connected network, which can take a very long time to learn, depending on the number of variables, or even exceed the memory capacity of the computer running BayesiaLab. Technically, SC=0 implies an infinite dataset, which results in all relationships between nodes becoming significant.

Setting the Number of Iterations determines the interval steps to be taken within the specified range of the Structural Coefficient. We choose 10 iterations over an SC range between 0.1 and 1, which gives us increments of 0.1. With more complex models and more data, we might be more conservative and start with a narrower range, e.g., 0.5 to 1.



Clicking `OK` opens up a report that shows the range of changes due to modifying the Structural Coefficient.



Structural Coefficient Analysis

Confidence Analysis

Learning Context

Number of Iterations: 10
Minimum Structural Coefficient: 0.1
Maximum Structural Coefficient: 1
Learning Algorithm: Augmented Markov Blanket
Parameter: Delete All Arcs Variable Selection: false Outgoing Arcs Fixed
Smoothed Probability Estimation: 1 Uniform Prior Samples

Confidence Analysis

Arc	Arc Frequency	Inverted Arc Frequency	Edge Frequency	Total Frequency
diagnosis -> concave points	100%	0%	0%	100%
diagnosis -> fractal_dimension	60%	0%	40%	100%
diagnosis -> perimeter	100%	0%	0%	100%
concave points -> perimeter	90%	10%	0%	100%
diagnosis -> texture	100%	0%	0%	100%
diagnosis -> concavity	-20%	0%	-30%	-50%
fractal_dimension -> concavity	-10%	-10%	-30%	-50%
concavity -> concave points	-30%	-10%	0%	-40%
area -> diagnosis	0%	-10%	-10%	-20%
diagnosis -> smoothness	-20%	0%	0%	-20%
area -> perimeter	-10%	-10%	0%	-20%
fractal_dimension -> smoothness	-20%	0%	0%	-20%
smoothness -> concave points	-10%	-10%	0%	-20%
fractal_dimension -> perimeter	-20%	0%	0%	-20%
diagnosis -> compactness	-10%	0%	0%	-10%
diagnosis -> symmetry	-10%	0%	0%	-10%
concavity -> compactness	-10%	0%	0%	-10%
compactness -> concave points	-10%	0%	0%	-10%
smoothness -> compactness	-10%	0%	0%	-10%
smoothness -> symmetry	-10%	0%	0%	-10%
smoothness -> concavity	-10%	0%	0%	-10%
area -> fractal_dimension	0%	0%	-10%	-10%
compactness -> symmetry	-10%	0%	0%	-10%
fractal_dimension -> texture	-10%	0%	0%	-10%
smoothness -> texture	-10%	0%	0%	-10%

Confidence Analysis

V-Structures	V-Structure Frequency
concave points -> perimeter <- fractal_dimension	-20%
compactness -> concave points <- perimeter	-10%
perimeter -> concave points <- smoothness	-10%
fractal_dimension -> smoothness <- concave points	-10%

Confidence Analysis

Comparison Structure	Structure Count	Structure Frequency	Contains Reference Structure	Structure's Maximum Structural Coefficient	Mean Log-Loss (Learning)	Standard Deviation Log-Loss (Learning)	Mean Log-Loss (Test)	Standard Deviation Log-Loss (Test)	Validation Measure
5	5.0	50%	Yes	1	13.263261442	3.289857212	13.419360541	3.432177592	14.605526827
4	1.0	10%	Yes	0.5	12.562297429	3.036070634	12.68787488	3.352168548	15.467382976
3	1.0	10%	No	0.4	12.562297429	3.036070634	12.68787488	3.352168548	15.467382976
2	1.0	10%	No	0.3	12.457234165	2.96077759	12.616277188	3.361779205	16.265148218
1	1.0	10%	No	0.2	10.977839459	2.43635078	11.312050976	3.104149365	18.363139778
0	1.0	10%	No	0.1	9.73379333	2.604894054	10.685794451	4.387421581	30.314151823

Close | Structure Comparison | Extract Network | Curve

We only show a portion of the report here and omit a discussion of its elements. For a thorough explanation of this report, please see Structural Coefficient Analysis.

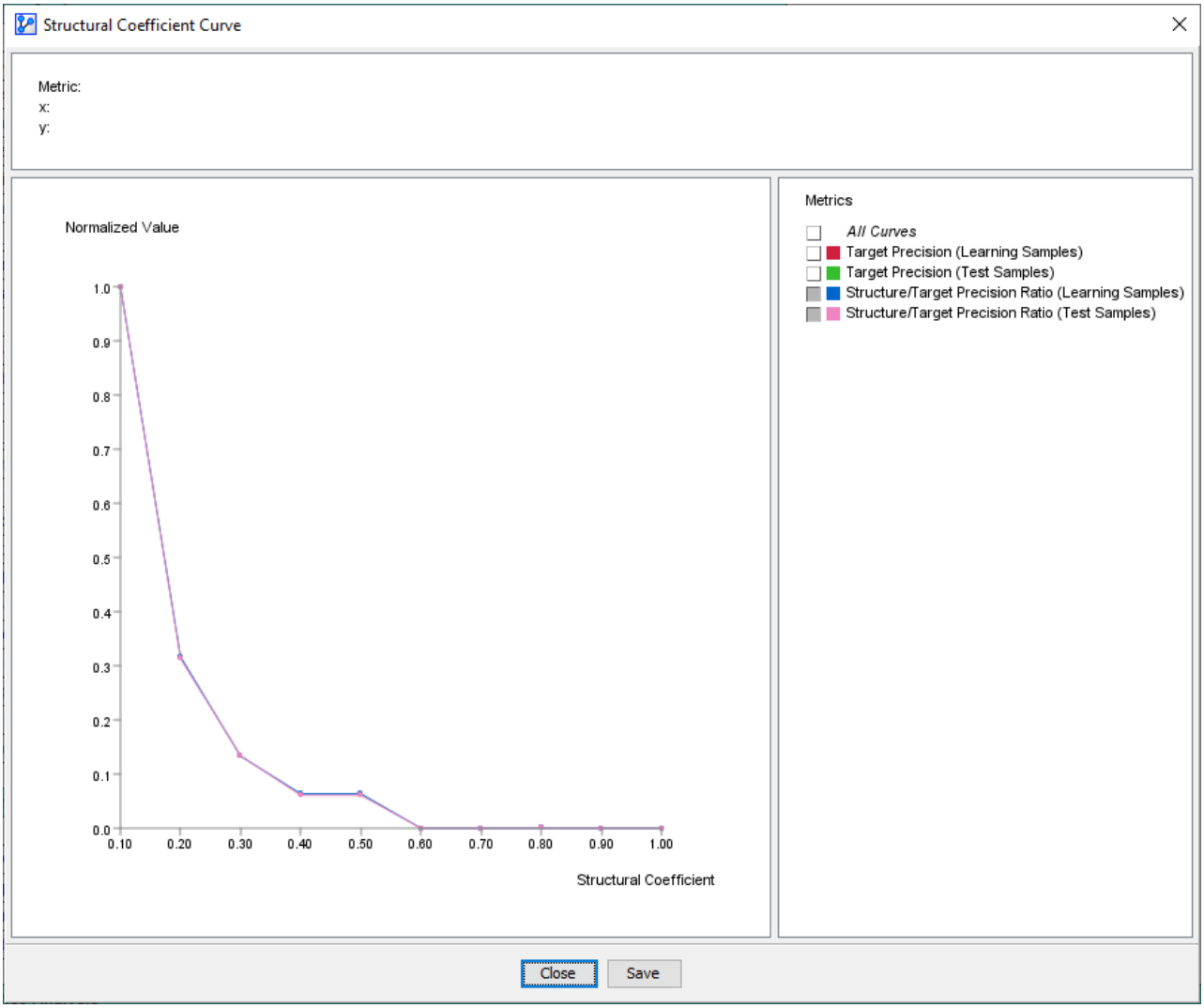
Instead, we focus on the `Curve` function, which can be activated by clicking on the corresponding button at the bottom of the report. This tool can plot the metrics that we specified earlier in the settings as we started the Structural Coefficient Analysis.

Our objective is to determine the correct level of network complexity for reliably high predictive performance while avoiding the over-fitting of the data. By clicking `Curve`, we can plot several different metrics for this purpose.

Structure/Target Precision Ratio

Selecting `Structure/Target Precision Ratio` provides a helpful measure for making trade-offs between predictive performance versus network complexity.

This plot can be best interpreted when following the curve from right to left. Moving to the left along the x-axis lowers the Structural Coefficient, which, in turn, results in a more complex Structure.



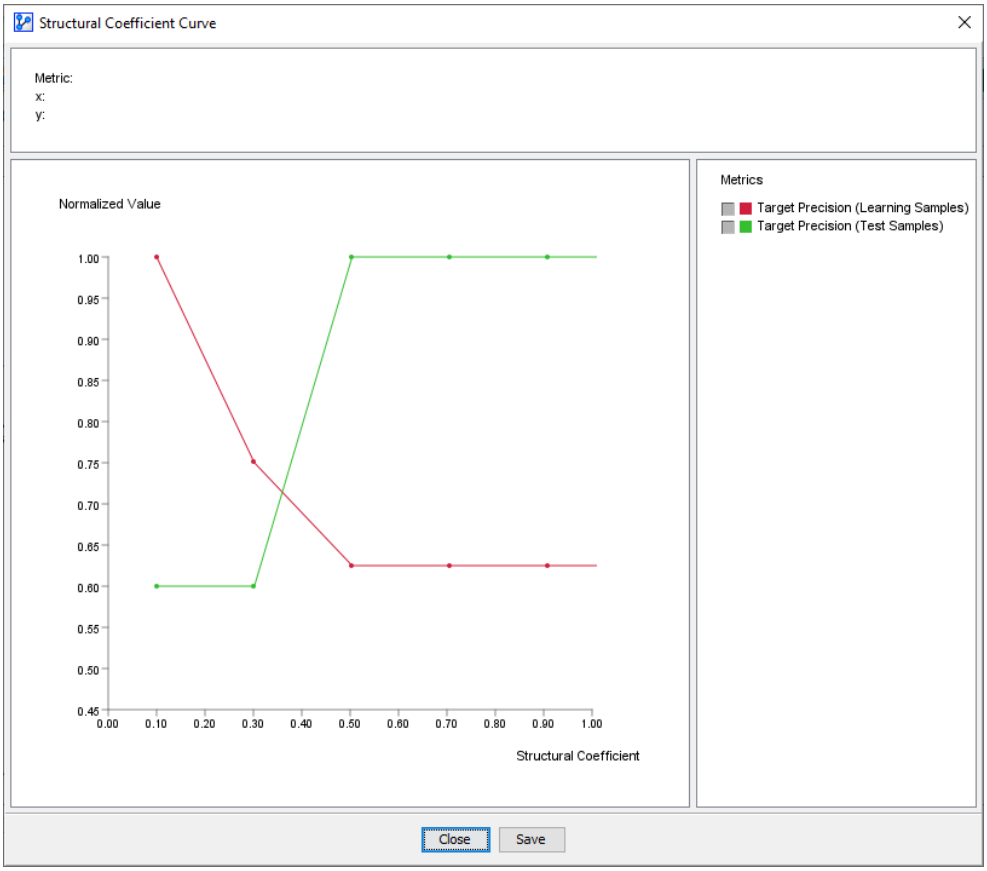
It becomes problematic when the Structure value increases faster than the Precision value, i.e. we increase complexity without improving Precision.

Typically, the "elbow" of the L-shaped curve identifies this critical point. Here, a visual inspection suggests that the "elbow" is around SC=0.4. The portion of the curve further to the left on the x-

axis, i.e., SC `<0.4`, shows that the structure is increasing without improving precision, which suggests overfitting. Hence, SC=0.4 could be a good value to examine further.

Another sign of overfitting is when the predictive performance of a model starts to diverge between the Learning Set and the Test Set. This means that the out-of-sample performance is no longer comparable to the in-sample performance.

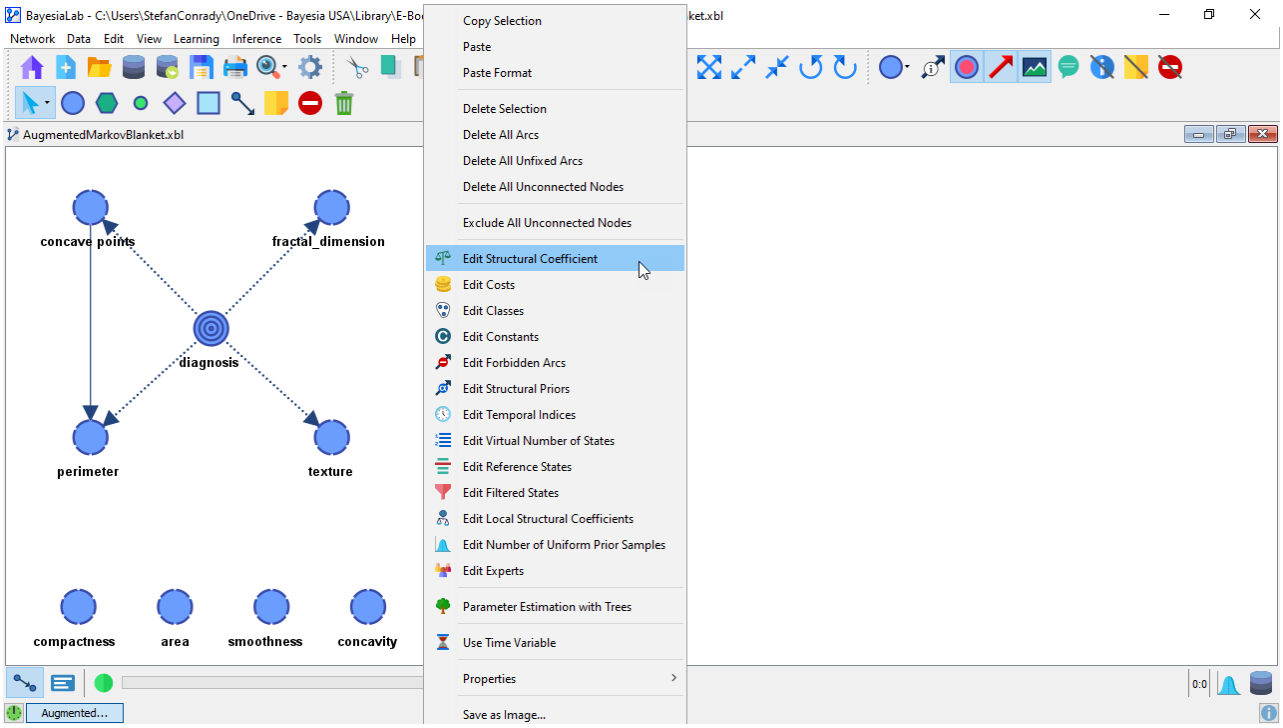
This is precisely what we can observe with the Target Precision Curves for both the Learning Set and the Test Set. For SC>0.5, the curves are parallel, which means that in-sample and out-of-sample performance move in sync. However, as the SC value drops below 0.5, the Learning Set performance increases while the Test Set performance drops, i.e., the curves diverge. The Target Precision for the Learning Set keeps increasing, while the Target Precision for the Test Set drops.



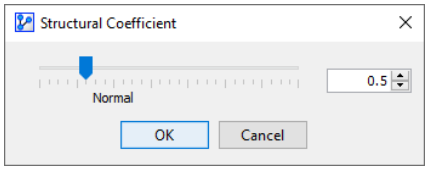
Augmented Markov Blanket (SC=0.5)

Having considered the curves in both of the above plots, we choose SC=0.5 for further evaluation.

The SC value can be set by right-clicking on the background of the Graph Panel and then selecting `Edit Structural Coefficient` from the Node Context Menu or via the menu: `Menu > Edit > Edit Structural Coefficient`.



The SC value can then be set with a slider or by typing in a numerical value.

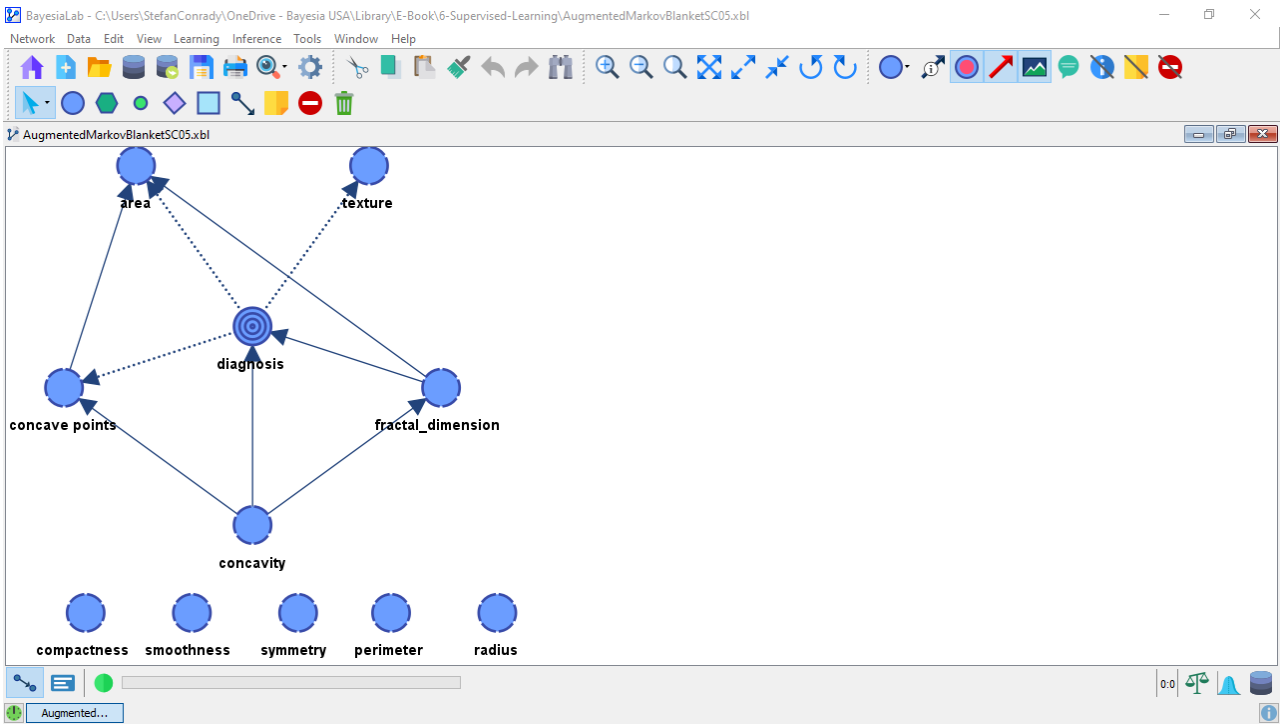


The Structural Coefficient icon ⚖ now indicates that we are employing an SC value other than the default of 1.

> The Structural Coefficient icon ⚖ features an unbalanced scale. This symbolizes that we departed from the balanced weighting of fit and complexity. Instead, we have "put our thumb on the scale" to pursue a better fit of our model while accepting a higher complexity.

After returning to the Modeling Mode F4 we relearn the network using the same Augmented Markov Blanket algorithm as before.

As expected, this produces a more complex network.



The key question is, will this increase in complexity deliver a performance advantage over the previously learned models?

Performance Analysis

So, we perform K-Folds Cross-Validation again, this time using the Augmented Markov Blanket at SC=0.5. The right panel in the overview below shows the results.

For comparison, we also show the performance of the earlier models we learned, i.e., the Markov Blanket (SC=1) in the left panel and the Augmented Markov Blanket (SC=1) in the center panel.

Markov Blanket (SC=1)		
K-Fold (10)		
Markov Blanket		
Target: diagnosis		
Value	Benign	Malignant
Gini Index	35.10%	59.26%
Relative Gini Index	94.41%	94.33%
Lift Index	1.443	1.942
Relative Lift Index	98.62%	97.97%
ROC Index	97.27%	97.23%
Calibration Index	63.29%	55.39%
Binary Log-Loss	0.266	0.266

Statistics		
Measure	Overall	Mean
R	0.882	0.883
R2	0.778	0.785
RMSE	0.229	0.221
NRMSE	22.86%	22.06%
Overall Precision	92.97%	92.95%
Mean Precision	92.10%	92.17%
Overall Reliability	92.95%	93.11%
Mean Reliability	92.79%	92.88%
Overall Relative Gini Index	94.39%	94.39%
Mean Relative Gini Index	94.37%	94.37%
Overall Relative Lift Index	98.38%	98.38%
Mean Relative Lift Index	98.30%	98.30%
Overall ROC Index	97.26%	97.26%
Mean ROC Index	97.25%	97.25%
Overall Calibration Index	60.41%	60.41%
Mean Calibration Index	59.34%	59.34%
Overall Log-Loss	0.266	0.266
Mean Binary Log-Loss	0.266	0.266

Occurrences		
Value	Benign (357)	Malignant (212)
Benign (365)	341	24
Malignant (204)	16	188
Reliability		
Value	Benign (357)	Malignant (212)
Benign (365)	93.43%	6.58%
Malignant (204)	7.84%	92.16%
Precision		
Value	Benign (357)	Malignant (212)
Benign (365)	95.52%	11.32%
Malignant (204)	4.48%	88.68%

Augmented Markov Blanket (SC=1)		
K-Fold (10)		
Augmented Markov Blanket		
Target: diagnosis		
Value	Benign	Malignant
Gini Index	35.23%	59.46%
Relative Gini Index	94.71%	94.68%
Lift Index	1.444	1.943
Relative Lift Index	98.66%	98.05%
ROC Index	97.42%	97.41%
Calibration Index	65.35%	59.51%
Binary Log-Loss	0.258	0.258

Statistics		
Measure	Overall	Mean
R	0.891	0.893
R2	0.793	0.8
RMSE	0.22	0.214
NRMSE	22.04%	21.42%
Overall Precision	94.20%	94.20%
Mean Precision	93.46%	93.68%
Overall Reliability	94.19%	94.57%
Mean Reliability	94.08%	94.32%
Overall Relative Gini Index	94.70%	94.70%
Mean Relative Gini Index	94.69%	94.69%
Overall Relative Lift Index	98.44%	98.44%
Mean Relative Lift Index	98.35%	98.35%
Overall ROC Index	97.42%	97.42%
Mean ROC Index	97.41%	97.41%
Overall Calibration Index	63.34%	63.34%
Mean Calibration Index	62.43%	62.43%
Overall Log-Loss	0.258	0.258
Mean Binary Log-Loss	0.258	0.258

Occurrences		
Value	Benign (357)	Malignant (212)
Benign (364)	344	20
Malignant (205)	13	192
Reliability		
Value	Benign (357)	Malignant (212)
Benign (364)	94.51%	5.50%
Malignant (205)	6.34%	93.66%
Precision		
Value	Benign (357)	Malignant (212)
Benign (364)	96.36%	9.43%
Malignant (205)	3.64%	90.57%

Augmented Markov Blanket (SC=0.5)		
K-Fold (10)		
Augmented Markov Blanket		
Target: diagnosis		
Value	Benign	Malignant
Gini Index	35.23%	59.54%
Relative Gini Index	94.73%	94.80%
Lift Index	1.441	1.944
Relative Lift Index	98.46%	98.10%
ROC Index	97.43%	97.47%
Calibration Index	64.88%	65.84%
Binary Log-Loss	0.277	0.277

Statistics		
Measure	Overall	Mean
R	0.894	0.896
R2	0.798	0.805
RMSE	0.217	0.211
NRMSE	21.72%	21.07%
Overall Precision	94.20%	94.20%
Mean Precision	93.65%	93.90%
Overall Reliability	94.19%	94.49%
Mean Reliability	93.91%	94.05%
Overall Relative Gini Index	94.76%	94.76%
Mean Relative Gini Index	94.76%	94.76%
Overall Relative Lift Index	98.33%	98.33%
Mean Relative Lift Index	98.28%	98.28%
Overall ROC Index	97.45%	97.45%
Mean ROC Index	97.45%	97.45%
Overall Calibration Index	65.20%	65.20%
Mean Calibration Index	65.36%	65.36%
Overall Log-Loss	0.277	0.277
Mean Binary Log-Loss	0.277	0.277

Occurrences		
Value	Benign (357)	Malignant (212)
Benign (360)	342	18
Malignant (209)	15	194
Reliability		
Value	Benign (357)	Malignant (212)
Benign (360)	95%	5%
Malignant (209)	7.18%	92.82%
Precision		
Value	Benign (357)	Malignant (212)
Benign (360)	95.80%	8.49%
Malignant (209)	4.20%	91.51%

Among a myriad of other available measures, we have typically referenced the Overall Precision for evaluation purposes. In this regard, the latest Augmented Markov Blanket (SC=0.5) does not show an improvement, i.e., the Overall Precision remains at 94.2%.

So, is there any benefit to the added complexity? It would depend on the context. Here, the objective is to distinguish between benign and malignant cell samples. Presumably, a false negative would be the worst forecast error. It would label a malignant sample as benign and perhaps cause a delay in a patient's treatment.

Focusing on the False Negative Rate of the three models, we see an improvement from 11.32% (left) to 9.43% (center) to 8.49% (right). This means that the best model in this regard reduces the number of False Negatives by one-third.

There are numerous other approaches available in BayesiaLab to help improve the model further, e.g., choosing a different learning algorithm, learning structural priors, reviewing the discretization, etc.

However, for the purposes of this tutorial, we conclude our model optimization efforts here and continue on the basis of the Augmented Markov Blanket (SC=0.5) in the next section: Model Inference.

Chapter 7: Unsupervised Learning

Unsupervised Structural Learning is perhaps the purest form of knowledge discovery, as no hypotheses are constraining the exploration of possible relationships between variables. BayesiaLab offers a wide range of algorithms for that purpose. Making this technology easily accessible can potentially transform how researchers approach high-dimensional problem domains.

> This chapter was written based on version 5.4 of BayesiaLab running on a Mac. All screenshots reflect that configuration.

Example: Stock Market

We find the mass of data available from financial markets to be an ideal proving ground for experimenting with knowledge discovery algorithms that generate Bayesian networks. Comparing machine-learned knowledge with our personal understanding of the stock market can perhaps allow us to validate BayesiaLab's "discoveries." For instance, any structure that is discovered by BayesiaLab's algorithms should be consistent with an equity analyst's understanding of fundamental relationships between stocks.

In this chapter, we will utilize Unsupervised Learning algorithms to automatically generate Bayesian networks from daily stock returns recorded over a six-year period. We will examine 459 stocks from the S&P 500 index, for which observations are available over the entire timeframe. We selected the S&P 500 as the basis for our study, as the companies listed on this index are presumably among the best-known corporations worldwide, so even a casual observer should be able to critically review the machine-learned findings. In other words, we are trying to machine learn the obvious, as any mistakes in this process would automatically become self-evident. Quite often, experts' reaction to such machine-learned findings is, "Well, we already knew that." Indeed, that is the very point we want to make, as machine learning can–within seconds–catch up with human expertise accumulated over the years and then rapidly expand beyond what is already known.

The power of such algorithmic learning will be still more apparent in entirely unknown domains. However, if we were to machine learn the structure of a foreign equity market for expository purposes, we would probably not be able to judge the resulting structure as plausible or not.

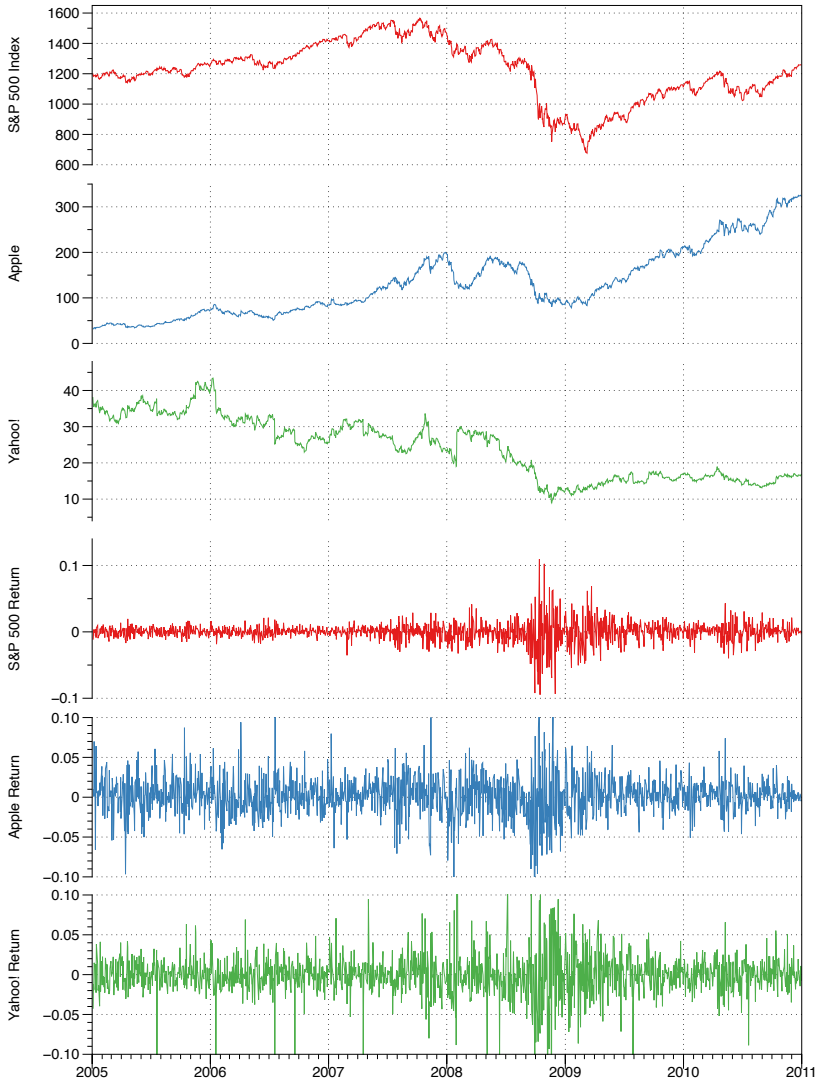
In addition to generating human-readable and interpretable structures, we want to illustrate how we can immediately use machine-learned Bayesian networks as "computable knowledge" for automated inference and prediction. Our objective is to gain both a qualitative and quantitative understanding of the stock market by using Bayesian networks. In the quantitative context, we will also show how BayesiaLab can carry out inference with multiple pieces of uncertain and even conflicting evidence. The ability of Bayesian networks to perform computations under uncertainty makes them suitable for a wide range of real-world applications.

Dataset

The S&P 500 is a free-float capitalization-weighted index of the prices of 500 large-cap common stocks actively traded in the United States, which has been published since 1957. The stocks included in the S&P 500 are those of large publicly held companies that trade on either of the two largest American stock market exchanges; the New York Stock Exchange and the NASDAQ. For our case study, we have tracked the daily closing prices of all stocks included in the S&P 500 index from January 3, 2005, through December 30, 2010, only excluding those stocks that were not traded continuously over the entire study period. This leaves a total of 459 stock prices with 1,510 observations each. The top three panels show the S&P 500 Index, plus the stock prices for Apple Inc. and Yahoo! Inc. Note that the plot of the S&P 500 Index is only shown for reference; the index will not be included in the analysis.

Data Preparation and Transformation

Rather than treating the time series in levels, we difference the stock prices and compute the daily returns. More specifically, we will take differences in the logarithms of the levels, which is a good approximation of the daily stock return in percent. After this transformation, 1,509 observations remain. The bottom three panels display the returns.



Analysis Workflow

- Data Import
- Unsupervised Learning
- Inference

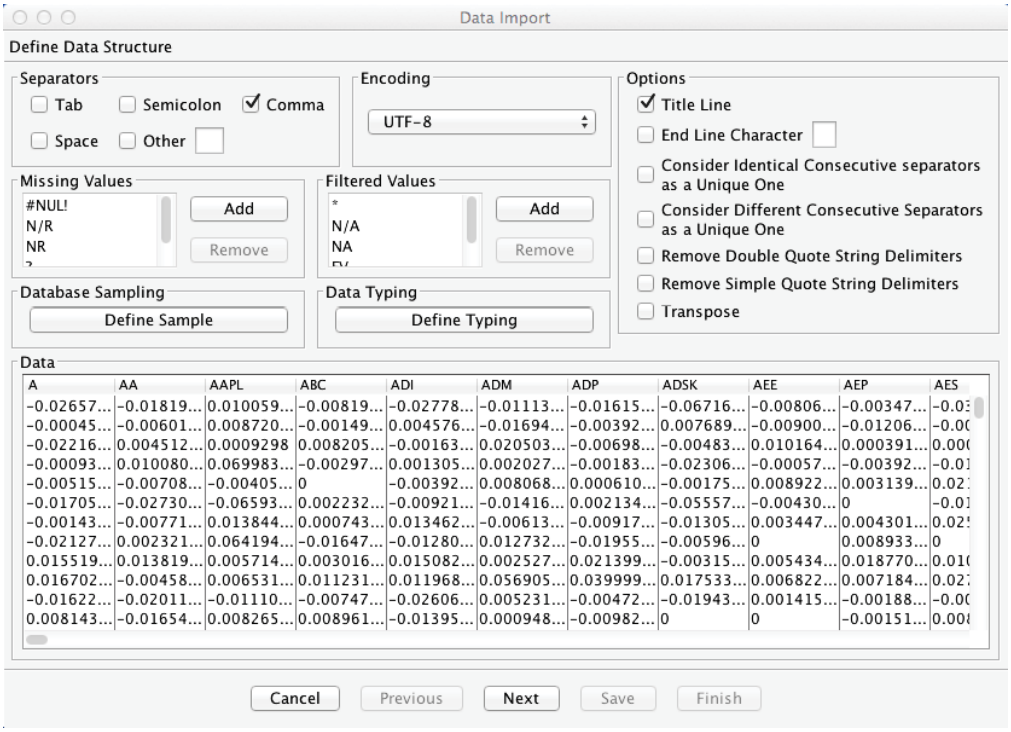
Data Import

Data Import

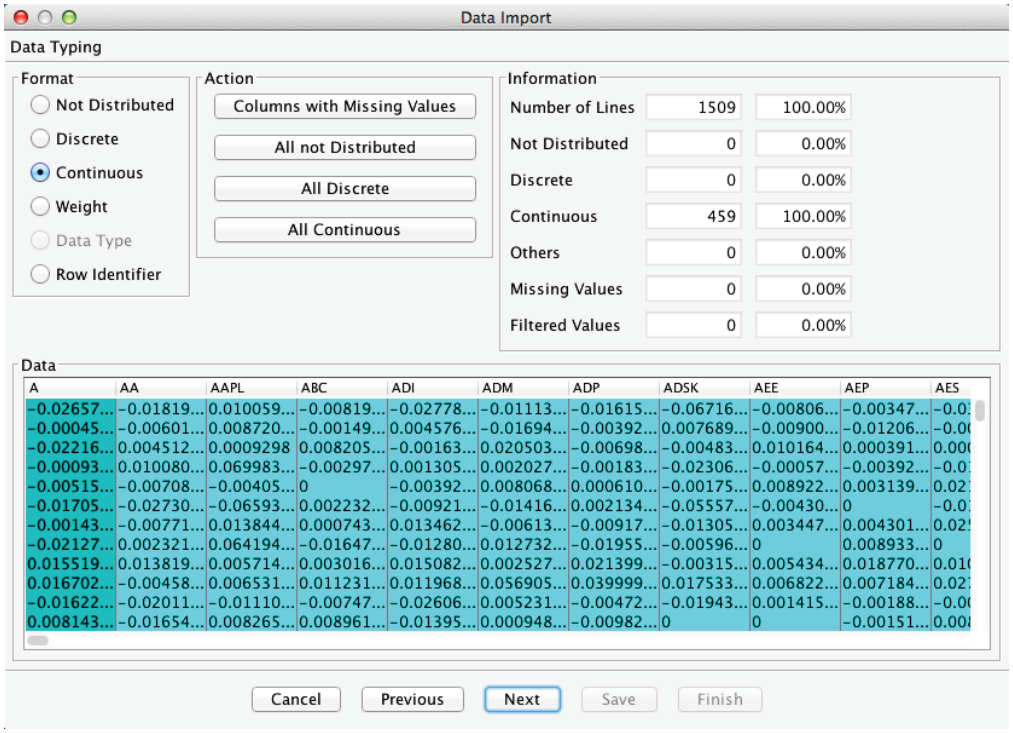
We use BayesiaLab's Data Import Wizard to load all 459 time series into memory from the data file:

Download: `SP500.csv`

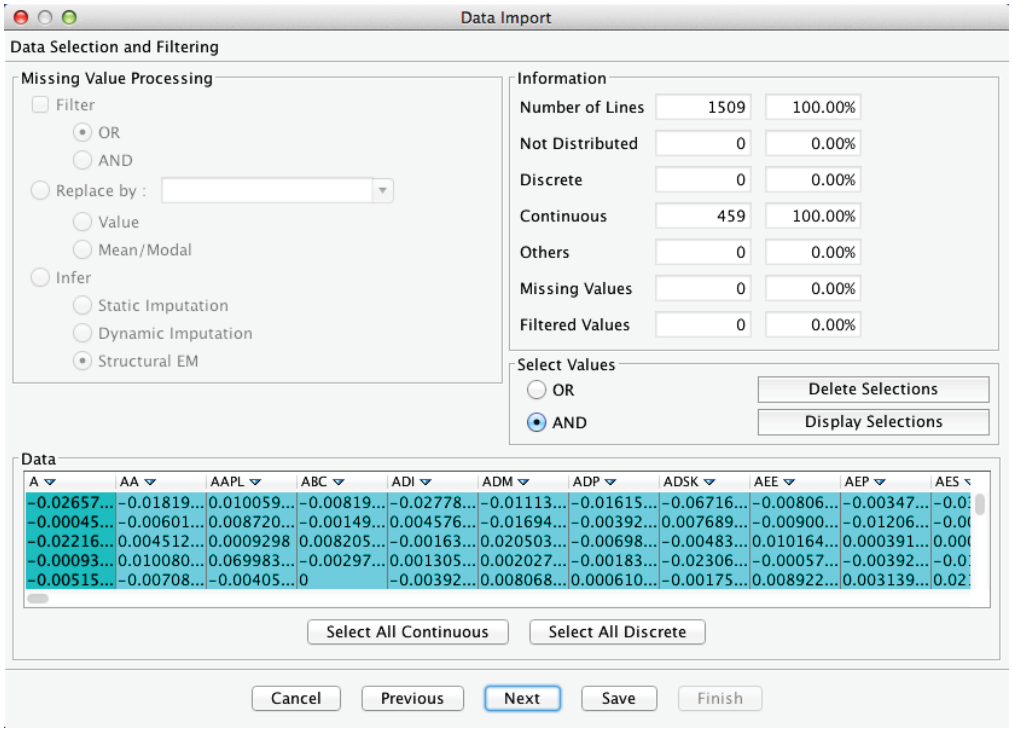
BayesiaLab automatically detects the column headers, which contain the ticker symbols that we will use as variable names (a ticker symbol is an abbreviation used to uniquely identify publicly traded stocks).



The next step identifies the variable types contained in the dataset and, as expected, BayesiaLab finds 459 Continuous variables.



There are no missing values in this database, so the next step of the Data Import Wizard can be skipped entirely. We still show this step below for reference, although all options are grayed out.

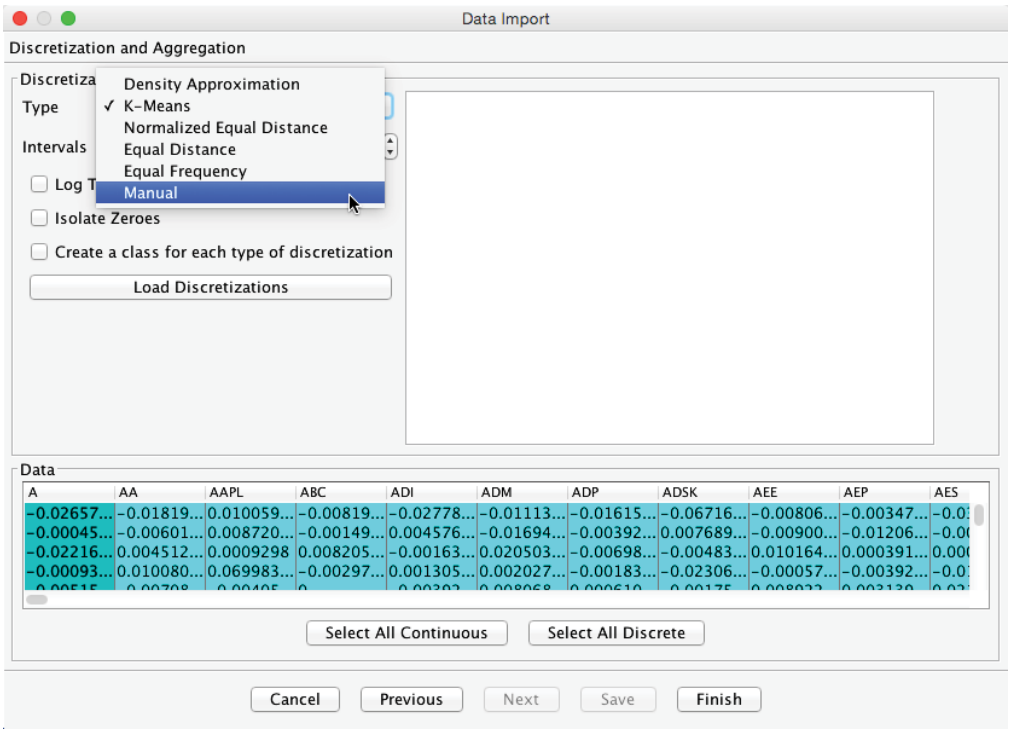


Data Discretization

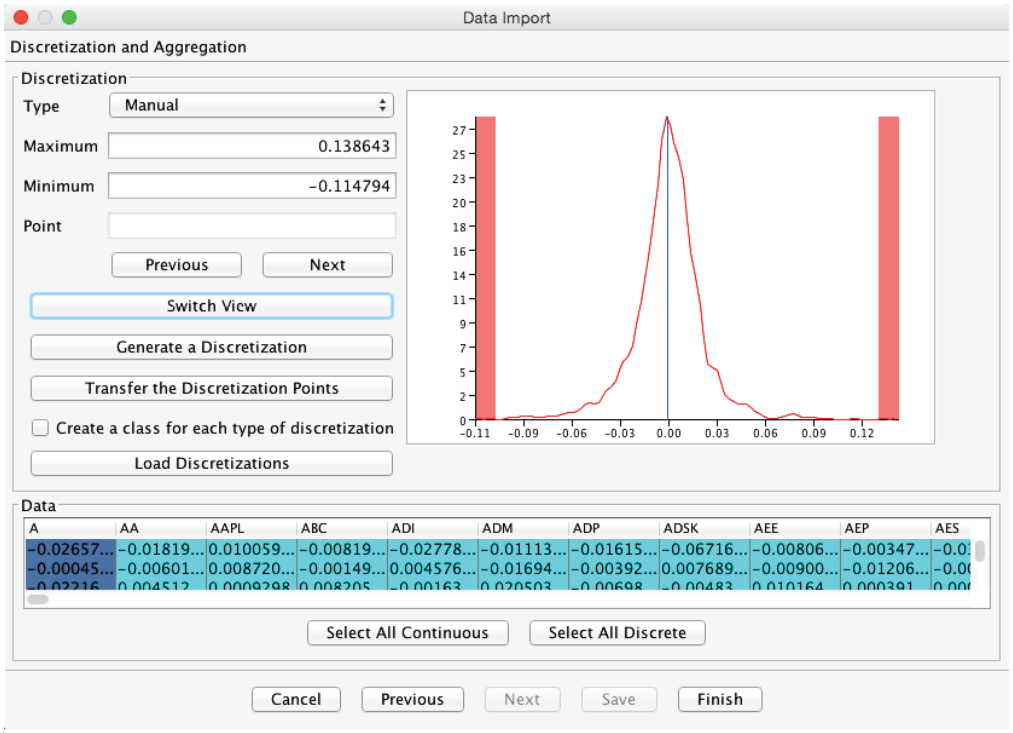
While we can defer a discussion of Missing Values Processing, for now, we must carefully consider our options in the next step of the Data Import Wizard. Here, we need to discretize all Continuous variables, which means all 459 variables in our case. In the context of Unsupervised Learning, we do not have a specific target variable. Hence, we have to choose one of the univariate discretization algorithms. Following the recommendations presented in Chapter 6, we choose K-Means. Furthermore, given the number of available observations, we aim for a discretization with 5 bins, as per the heuristic discussed in Chapter 6 under Discretization Intervals.

While helpful, any such heuristics should not be considered conclusive. Only once a model is learned can we properly evaluate the adequacy of the selected discretization. In BayesiaLab, we also have access to the Discretization function again anytime after completing the Data Import Wizard, which makes experimentation with different discretization methods and intervals very easy. In the Modeling Mode F4, we can start a new discretization with `Menu > Learning > Discretization`.

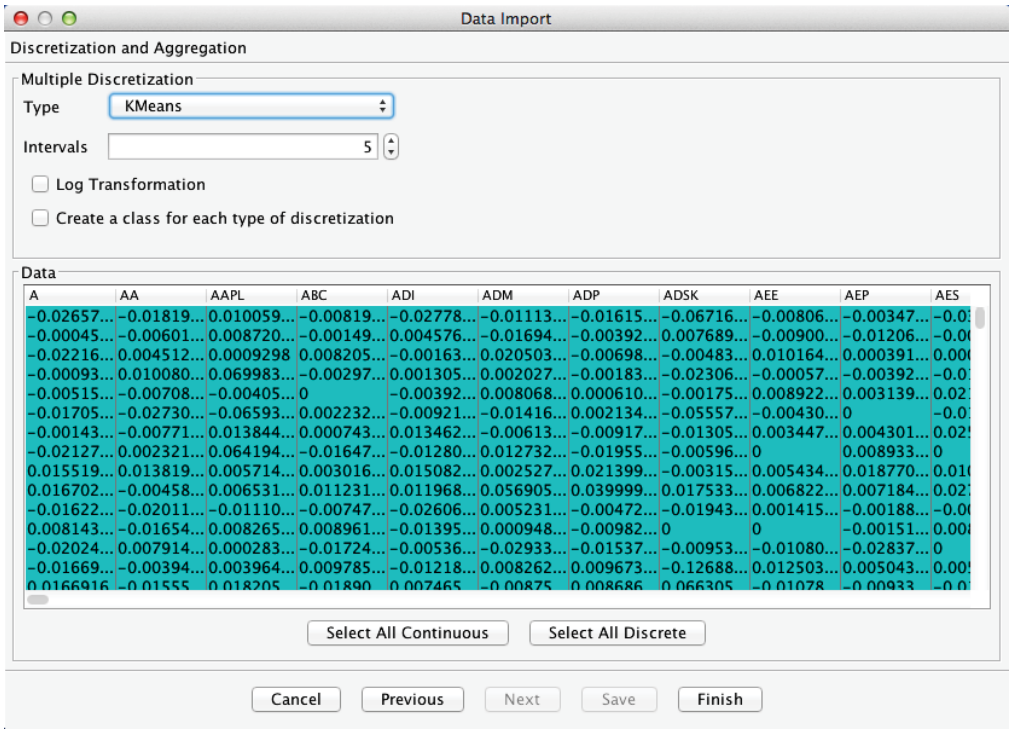
Before proceeding with the automatic discretization of all variables, we examine the type of density functions that we can find in this dataset. We use the built-in plotting function for this purpose, which is available in the next step of the Data Import Wizard.



After selecting Manual from the Discretization drop-down menu and then clicking `Switch View`, we obtain the probability density function of the first variable A.

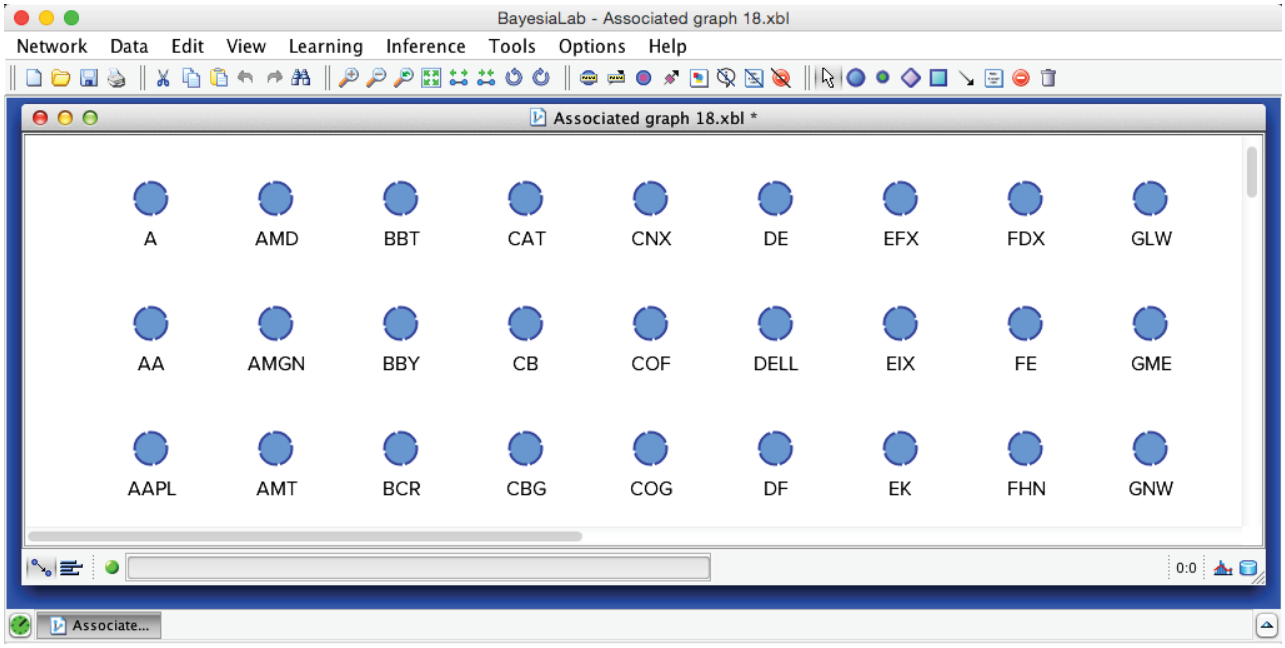


Without formally testing it for normality, we judge that the distribution of A (Agilent Technologies Inc.) does resemble the shape of a Normal distribution. In fact, the distributions of all variables in this dataset appear fairly similar, which further supports our selection of the K-Means algorithm. We click `Finish` to perform the discretization. A progress bar is shown to report the progress.



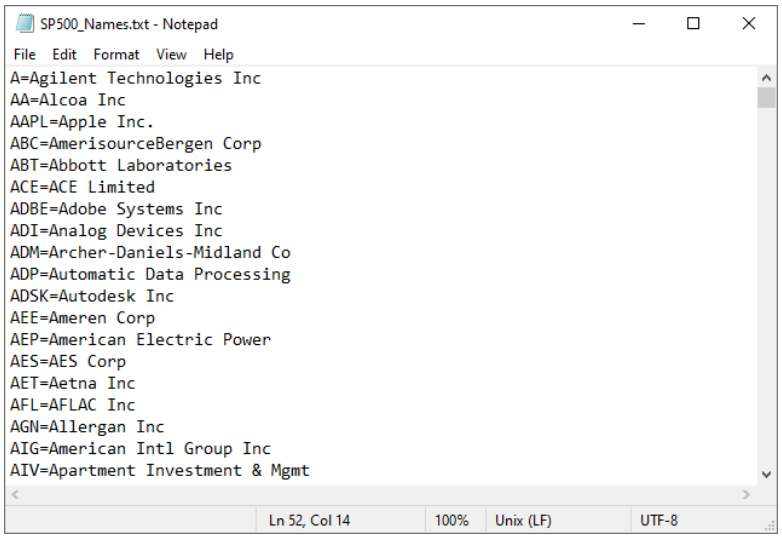
Modeling Mode

After completing the Data Import Wizard, the variables are presented in the Graph Panel. The original variable names, which were stored in the first row of the dataset, become our Node Names.

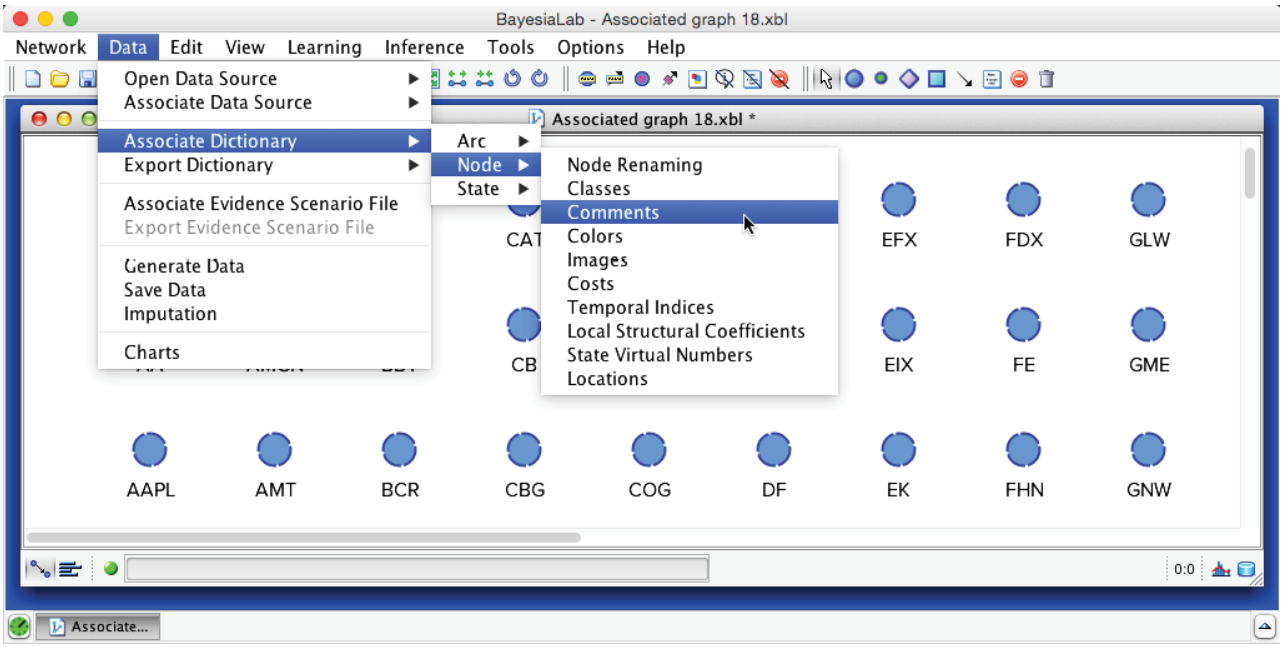


At this point, it is practical to add Node Comments to associate full company names with the short ticker symbols. We use a Dictionary file for that purpose:

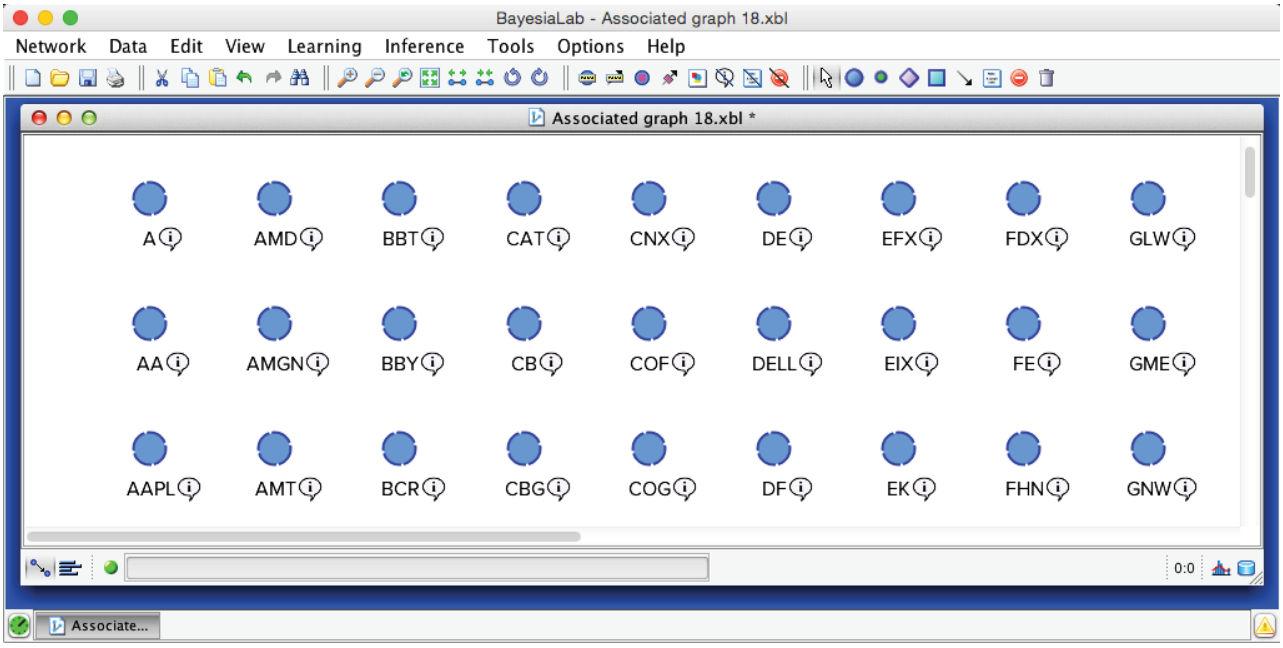
Download: `SP500_Names.txt`



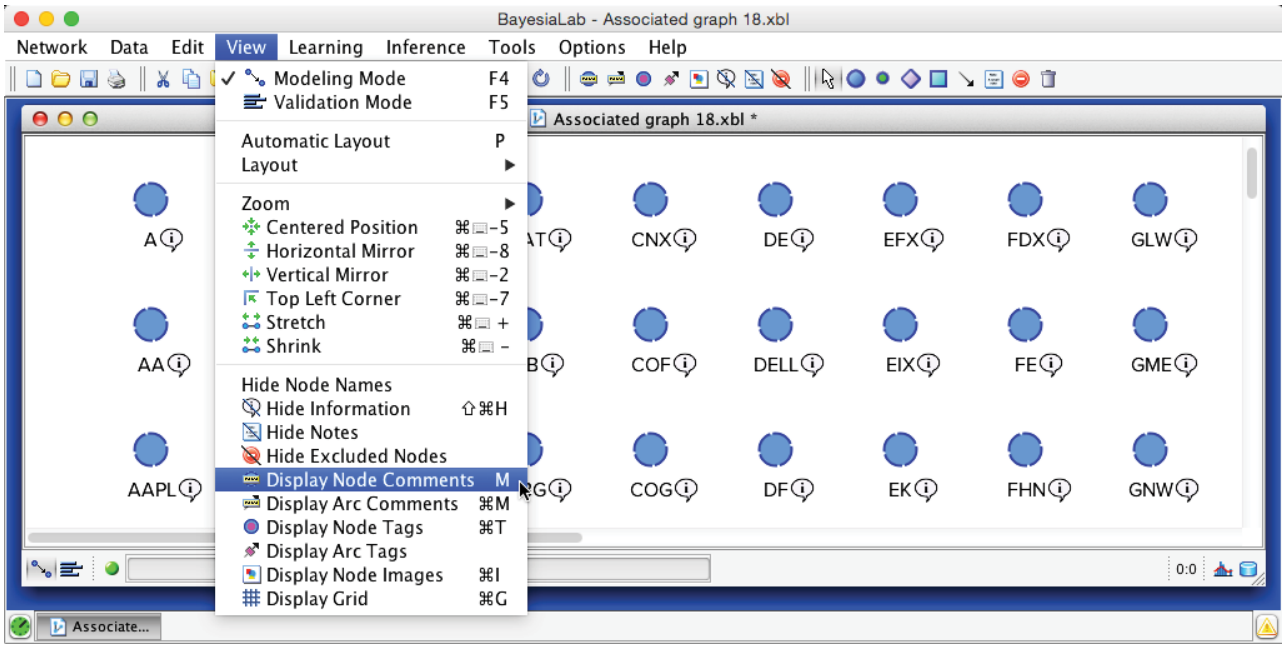
This file can be loaded into BayesiaLab via `Menu > Data > Associate Dictionary > Node > Comments`.



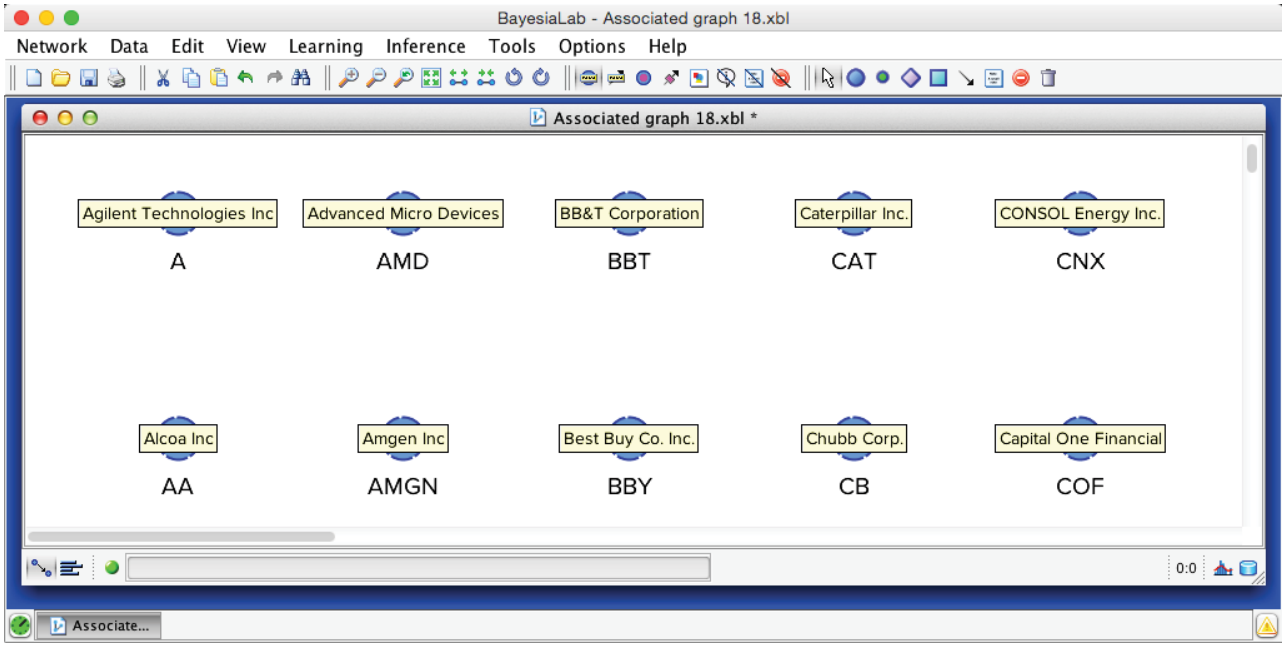
Once the Node Comments are loaded, a small call-out symbol (💬) appears next to each Node Name, confirming that associating the Dictionary completed successfully.



As the name implies, selecting `Menu > View > Display Node Comments` reveals the full company names.



Node Comments can be displayed for either all nodes or only for selected ones.



Data Import Review

Before proceeding with the first learning step, we recommend switching to the Validation Mode F5 to verify the results of the import and discretization.

This gives us an opportunity to compare the variables' statistics with our understanding of the domain. At first glance, mean values of near-zero for all distributions might suggest that stock prices remained "flat" throughout the observation period. For the S&P 500 index, this was actually true. However, it could not be true for all individual stocks, given that the Apple stock, for instance, increased ten-fold in value between 2005 and 2010. The seemingly low returns are due to the fact that we are studying daily returns rather than annual returns. On that basis, even the rapid appreciation of the Apple stock translates into an average daily return of "only" 0.2%. A "sanity check" of this kind is the prudent thing to do before proceeding to machine learning.

Inference

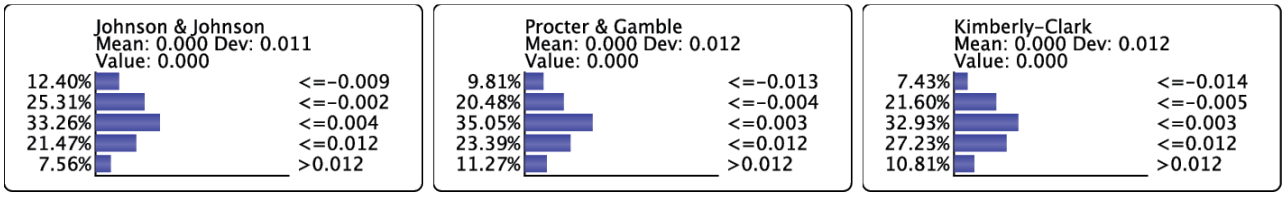
Inference

Throughout this book so far, we have performed inference with various types of evidence. The nature of the problem domain routinely determined the kind of evidence we used.

We will now use this example to systematically try out all types of evidence for performing inference. With that, we depart from our habit of showing only realistic applications. One could certainly argue that not all types of evidence are plausible in the context of a Bayesian network that represents the stock market. In particular, any inference we perform here with arbitrary evidence should not be interpreted as an attempt to predict stock prices. Nevertheless, for the sake of an exhaustive presentation, even this somewhat contrived exercise shall be educational.

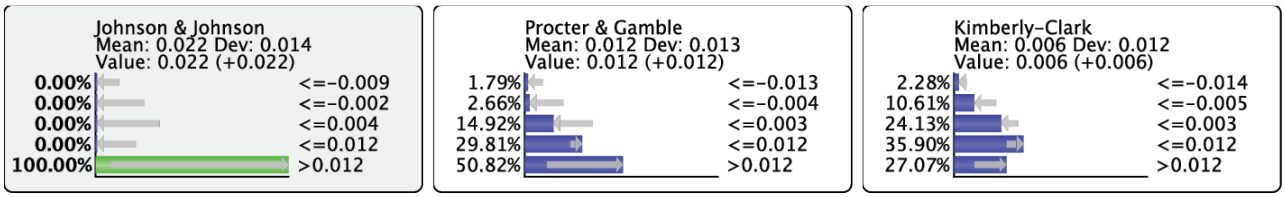
Within our large network of 459 nodes, we will only focus on a small subset of nodes, namely *PG* (Procter & Gamble), *JNJ* (Johnson & Johnson), and *KMB* (Kimberly-Clark). These nodes come from the "neighborhood" shown earlier.

We highlight *PG*, *JNJ*, and *KMB* to bring up their Monitors. Prior to setting any evidence, we see their marginal distributions in the Monitors. We see that the expected value (mean value) of the returns is 0.

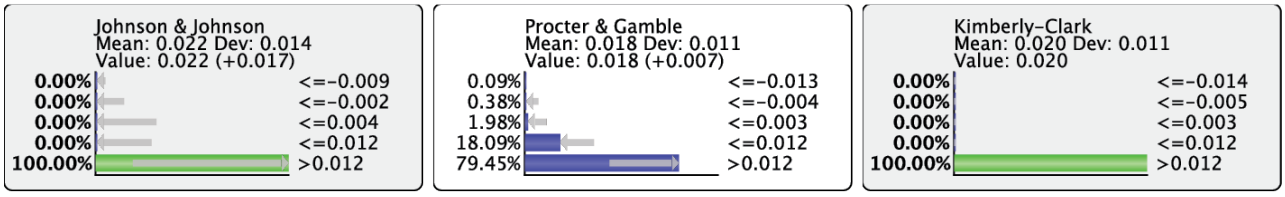


Inference with Hard Evidence

Next, we double-click the state `JNJ>0.012` to compute the posterior probabilities of *PG* and *KMB*, given this evidence. The gray arrows indicate how the distributions have changed compared to before, prior to setting evidence. Given the evidence, the expected values of *PG* and *KMB* are now 1.2% and 0.6%, respectively.



If we also set *KMB* to its highest state (`KMB>0.012`), this would further reduce the uncertainty of *PG* and compute an expected value of 1.8%. This means that *PG* had an average daily return of 1.8% on days when this evidence was observed.



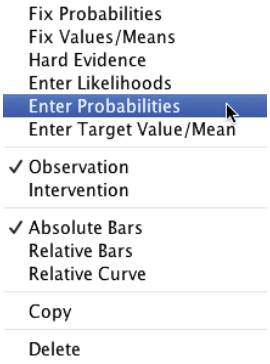
Inference with Probabilistic and Numerical Evidence

Given the discrete states of nodes, setting Hard Evidence is presumably intuitive to understand. However, the nature of many real-world observations calls for so-called Probabilistic Evidence or Numerical Evidence. For instance, the observations we make in a domain can include uncertainty. Also, Evidence Scenarios can consist of values that do not coincide with the values of nodes' states. So, as an alternative to Hard Evidence, we can use BayesiaLab to set such evidence.

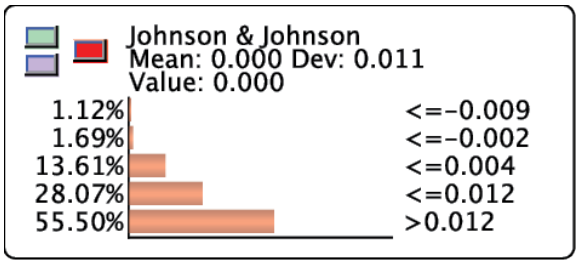
Probabilistic Evidence

Probabilistic Evidence is a convenient way of directly encoding our assumptions about possible conditions of a domain. For example, a stock market analyst may consider a scenario with a specific probability distribution for *JNJ* corresponding to a hypothetical period of time (i.e., a subset of days). Given his understanding of the domain, he can assign probabilities to each state, thus encoding his belief.

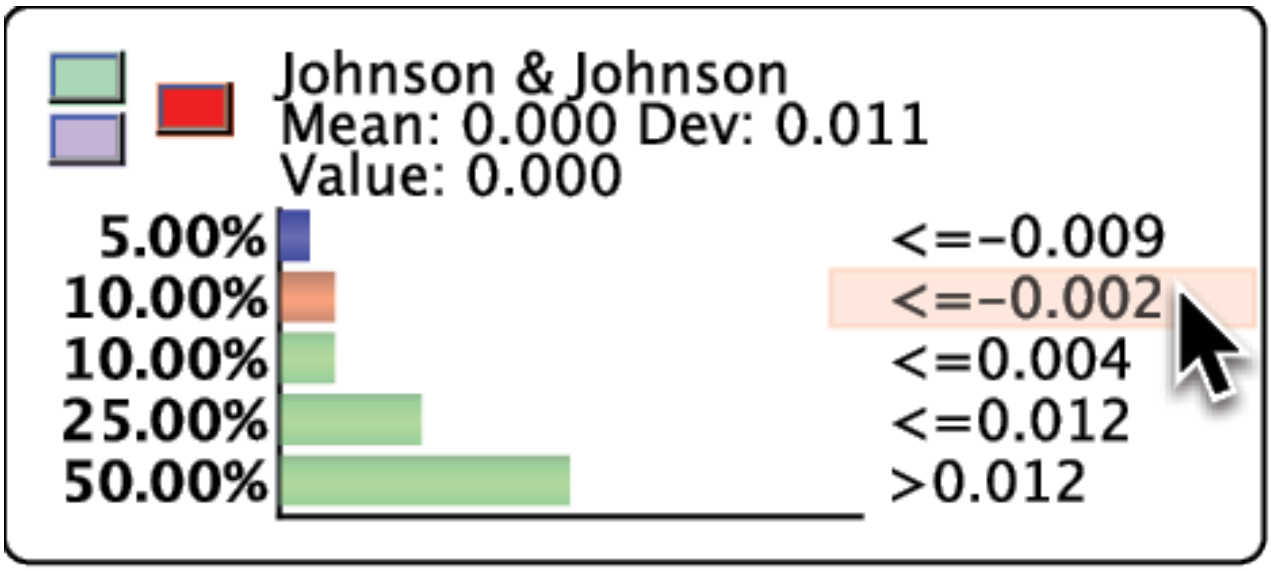
After removing the prior evidence, we can set such beliefs as Probabilistic Evidence by right-clicking the *JNJ* Monitor and then selecting `Enter Probabilities`.



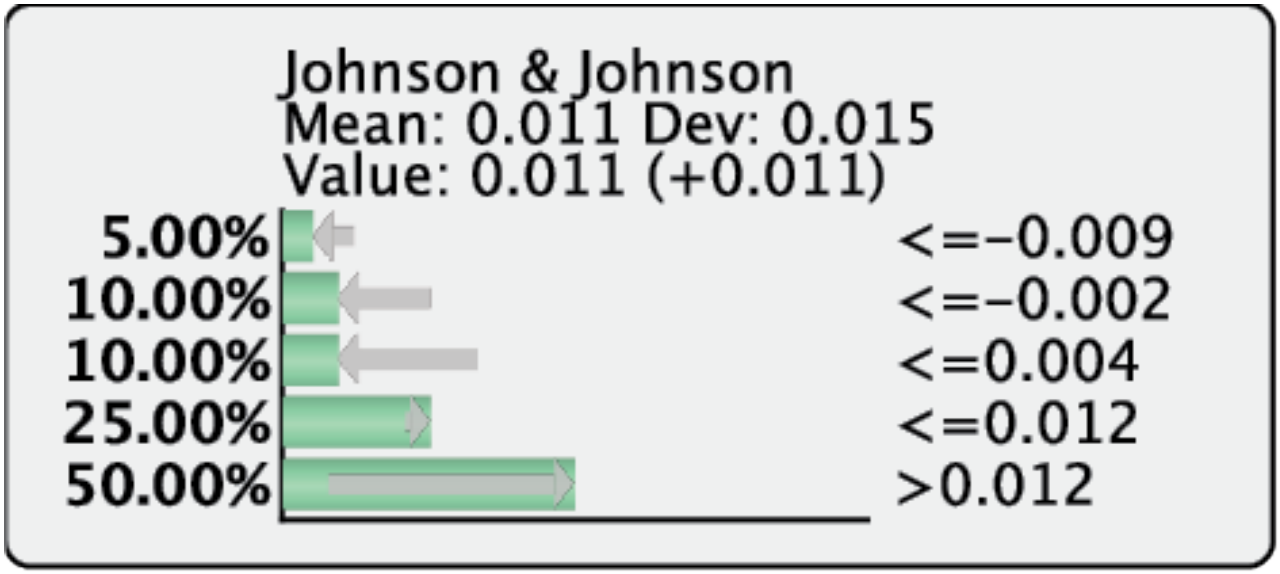
For the distribution of Probabilistic Evidence, the sum of the probabilities must be equal to 100%. We can adjust the Monitor's bar chart by dragging the bars to the probability levels that reflect the scenario under consideration. By double-clicking on the percentages, we can also directly enter the desired probabilities. Note that changing the probability of any state automatically updates the probabilities of all other states to maintain the sum constraint.



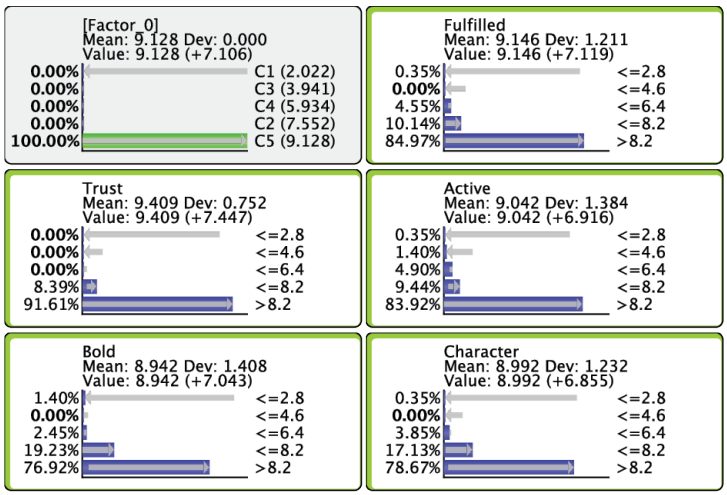
To remove a degree of freedom in the sum constraint, we left-click the State Name/Value in the Monitor to the right of each bar. Doing so locks the currently set probability and turns the corresponding bar green. The probability of this state will no longer be automatically updated while the probabilities of other states are being edited. This feature is essential for defining a distribution on nodes with more than two states. Another left-click on the same State Name/Value unlocks the probability again.



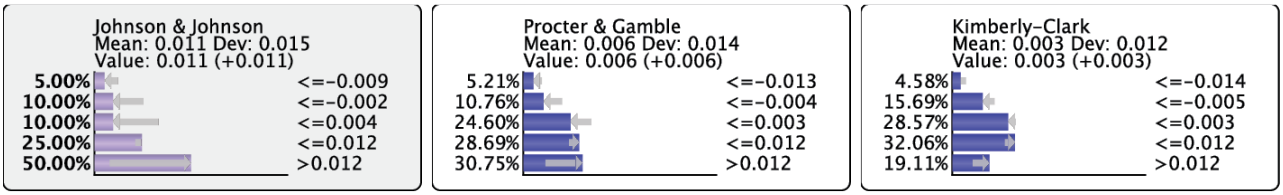
There are two ways to validate the entered distribution, via the green and the purple buttons. Clicking the green button defines a static likelihood distribution. This means that any additional piece of evidence on other nodes can update the distribution we set.



Clicking the purple button "fixes" the probability distribution we entered by defining dynamic likelihoods. This means that each new piece of evidence triggers an update of the likelihood distribution in order to maintain the same probability distribution.



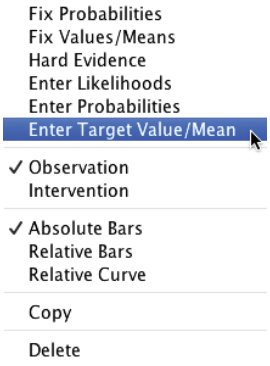
Using either validation method, BayesiaLab computes a likelihood distribution that produces the requested probability distribution. By setting this distribution, BayesiaLab also performs inference automatically and updates the probabilities of the other nodes in the network.



Numerical Evidence

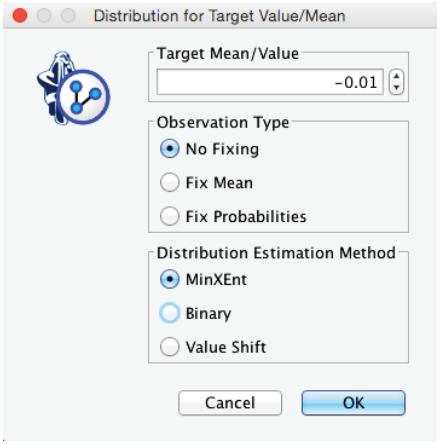
Instead of a specific probability distribution, an observation or scenario may exist as a single numerical value, meaning we must set Numerical Evidence. For instance, a stock market analyst may wish to examine how other stocks performed given a hypothetical period of time during which the average of the daily returns of *JNJ* was -1%. Naturally, this requires that we set evidence on *JNJ* with an expected (mean) value of -0.01 (=-1%). However, this task is not as straightforward as it may sound. The question will become apparent as we follow the steps to set this evidence.

First, we right-click *JNJ* Monitor and then select `Enter Target Value/Mean` from the Contextual Menu.



Next, we type "-0.01" into the dialog box for Target Mean/Value. Additionally, as was the case with Probabilistic Evidence, we have to choose the type of validation, but we now have three options under Observation Type:

- No Fixing, which is the same as the green button, i.e., validation with static likelihood.
- Fix Mean, which is the same as the purple button, except that the likelihood is dynamically computed to maintain the mean value, although the probability distribution can change as a result of setting additional evidence.
- Fix Probabilities, which is the same as the purple button, i.e., validation with dynamic likelihood.

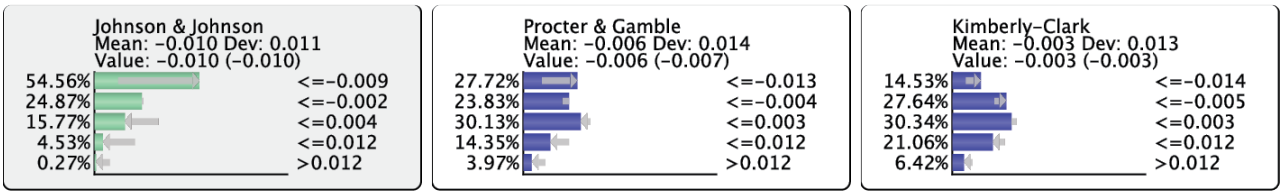


Apart from setting the validation method, we also need to choose the `Distribution Estimation Method`, as we need to come up with a distribution that produces the desired mean value. Needless to say, there is a great number of distributions that could potentially produce a mean value of -0.01. However, which one is appropriate?

To make a prudent choice, we must first understand what the evidence represents. Only then can we choose from the three available algorithms for generating the Target Distribution that will produce the Target Mean/Value.

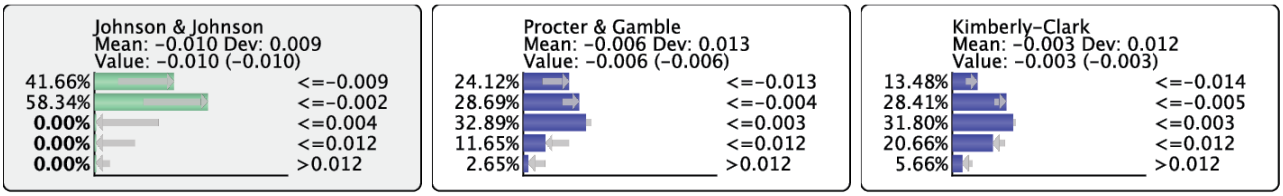
MinXEnt ("Minimum Cross-Entropy")

Using the MinXEnt algorithm, the Target Distribution, which produces the Target Mean/Value, is computed so that the Cross-Entropy between the original probability distribution of the node and the Target Distribution is minimized. The Monitor below shows the distribution with a mean of -0.01, which is "closest" in terms of Cross-Entropy to the original marginal distribution shown earlier.



Binary

If we select Binary, the Target Mean/Value is generated by interpolating between values of two adjacent states, hence the name. Here, a "mix" of the values of two states, i.e., *JNJ* `<=-0.009` and *JNJ* `<=-0.002`, produces the desired mean of -0.01.



Value Shift

With Value Shift, the Target Mean/Value is generated by shifting the values of each particle (or virtual observation) by the exact same amount.

Target Value/Mean Considerations

As we see in the examples above, using different Target Distributions as Numerical Evidence–albeit with the same mean value–results in different probability distributions.

Binary

The Binary algorithm produces the desired value through interpolation, as in Fuzzy Logic. Among the three available methods, it generates distributions that have the lowest degree of uncertainty. Using the Binary algorithm for generating a Target Mean/Value would be appropriate if two conditions are met:

1. There is no uncertainty regarding the evidence, i.e., we want the evidence to represent a specific numerical value. "No uncertainty" would typically apply in situations in which we want to simulate the effects of nodes that represent variables under our control.
2. The desired numerical value is not directly available by setting Hard Evidence. In fact, a distribution produced by the Binary algorithm would coincide with Hard Evidence if the requested Target Value/Mean precisely matched the value of a particular state.

Given that it is impossible to set prices in the stock market directly, it is clearly not a good example for illustrating this using the Binary algorithm. Perhaps a price elasticity model would be more appropriate. In such a model, we would want to infer sales volume based on one specific price level instead of a broad range of price levels within a distribution.

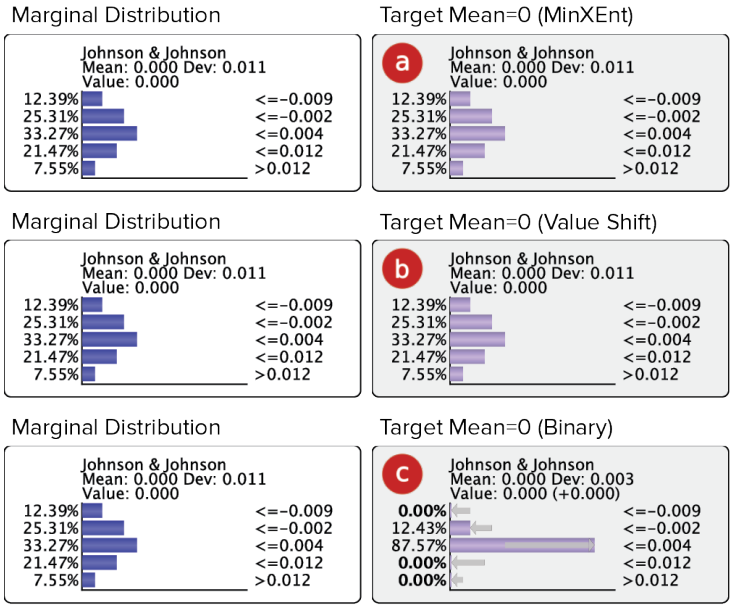
MinXEnt and Value Shift

The other two algorithms, MinXEnt and Value Shift, generate Soft Evidence. This means that the Target Distribution they supply should be understood as posterior distribution given evidence set on a "hidden cause", i.e., evidence on a variable not included in the model. As such, using MinXEnt or Value Shift is suitable for creating evidence that represents changing levels of measures like customer satisfaction. Unlike setting the price of a product, we cannot directly adjust the satisfaction of all customers to a specific level. This would imply setting an unrealistic distribution with low or no uncertainty.

More realistically, we would have to assume that higher satisfaction results from an enhanced product or better service, i.e., a cause from outside the model. Thus, we need to generate evidence for customer satisfaction as if a hidden cause produced it. This also means that MinXEnt and Value Shift will produce a distribution close to the marginal one if the targeted Numerical Evidence is close to the marginal value.

Special Cases of Numerical Evidence

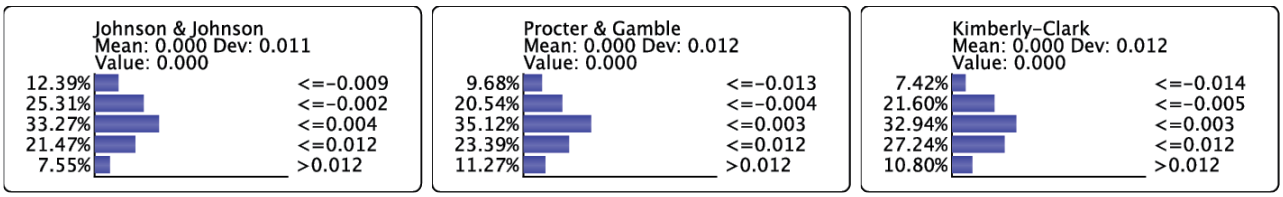
If the Numerical Evidence equals the current expected value, using MinXEnt (a) or Value Shift(b) will not change the distribution. Using the Binary algorithm (c), however, will return a different distribution (except in the context of a binary node).



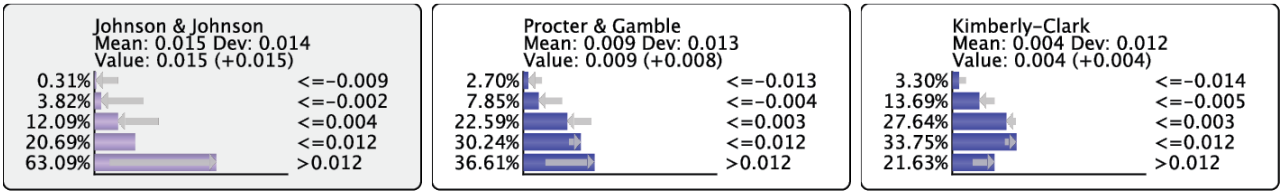
Conflicting Evidence

In the examples shown so far, setting evidence typically reduced uncertainty with regard to the node of interest. Just by visually inspecting the distributions, we can tell that setting evidence generally produces "narrower" posterior probabilities.

However, this is not always the case. Occasionally, separate pieces of evidence can conflict with each other. We illustrate this by setting such evidence on *JNJ* and *KMB*. We start with the marginal distribution of all nodes.

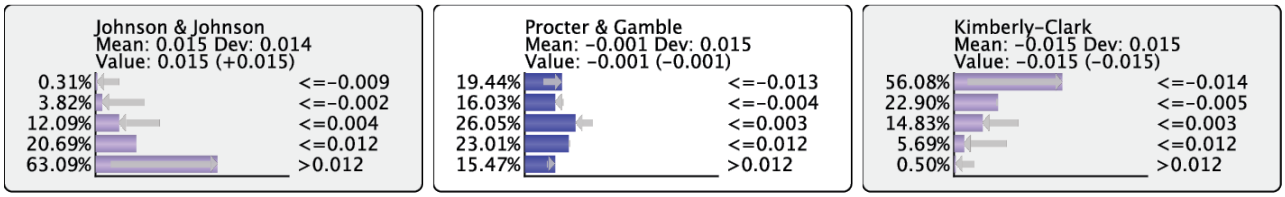


After setting Numerical Evidence (using MinXEnt) with a Target Mean/Value of +1.5% on *JNJ*.



The posterior probabilities inferred from the *JNJ* evidence indicate that the *PG* distribution is more positive than before. More importantly, the uncertainty regarding *PG* is lower. A stock market analyst would perhaps interpret the *JNJ* movement as a positive signal and hypothesize about a positive trend in the CPG industry. In an effort to confirm his hypothesis, he would probably look for additional signals that either confirm the trend and the related expectations regarding *PG* and similar companies.

In the *KMB* Monitor, the gray arrows and "(+0.004)" indicate that the first evidence increases the expectation that *KMB* will also increase in value. If we observed, however, that *KMB* decreased by 1.5% (once again using MinXEnt), this would go against our expectations.



The result is that we now have a more uniform probability distribution for *PG*–rather than a narrower distribution. This increases our uncertainty about the state of *PG* compared to the marginal distribution.

Even though it appears that we have "lost" information by setting these two pieces of evidence, we may have a knowledge gain after all: we can interpret the uncertainty regarding *PG* as a higher expectation of volatility.

Measures of Conflict

Beyond a qualitative interpretation of contradictory evidence, our Bayesian network model allows us to examine "conflict" beyond its common-sense meaning. A formal conflict measure can be defined by comparing the joint probabilities of the current model versus a reference model, given the same set of evidence for both.

A fully unconnected network is commonly used as the reference model, the so-called "straw model." It is a model that considers all nodes to be marginally independent. If the joint probability of the set of evidence returned by the model under study is lower than that of the reference model, we determine that we have a conflict. Otherwise, if the joint probability is higher, we conclude that the pieces of evidence are consistent.

The conflict measures that are available in BayesiaLab are formally defined as follows:

Global Conflict (Overall Conflict)

$$GC(E) = \frac{\log_2 P(e_1, ..., e_n)}{\prod_{i=1}^{n} P(e_i)}$$

where E is the current set of evidence consisting of n observations and e_i is the i^{th} piece of evidence.

Bayes Factor

$$BF(E,h) = \log_2 \frac{P(h \mid E)}{P(h)}$$

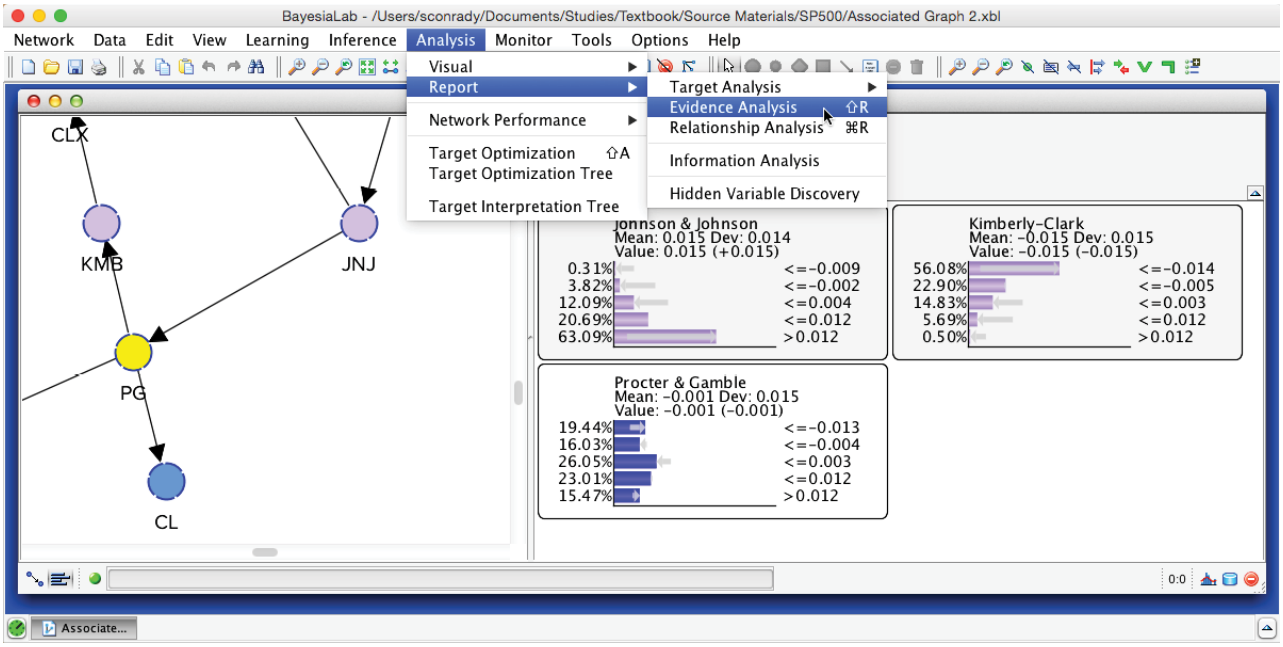
where h is a hypothetical piece of evidence that has not yet been set or observed.

Local Conflict (Local Consistency)

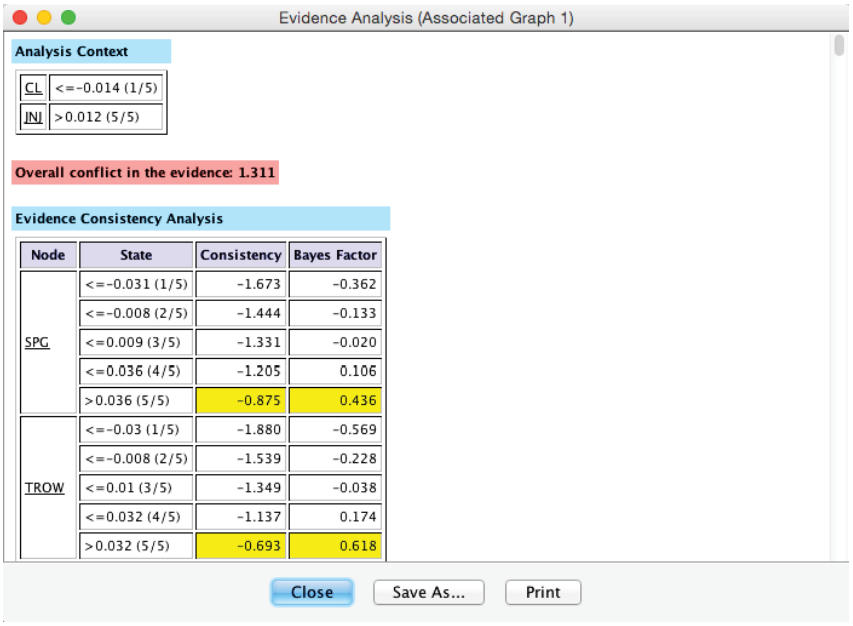
$$LC(E,h) = GC(E) + BF(E,h) \ = \log_2 \frac{P(E)P(h \mid E)}{\prod_{i=1}^{n} P(e_i)P(h)}$$

Evidence Analysis Report

Using these definitions, we can compute to what extent a new observation would be consistent with the current set of evidence. BayesiaLab provides us with this capability in the form of the Evidence Analysis Report, which can be generated by selecting `Menu > Analysis > Report > Evidence Analysis` from the main menu.



The Evidence Analysis Report displays two closely-related metrics, Local Consistency (LC) and the Bayes Factor (BF), for each state of each unobserved node in the network, given the set of evidence. The top portion of this report is shown below. Also, as we anticipated, an Overall Conflict between the two pieces of evidence is shown at the top of the report.



Summary

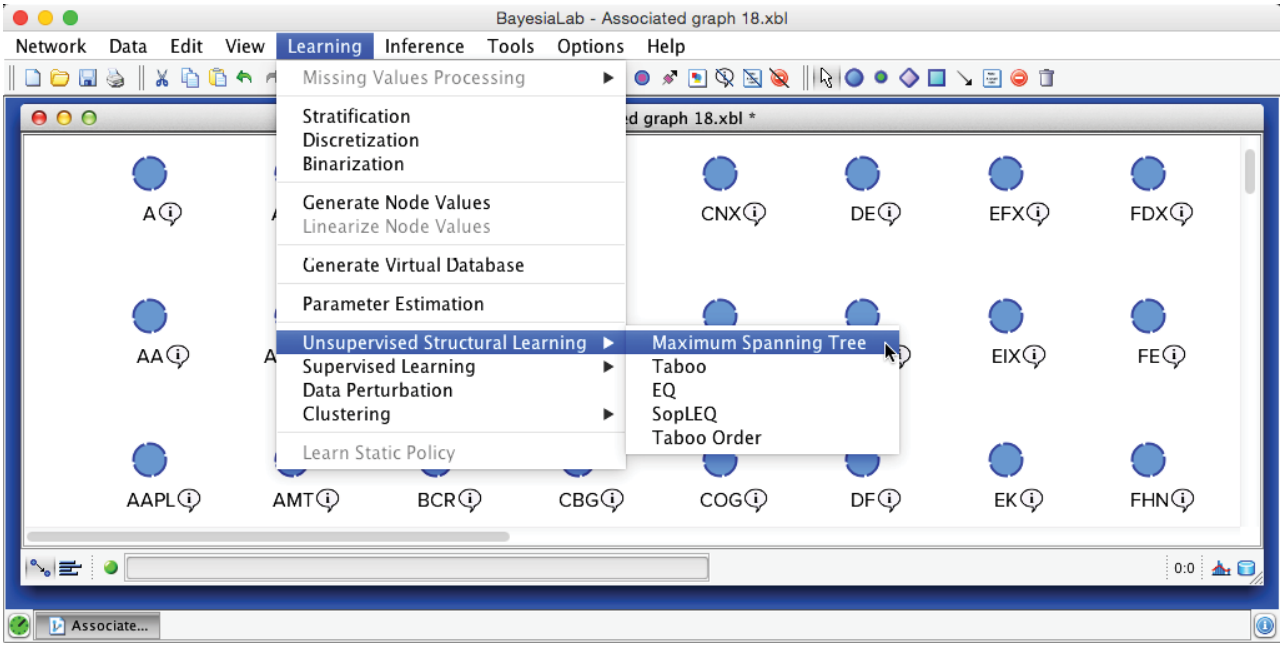
Unsupervised Learning is a practical approach for obtaining a general understanding of simultaneous relationships between many variables in a database. The learned Bayesian network facilitates visual interpretation plus the computation of omnidirectional inference, which can be based on any type of evidence, including uncertain and conflicting observations. Given these properties, Unsupervised Learning with Bayesian networks becomes a universal tool for knowledge discovery in high-dimensional domains.

Unsupervised Learning

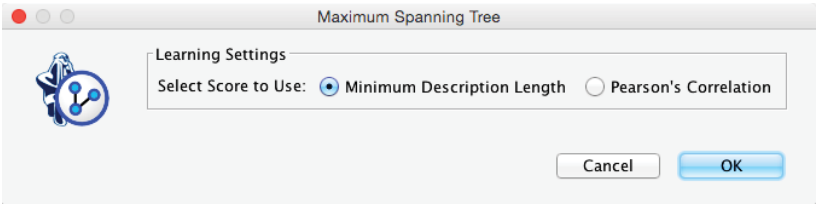
The computational complexity of BayesiaLab's Unsupervised Learning algorithms exhibits quadratic growth as a function of the number of nodes. However, the Maximum Weight Spanning Tree (MWST) is constrained to learning a tree structure (one parent per node), which makes it much faster than the other algorithms. More specifically, the MWST algorithm includes only one procedure with quadratic complexity, namely the initialization procedure that computes the matrix of bivariate relationships.

Given the number of variables in this dataset, we decide to use the MWST. Performing the MWST algorithm with a file of this size should only take a few seconds. Moreover, using BayesiaLab's layout algorithms, the tree structures produced by MWST can be easily transformed into easy-to-interpret layouts. Thus, MWST is a practical first step for knowledge discovery. Furthermore, this approach can be useful for verifying that there are no coding problems, e.g., with variables that are entirely unconnected. Given the quick insights that can be gleaned from it, we recommend using MWST at the beginning of most studies.

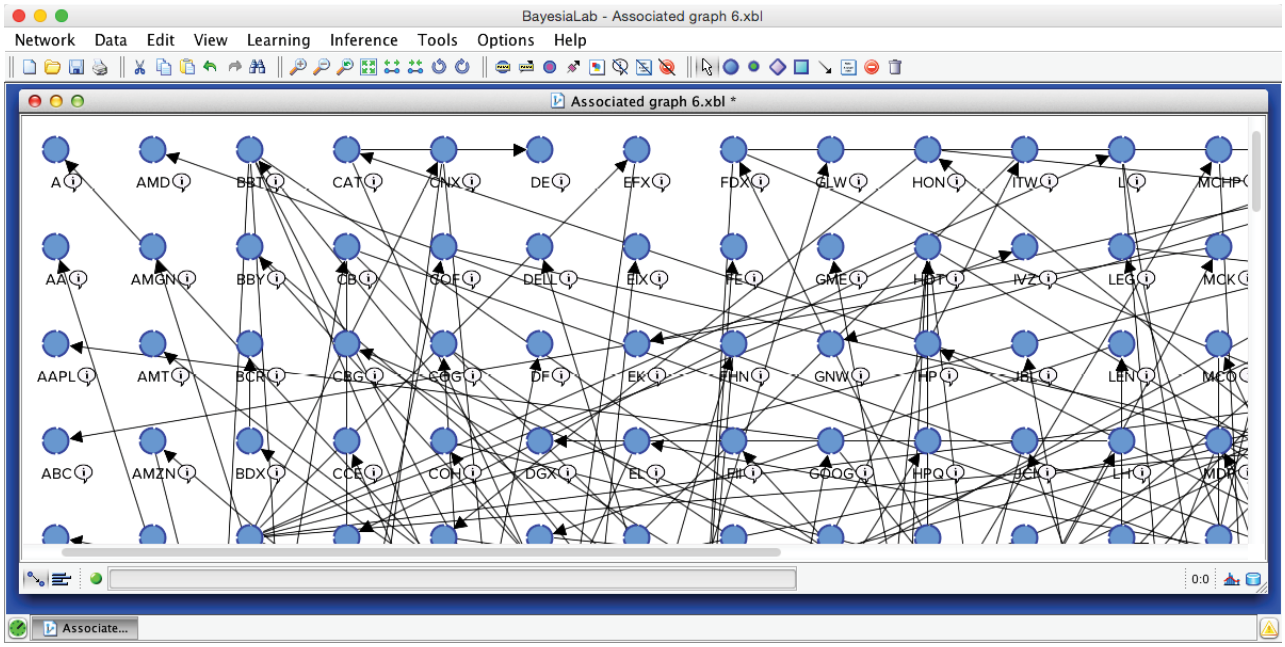
We return to Modeling Mode F4 and select `Menu > Learning > Unsupervised Structural Learning > Maximum Spanning Tree`.



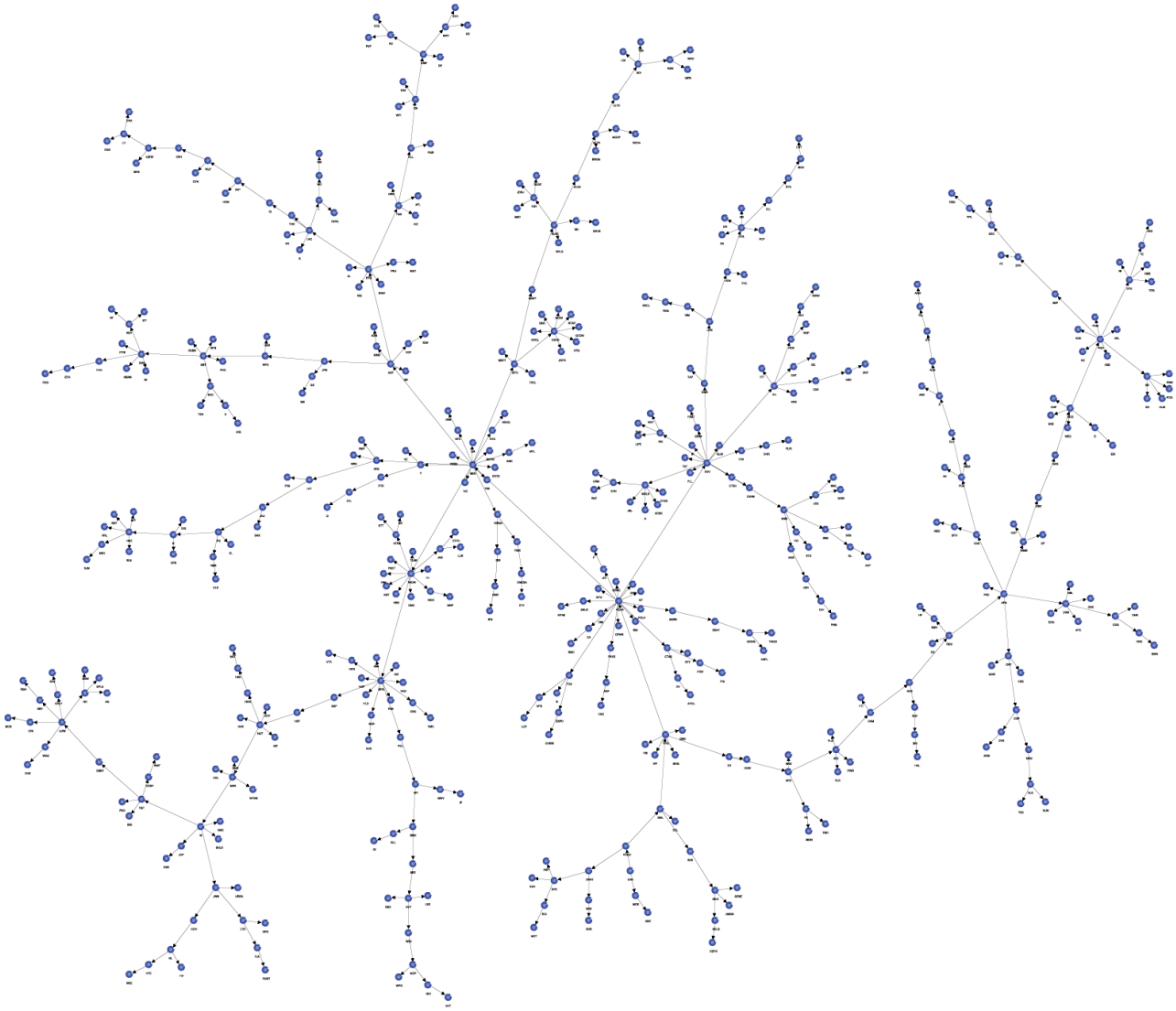
In addition to its one-parent constraint, MWST is also unique in that it is the only learning algorithm in BayesiaLab that allows us to choose the scoring method for learning, i.e., Minimum Description Length (MDL) or Pearson's Correlation. Unless we are certain about the linearity of the yet-to-be-learned relationships between variables, Minimum Description Length is the better choice and, hence, the default setting.



At first glance, the resulting network does not appear simple and tree-like at all.



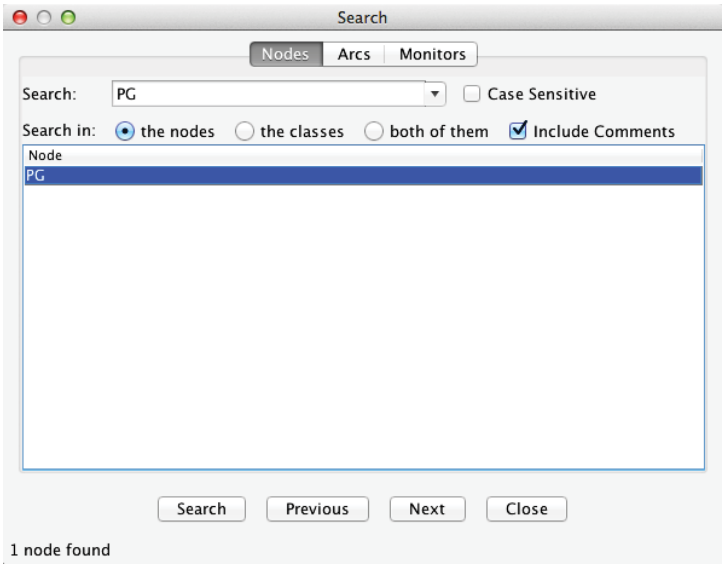
This can be addressed with BayesiaLab's built-in layout algorithms. `Selecting Menu > View > Automatic Layout` (Shortcut: P) quickly rearranges the network to reveal the tree structure. The resulting reformatted Bayesian network can now be readily interpreted.



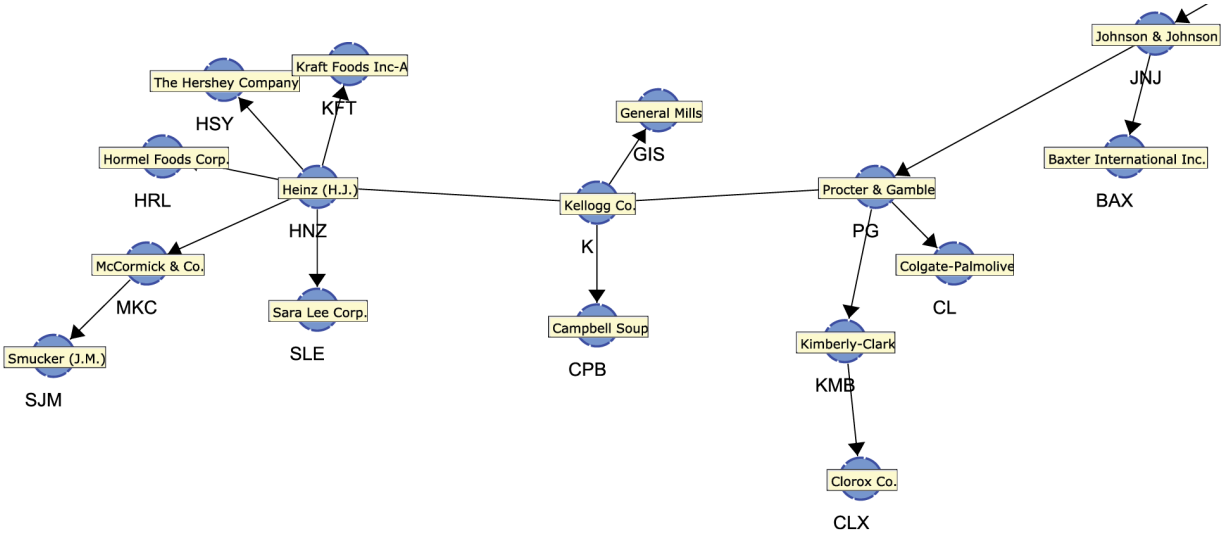
Network Analysis

Let us suppose we are interested in Procter & Gamble (*PG*). First, we look for the corresponding node using the Search function (Ctrl+F). Note that we can search for the full company name if we check `Include Comments`. Furthermore, we can use a combination of wildcards in the search, e.g., "*" as a placeholder for a character string of any length or "?" for a single character.

Selecting *PG* from the listing search results makes the corresponding node flash for a few seconds so it can be found among the hundreds of nodes on the screen.



Once located, we can zoom in to see *PG* and numerous adjacent nodes.



As it turns out, the "neighborhood" of Procter & Gamble contains many familiar company names, mostly from the consumer packaged goods industry. Perhaps these companies appear all too obvious, and one might wonder what insight we gained at this point. The chances are that even a casual observer of the industry would have mentioned Kimberly-Clark, Colgate-Palmolive, and Johnson & Johnson as businesses operating in the same field as Procter & Gamble. Therefore, one might argue similar stock price movements should be expected.

The key point here is that–without any prior knowledge of this domain–a computer algorithm automatically extracted a structure that is consistent with the understanding of anyone familiar with this domain.

Beyond interpreting the qualitative structure of this network, there is a wide range of functions for gaining insight into this high-dimensional problem domain. For instance, we may wish to know which node within this network is most important. In Chapter 6, we discussed the question in the context of a predictive model, which we learned with Supervised Learning. Here, on the other hand, we learned the network with an Unsupervised Learning algorithm, which means that there is no Target Node. As a result, we need to think about the importance of a node with regard to the entire network, as opposed to a specific Target Node.

We need to introduce a number of new concepts to equip us for the discussion about node importance within a network. We draw on concepts from information theory, which we first introduced in Chapter 5 under Information-Theoretic Concepts.

Arc Force

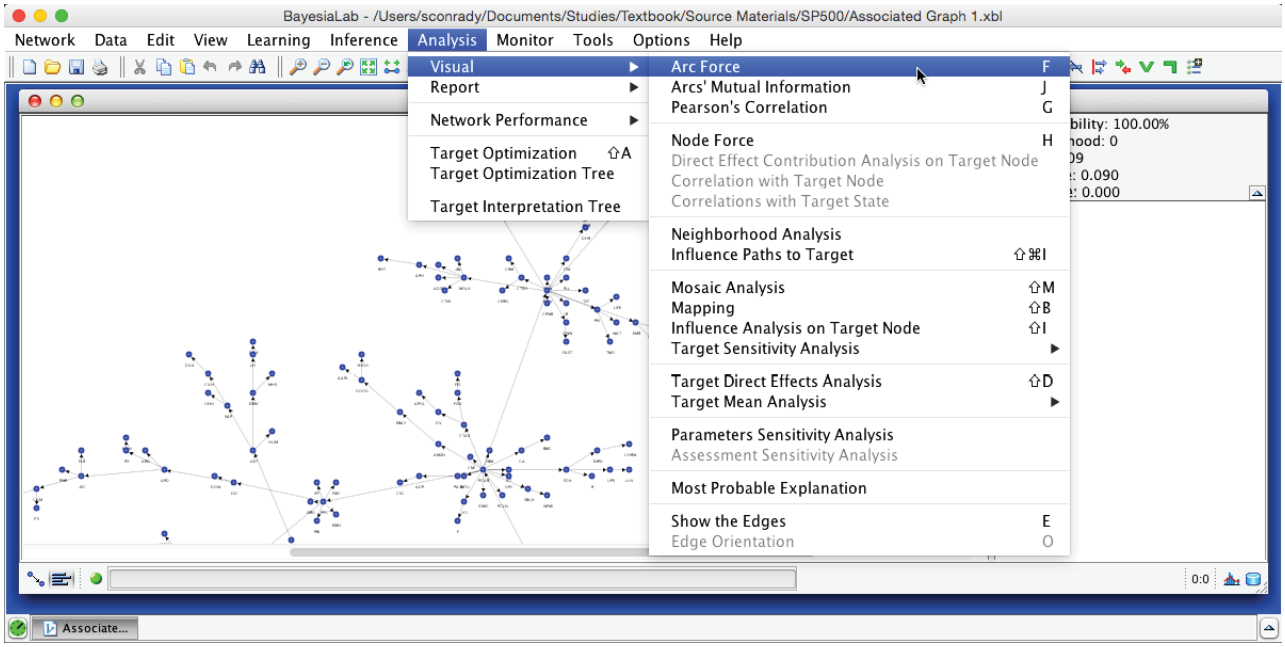
BayesiaLab's Arc Force is computed by using the Kullback-Leibler Divergence, denoted by D_{KL}, which compares two joint probability distributions, P and Q, defined on the same set of variables X.

$$D_{KL}\left(P(X) \parallel Q(X)\right) = \sum_{X} P(X) \log_2 \frac{P(X)}{Q(X)}$$

where P is the current network, and Q is the exact same network as P, except that we removed the arc under study.

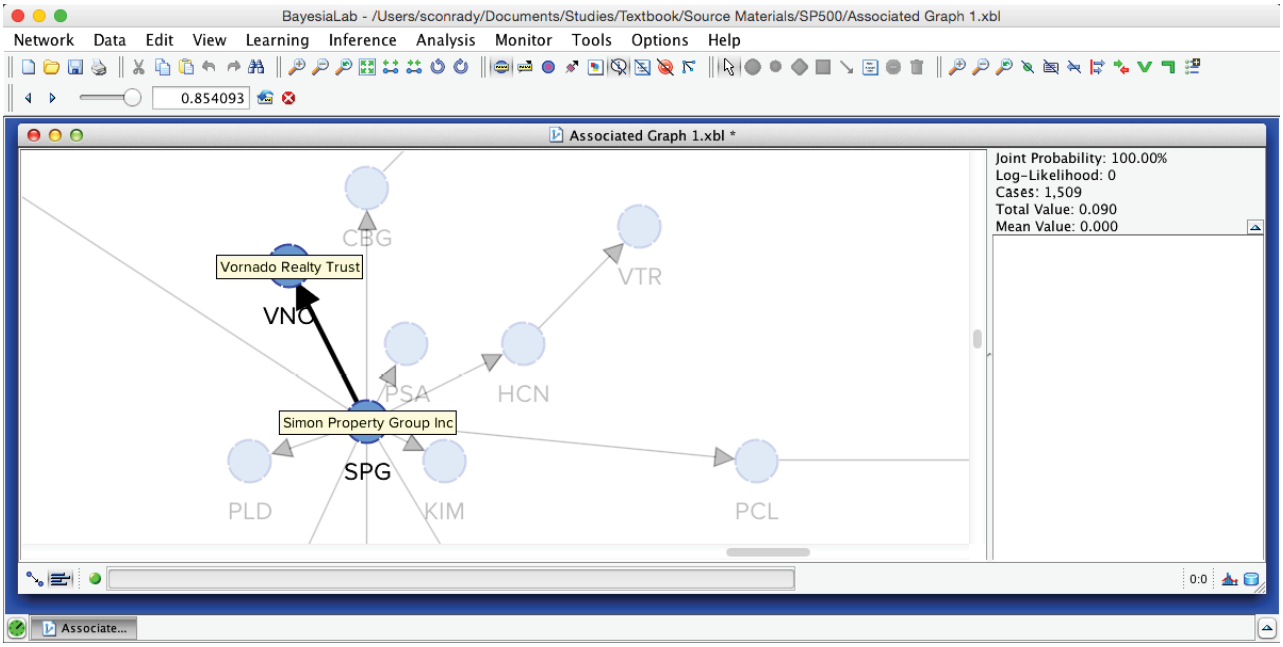
It is important to note that Mutual Information and Arc Force are closely related (see Comparing Mutual Information and Arc Force). If the child node in the pair of nodes under study has no other parents, Mutual Information and Arc Force are, in fact, equivalent. However, the Arc Force is more powerful as a measure as it considers the network's Joint Probability Distribution rather than only focusing on the bivariate relationship.

The Arc Force can be displayed directly on the Bayesian network graph. Upon switching to the Validation Mode  `F5`, we select `Menu > Analysis > Visual > Arc Force`.



Upon activating Arc Force, we can see that the arcs have different thicknesses. Also, an additional control panel becomes available in the menu.

The slider in this control panel allows us to set the Arc Force threshold below which arcs and nodes will be grayed out in the Graph Panel. By default, it is set to 0, which means that the entire network is visible. Using the Previous and Next buttons, we can step through all threshold levels. For instance, by starting at the maximum and then going down one step, we highlight the arc with the strongest Arc Force in this network, which is between SPG (Simon Property Group) and VNO (Vornado Realty Trust).

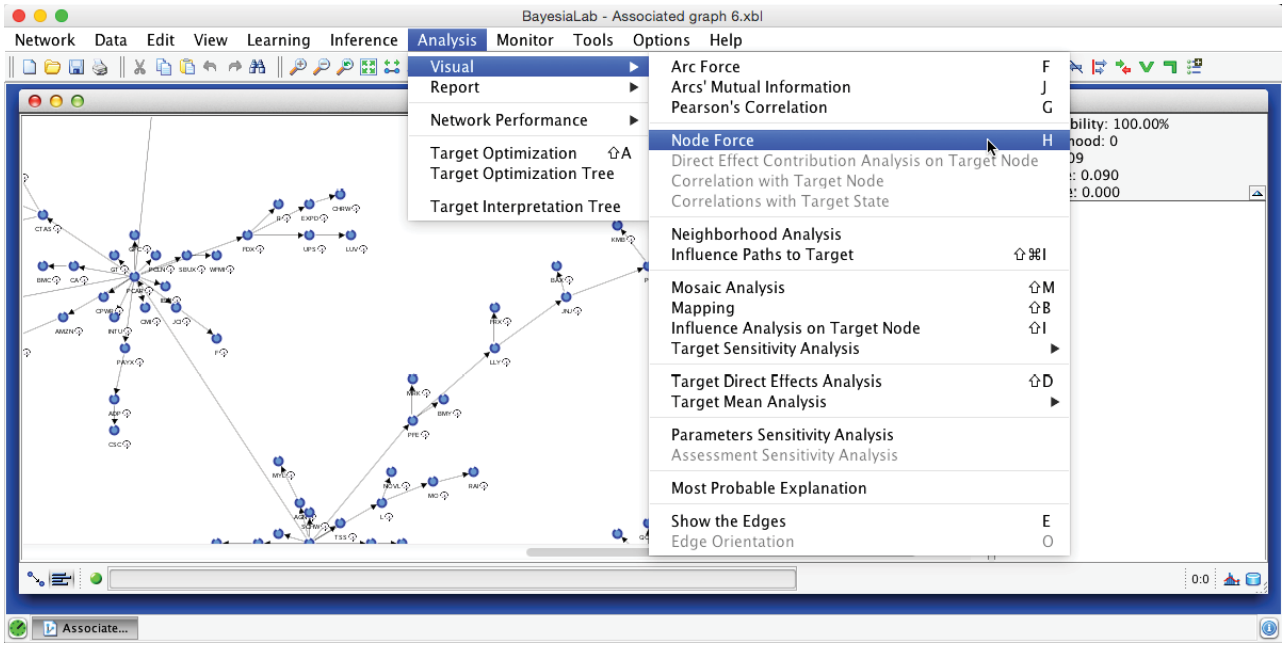


Node Force

The Node Force can be derived directly from the Arc Force. More specifically, there are three types of Node Force in BayesiaLab:

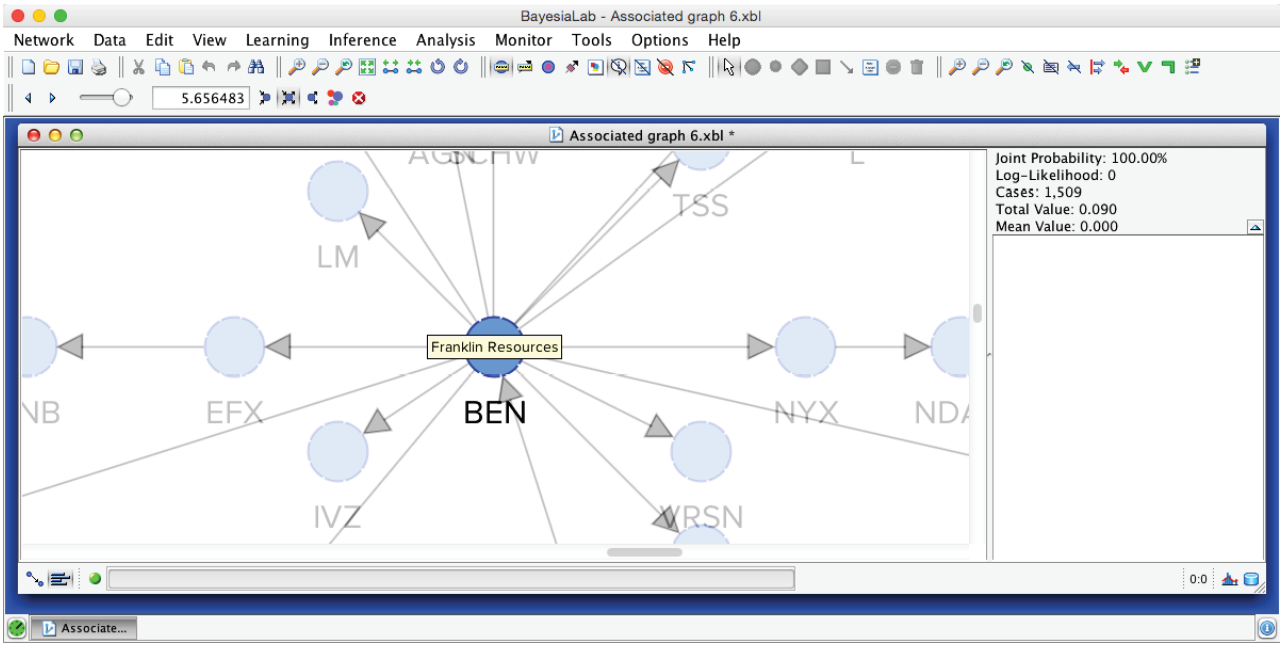
- The **Incoming Node Force** is the sum of the Arc Forces of all incoming arcs.
- The **Outgoing Node Force** is the sum of the Arc Forces of all outgoing arcs.
- The **Total Node Force** is the sum of the Arc Forces of all incoming and outgoing arcs.

The Node Force can be shown directly on the Bayesian network graph. Upon switching to the Validation Mode F5, we select `Analysis > Visual > Node Force`.



After starting Node Force, we have another additional control panel available in the menu.

The slider in this control panel allows us to set the Node Force threshold below which nodes will be grayed out in the Graph Panel. By default, it is set to 0, meaning all nodes are visible. Conversely, by setting the threshold to the maximum, all nodes are grayed out. Using Previous and Next , we can step through the entire range of thresholds. This functionality is analogous to the control panel for Arc Force. For example, by starting at the maximum and then going down one step, we can find the node with the strongest Node Force in this network, which is BEN (Franklin Resources), a global investment management organization.

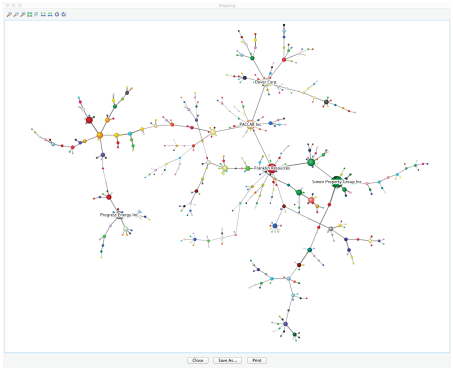


Node Force Mapping

This analysis tool also features a "local" Mapping function, which is particularly useful when dealing with big networks, such as the one in this example with hundreds of nodes. We refer to this as a "local" Mapping function in the sense of only being available in the context of Node Force Analysis, as opposed to the "general" Mapping function, which is always available within the Validation Mode F5 as a standalone analysis tool (`Menu > Analysis > Visual > Mapping`).

We launch the `Mapping` window by clicking the Mapping icon on the control panel to the right of the slider. In this network view, the size of the nodes is directly proportional to the selected type of Node Force (Incoming, Outgoing, Total). The width of the links is proportional to the Arc Force. Changing the threshold values (with the slider, for example) automatically updates the view.

Choosing `Static Font Size` from the Context Menu and then, for instance, reducing the threshold by four more steps reveals the five strongest nodes while maintaining an overview of the entire network.



Chapter 8: Probabilistic Structural Equation Models for Key Driver Analysis

Structural Equation Modeling (SEM) is a statistical technique for estimating and testing causal relationships by combining observed data with qualitative causal assumptions. The foundations of SEM were established through the work of Sewall Wright (1921), Trygve Haavelmo (1943), and Herbert Simon (1953), and the framework was formally unified and extended by Judea Pearl (2000). SEMs support both confirmatory and exploratory modeling, making them suitable for validating existing theories as well as generating new ones.

In BayesiaLab, Probabilistic Structural Equation Models (PSEMs) serve a conceptually similar purpose to traditional SEMs but are constructed on the foundation of Bayesian networks rather than systems of equations. While SEMs typically require a high level of statistical expertise and involve numerous manual steps, PSEMs are designed to be more accessible–particularly to subject matter experts without advanced statistical training. Moreover, the PSEM modeling process in BayesiaLab is significantly faster and more efficient, often reducing modeling time by several orders of magnitude.

Once validated, a PSEM can be used like any other Bayesian network within BayesiaLab. This enables users to apply a full suite of analytical, simulation, and optimization tools, thereby maximizing the utility of the causal knowledge embedded in the model.

Example: Consumer Survey

This chapter introduces a prototypical application of Probabilistic Structural Equation Modeling (PSEM) focused on key driver analysis and product optimization using consumer survey data. The analysis explores how consumers perceive various product attributes and how these perceptions influence their purchase intent.

To address the uncertainty inherent in survey data, the study incorporates "latent" variables–underlying constructs not directly captured in the survey responses. These are derived from patterns among the "manifest" variables, which are the directly observed survey items. Incorporating latent variables enables the construction of models that are more robust and interpretable than those relying solely on manifest data.

The overarching goal is to enhance the interpretability of survey results for researchers and increase their practical relevance for decision-makers. Ultimately, the PSEM aims to guide the prioritization of marketing and product strategies to maximize consumer purchase intent.

Dataset

This study is based on a monadic consumer survey about perfumes conducted by a market research agency in France. In this study, each respondent evaluated only one perfume. In this example, we use survey responses from 1,320 women who evaluated 11 fragrances (representative of the French market) on a wide range of attributes.

- 27 ratings on fragrance-related attributes, such as Sweet, *Flowery*, *Feminine*, etc., measured on a 1-10 scale.
- 12 ratings with regard to imagery about someone who wears the respective fragrance, e.g. Sexy, Modern, measured on a 1-10 scale.
- 1 variable for *Intensity*, a measure reflecting the level of intensity, measured on a 1-5 scale. The variable *Intensity* is listed separately due to the a priori knowledge of its non-linearity and the existence of a "just-about-right" level.
- 1 variable for *Purchase Intent*, measured on a 1-6 scale.
- 1 nominal variable, *Product*, for product identification.

Workflow Overview

A PSEM is a hierarchical Bayesian network that can be generated through a series of machine-learning and analysis tasks:

- All relationships in a PSEM are probabilistic–hence the name, as opposed to having deterministic relationships plus error terms in traditional SEMs.
- PSEMs are nonparametric, which facilitates the representation of nonlinear relationships plus relationships between categorical variables.
- The structure of PSEMs is partially or fully machine-learned from data.
- Unsupervised Learning to discover the strongest relationships between the manifest variables.
- Variable Clustering, based on the learned Bayesian network, to identify groups of variables that are strongly connected.
- Multiple Clustering: we consider the strong intra-cluster connections identified in the Variable Clustering step to be due to a "hidden common cause." For each cluster of variables, we use Data Clustering–on the variables within the cluster only–to induce a latent variable representing the hidden cause.
- Unsupervised Learning to find the interrelations between the newly-created latent variables and their relationships with the Target Node.

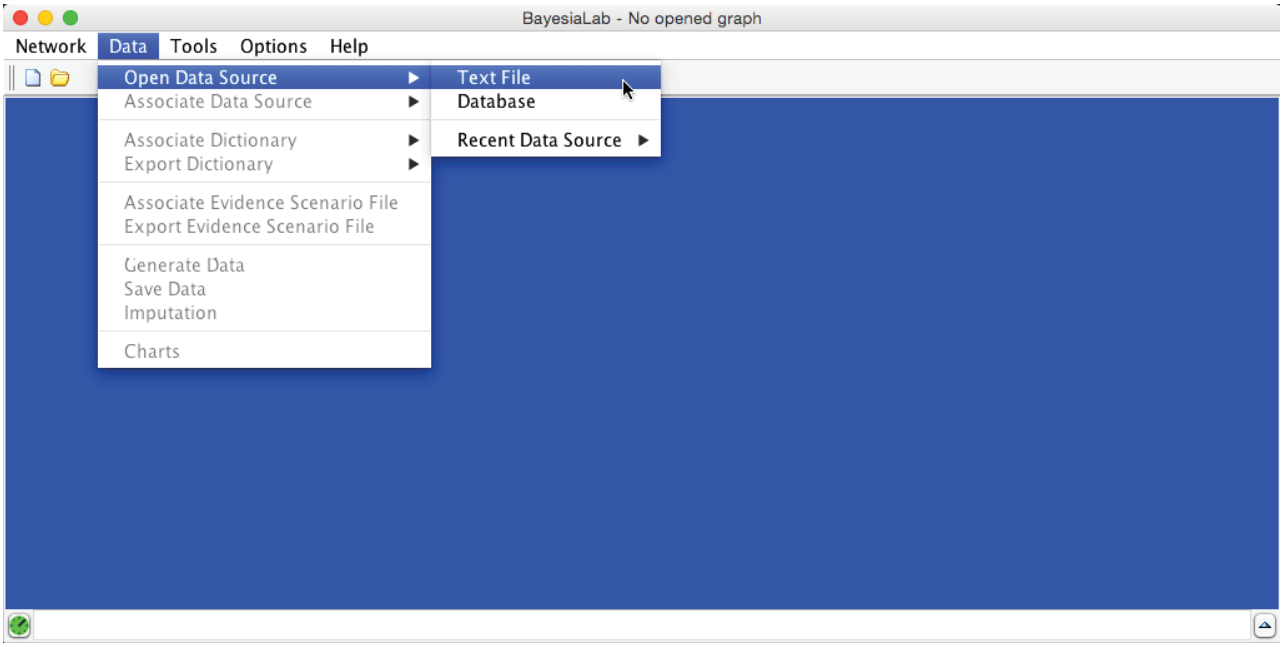
Workflow Details

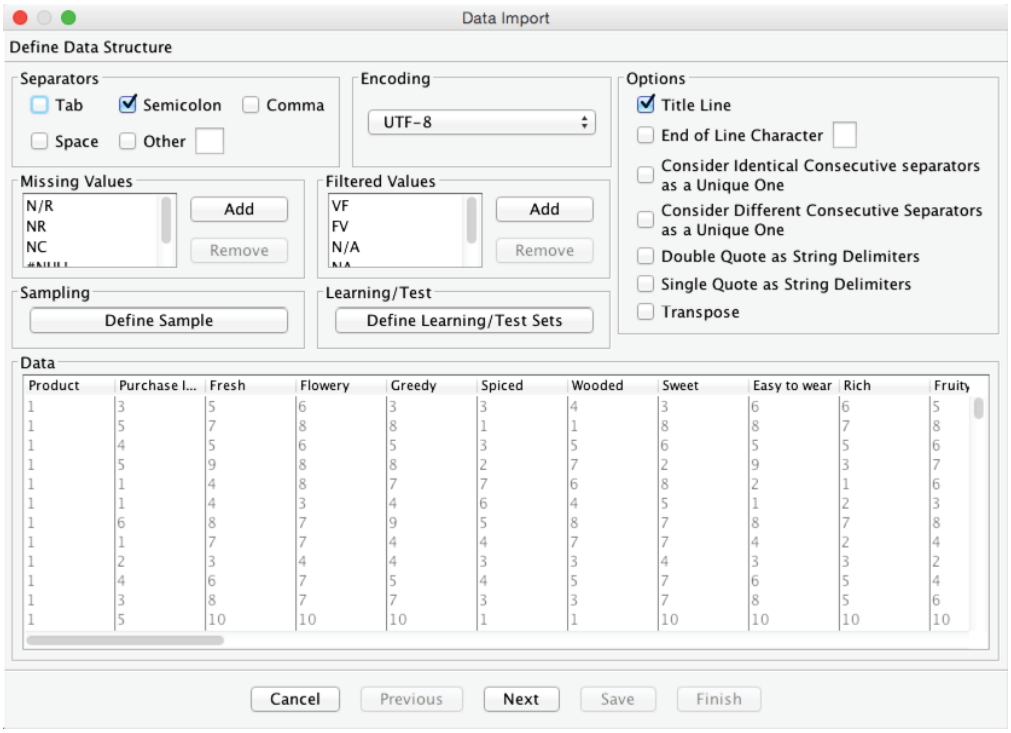
- Data Import
- Step 1: Unsupervised Learning
- Step 2: Variable Clustering
- Step 3: Multiple Clustering
- Step 4: Completing the Probabilistic Structural Equation Model
- Key Driver Analysis
- Product Optimization

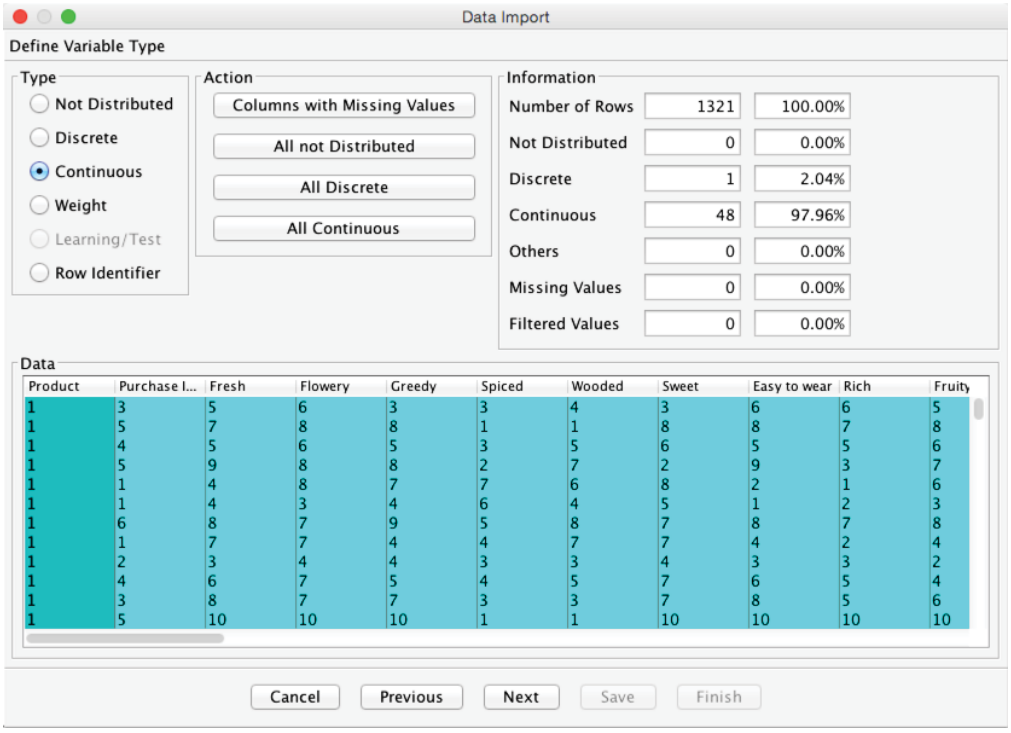
Data Import

We have already described all the steps of the Data Import Wizard in previous chapters. Therefore, we present most of the following screenshots without commentary and only highlight items specific to this example. To start the Data Import Wizard, we open the file:

Download: `Perfume.csv`

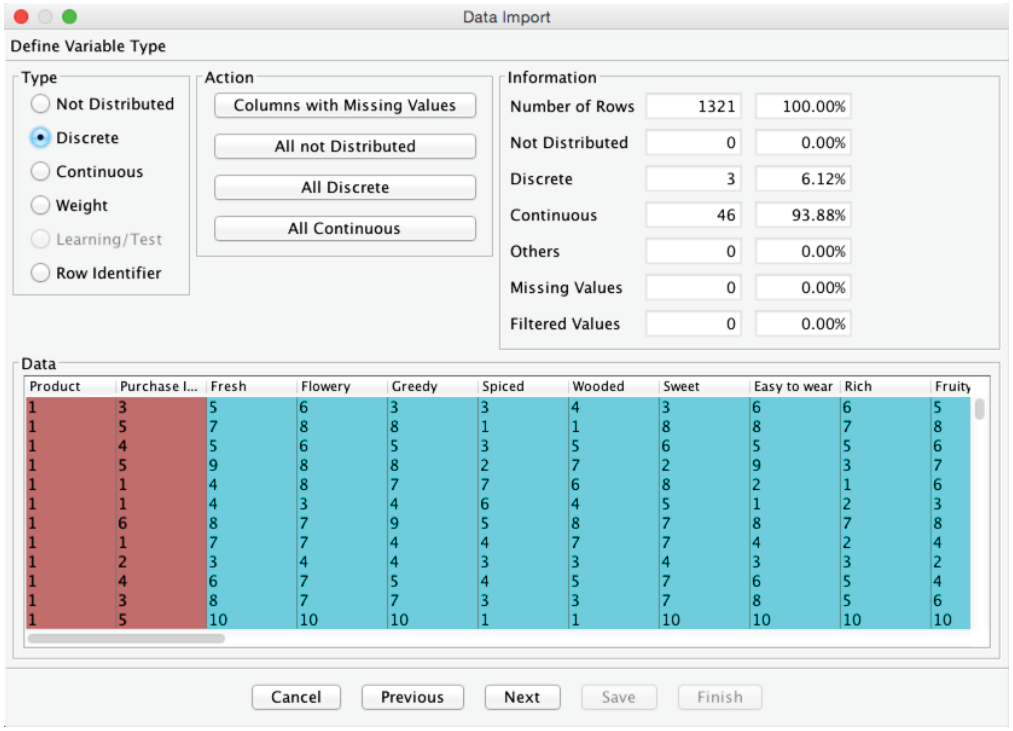


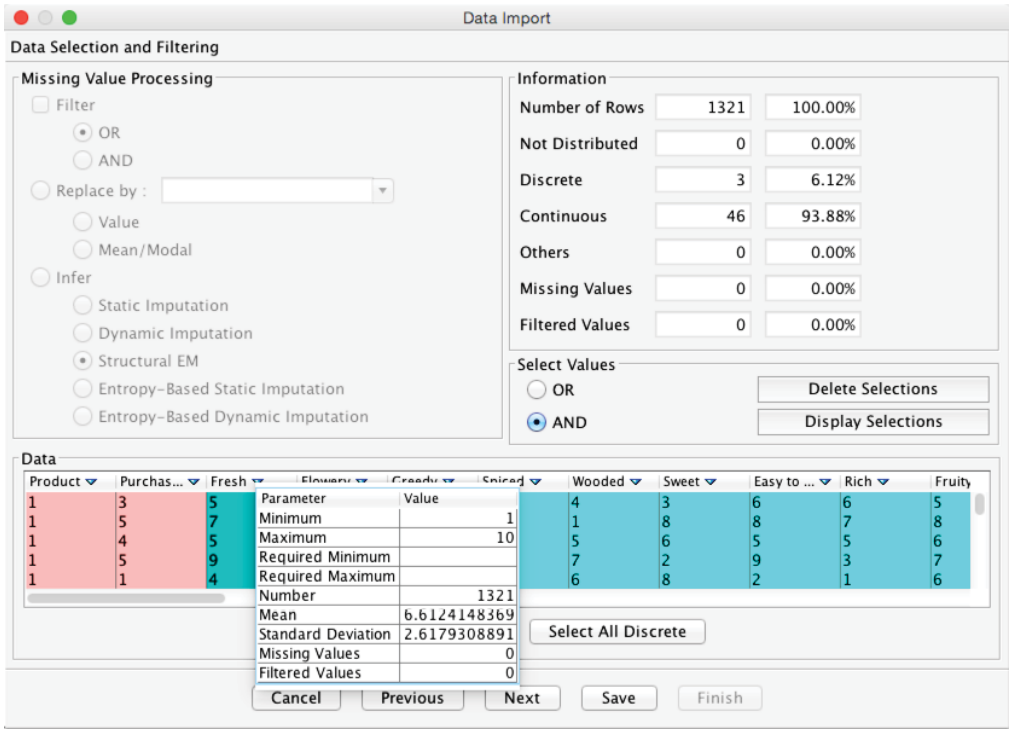




For this example, we need to override the default data type for the variable named *Product* as it is a nominal product identifier rather than a numerical value. We can change this variable's data type

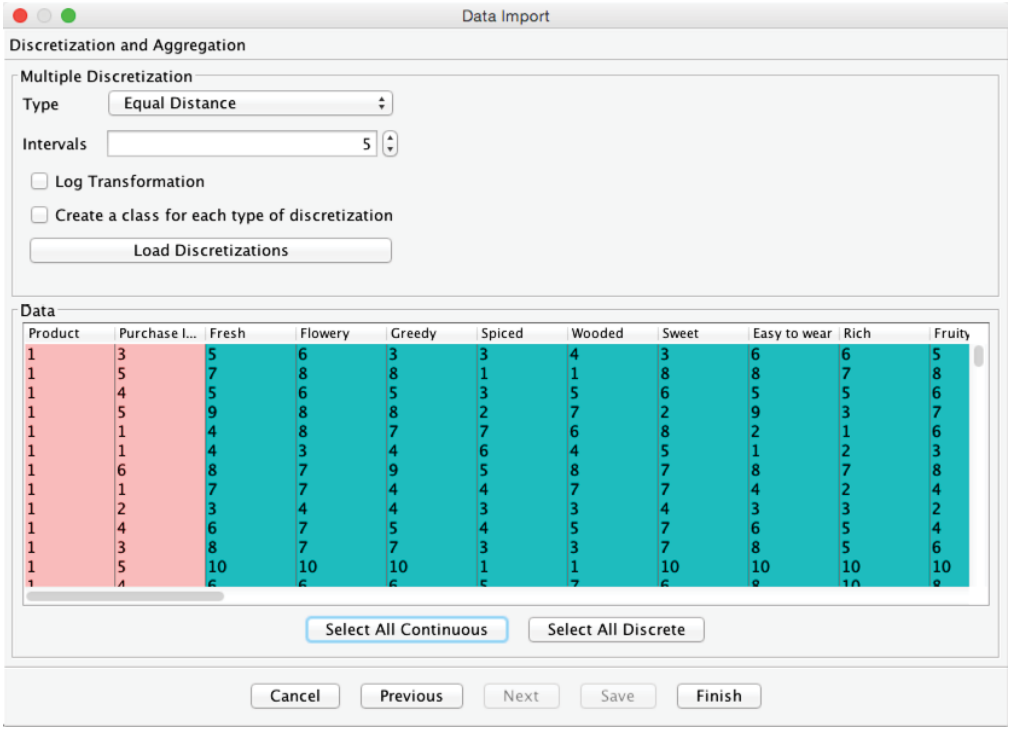
by highlighting the *Product* column and clicking the Discrete radio button. This changes the color of the *Product* column to red. We also define *Purchase Intent* and *Intensity* as Discrete variables. Their number of states is suitable for our purposes.





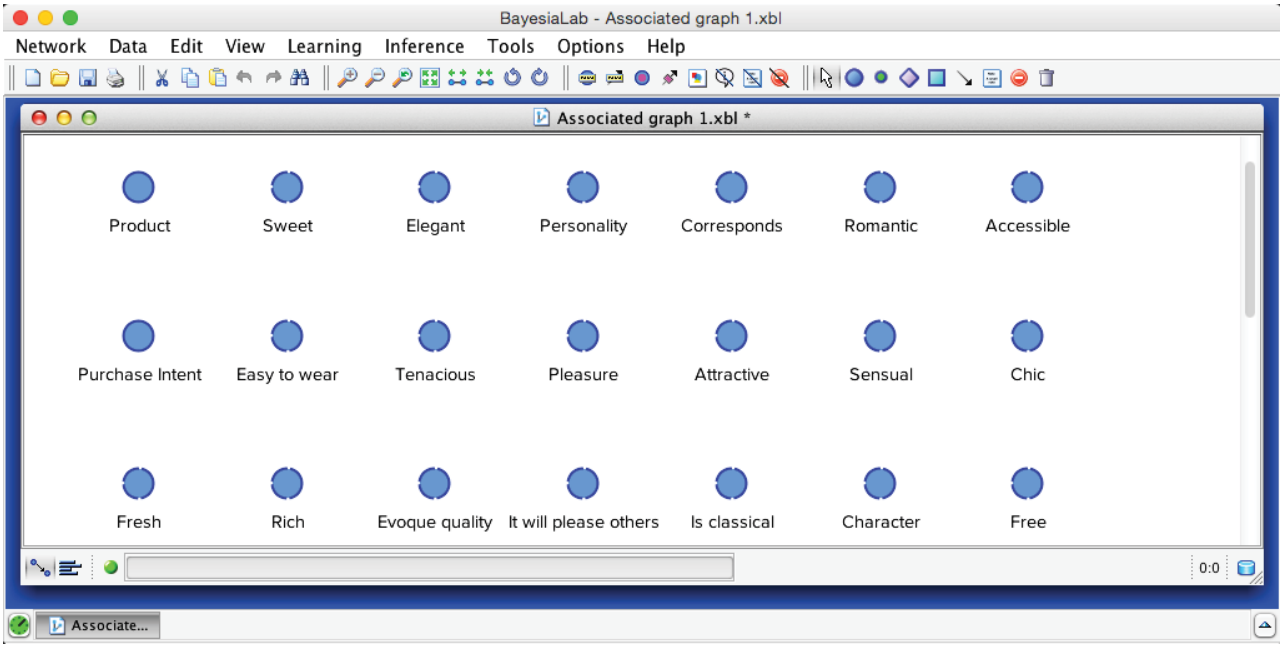
The next step is the Discretization and Aggregation screen. Given the number of observations, it is appropriate to reduce the number of states of the ratings from the original 10 states (1-10) to a smaller number. All these variables measure satisfaction on the same scale, i.e., from 1 to 10. Following our earlier recommendations (see Discretization Intervals in Chapter 6), the best choice in this context is the Equal Distance discretization.

By clicking `Select All Continuous`, we highlight all to-be-discretized variables. Then, we choose the type of discretization to be applied, which is Equal Distance. Furthermore, given the number of observations, we choose 5 bins for the discretization.



Clicking `Finish` finalizes the import process. Upon completion, we are prompted whether we want to view the Import Report.

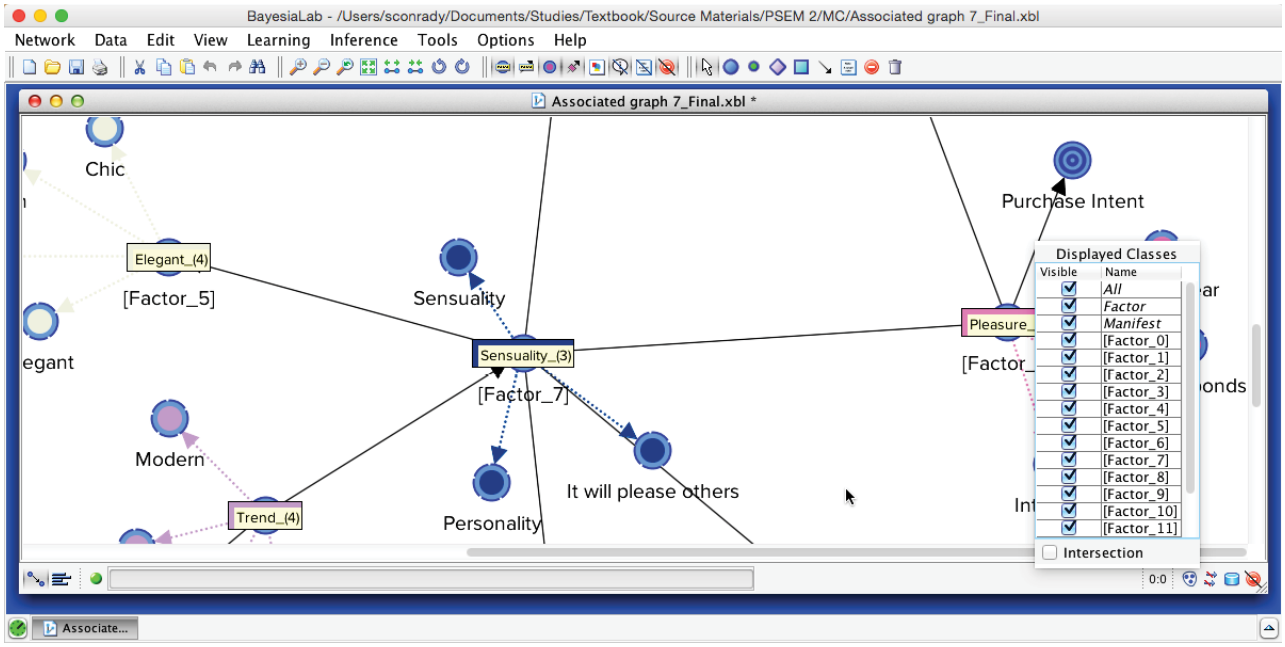
As there is no uncertainty with regard to the outcome of the discretization, we decline and automatically obtain a fully unconnected network with 49 nodes.



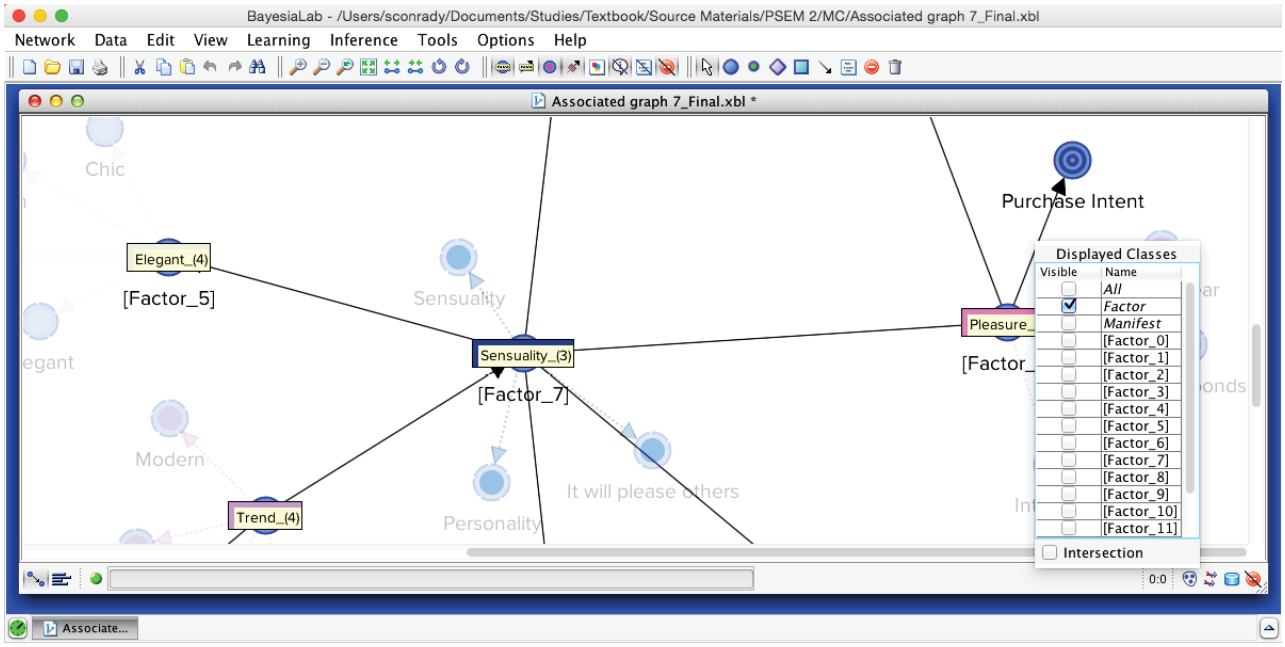
Key Driver Analysis

Our Probabilistic Structural Equation Model is now complete, and we can use it to perform the analysis part of this exercise, namely to find out what "drives" *Purchase Intent*. We return to the Validation Mode .

To understand the relationship between the factors and *Purchase Intent*, we want to "tune out" all Manifest variables for now. We can do so by right-clicking the Classes icon in the bottom right corner of the Graph Panel window. This brings up a list of all Classes. By default, all are checked and thus visible.



For our purposes, we want to un-check `All` and then only check the class `Factor`.



In the resulting view, all the Manifest Nodes are transparent, so the relationship between the Factors becomes visually more prominent. By de-selecting the Manifest Nodes in this way, we also exclude them from the following visual analysis.

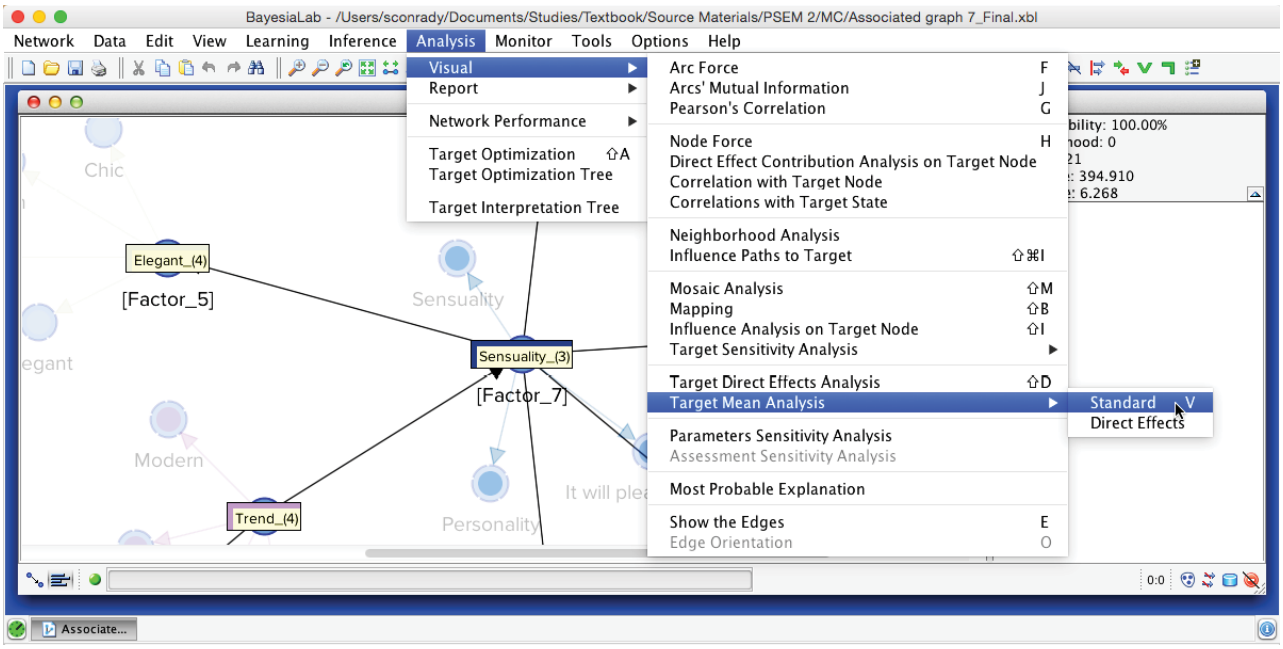
Target Analysis

In line with our objective of learning about the key drivers in this domain, we proceed to analyze the association of the newly created Factors with *Purchase Intent*.

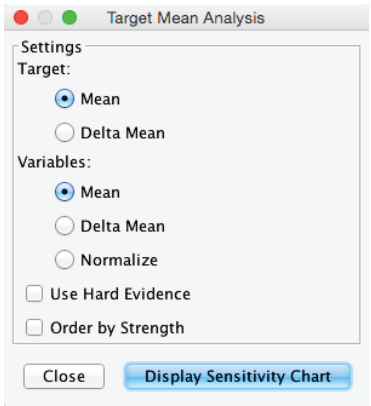
We return to the Validation Mode, in which we can use two approaches to learn about the relationships between Factors and the Target Node: we first perform a visual analysis and then generate a report in table format.

Visual Analysis

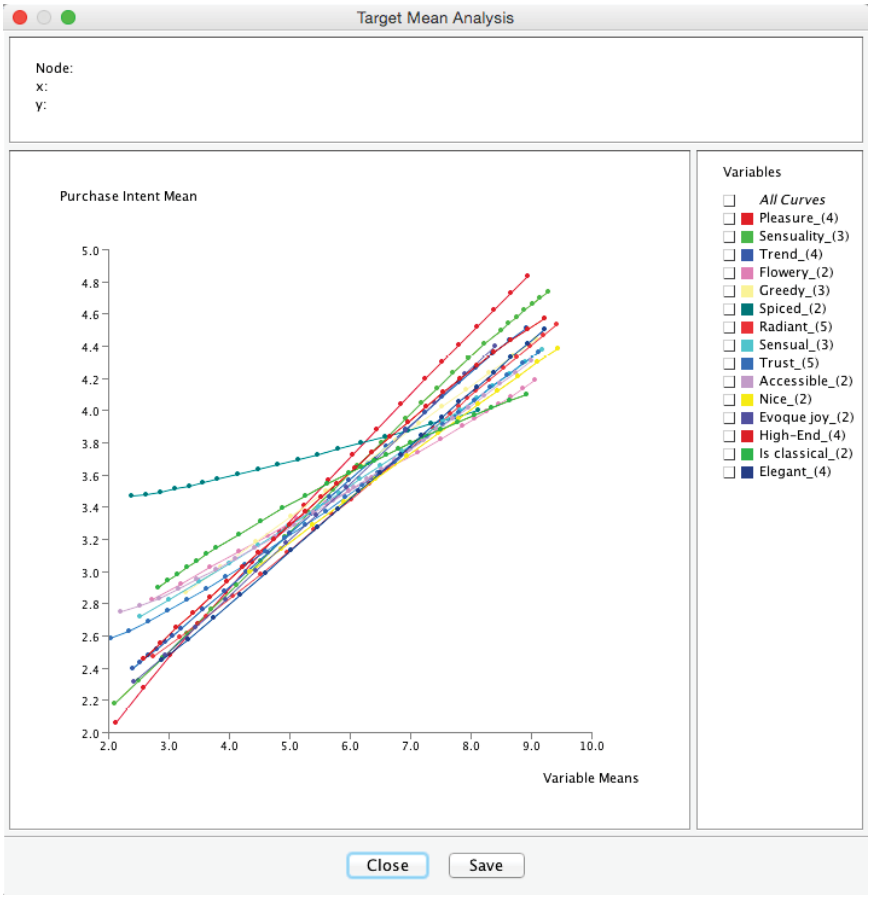
We initiate the visual analysis by selecting `Menu > Analysis > Visual > Target Mean Analysis > Standard`:



This brings up a dialog box with the options shown below. Given the context, selecting `Mean` for both the Target Node and the Nodes is appropriate.

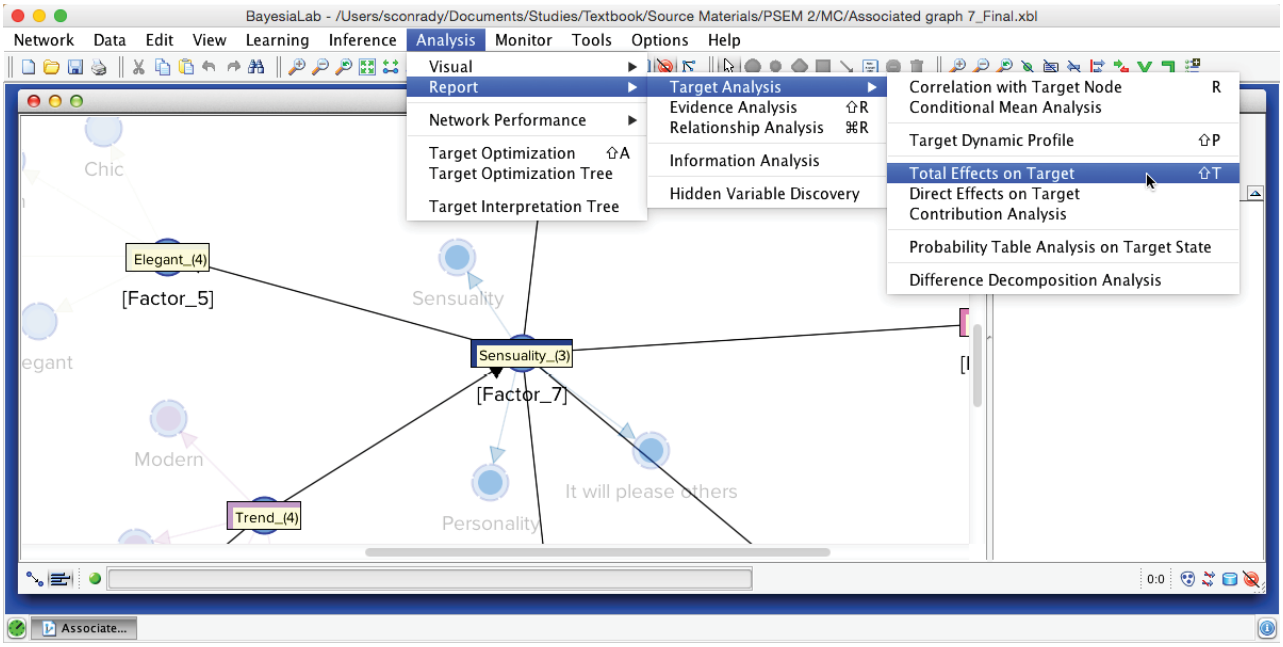


Upon clicking `Display Sensitivity Chart`, the resulting plot shows the response curves of the Target Node as a function of the values of the Factors. This allows an immediate interpretation of the strength of the associations.



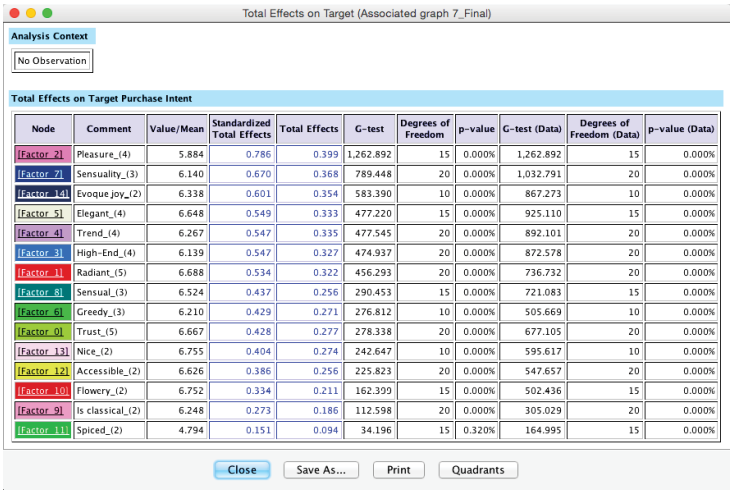
Target Analysis Report

As an alternative to the visual analysis, we now run the Target Analysis Report: `Menu > Analysis > Report > Target Analysis > Total Effects on Target`.



> Although "effect" carries a causal connotation, we must emphasize that we strictly examine associations. This means that we perform observational inference as we generate this report.

A new window opens up to present the report. Under `Menu > Options > Settings > Reporting`, we can check `Display the Node Comments in Tables` so that Node Comments appear in addition to the Node Names in all reports.



Node	Comment	Value/Mean	Standardized Total Effects	Total Effects	G-test	Degrees of Freedom	p-value	G-test (Data)	Degrees of Freedom (Data)	p-value (Data)
[Factor_2]	Pleasure_(4)	5.884	0.786	0.399	1,262.892	15	0.000%	1,262.892	15	0.000%
[Factor_7]	Sensuality_(3)	6.140	0.670	0.368	789.448	20	0.000%	1,032.791	20	0.000%
[Factor_14]	Evoque joy_(2)	6.338	0.601	0.354	583.390	10	0.000%	867.273	10	0.000%
[Factor_5]	Elegant_(4)	6.648	0.549	0.333	477.220	15	0.000%	925.110	15	0.000%
[Factor_4]	Trend_(4)	6.267	0.547	0.335	477.545	20	0.000%	892.101	20	0.000%
[Factor_3]	High-End_(4)	6.139	0.547	0.327	474.937	20	0.000%	872.578	20	0.000%
[Factor_1]	Radiant_(5)	6.688	0.534	0.322	456.293	20	0.000%	736.732	20	0.000%
[Factor_8]	Sensual_(3)	6.524	0.437	0.256	290.453	15	0.000%	721.083	15	0.000%
[Factor_6]	Greedy_(3)	6.210	0.429	0.271	276.812	10	0.000%	505.669	10	0.000%
[Factor_0]	Trust_(5)	6.667	0.428	0.277	278.338	20	0.000%	677.105	20	0.000%
[Factor_13]	Nice_(2)	6.755	0.404	0.274	242.647	10	0.000%	595.617	10	0.000%
[Factor_12]	Accessible_(2)	6.626	0.386	0.256	225.823	20	0.000%	547.657	20	0.000%
[Factor_10]	Flowery_(2)	6.752	0.334	0.211	162.399	15	0.000%	502.436	15	0.000%
[Factor_9]	Is classical_(2)	6.248	0.273	0.186	112.598	20	0.000%	305.029	20	0.000%
[Factor_11]	Spiced_(2)	4.794	0.151	0.094	34.196	15	0.320%	164.995	15	0.000%

The Total Effect (TE) is estimated as the derivative of the Target Node with respect to the driver node under study.

$$TE(X,Y) = \frac{\delta_Y}{\delta_X}$$

where X is the analyzed variable and Y is the Target Node. The Total Effect represents the change in the mean of the Target Node associated with — and not necessarily caused by — a small modification of the mean of a driver node. The Total Effect is the ratio of these two values.

This way of measuring the effect of the Factors on the Target Node assumes the relationships to be locally linear. Even though this is not always a correct assumption, it can be reasonable to simulate small changes in satisfaction levels.

As per the report, [Factor_2] provides the strongest Total Effect with a value of 0.399. This means that observing an increase of one unit in the level of the concept represented by [Factor_2] predicts a posterior probability distribution of *Purchase Intent* that has an expected value that is 0.399 higher compared to the marginal value.

The Standardized Total Effect (STE) is also displayed. It represents the Total Effect multiplied by the ratio of the standard deviation of the driver node and the standard deviation of the Target Node.

$$STE(X,Y) = \frac{\delta_Y}{\delta_X} \times \frac{\sigma_X}{\sigma_Y}$$

This means that Standardized Total Effect takes into account the "potential" of the driver under study.

In the report, the results are sorted by the Standardized Total Effect in descending order. This immediately highlights the order of importance of the Factors relative to the Target Node, *Purchase Intent*.

Independence Tests

In the columns further to the right in the report, the results of independence tests between the nodes are reported:

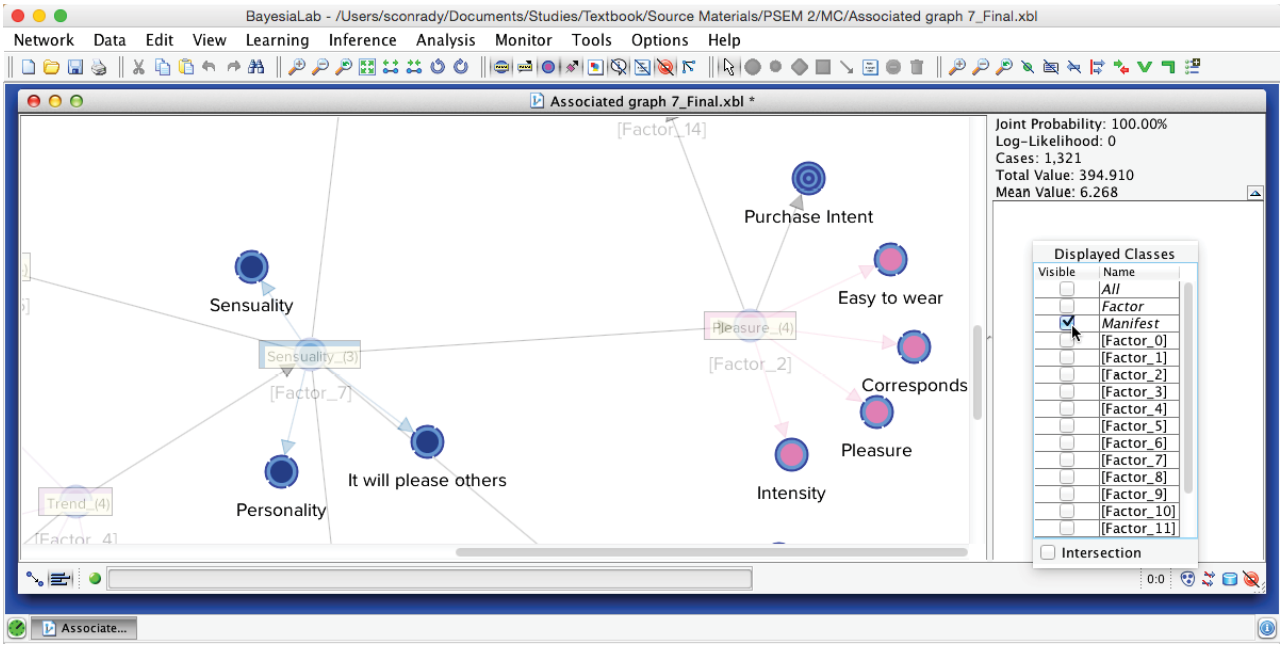
- Chi-Square (χ^2) test or G-test: The independence test is computed on the basis of the network between each driver node and the target variable. It is possible to change the type of independence from the Chi-Square (χ^2) test to the G-test via `Menu > Options > Settings > Statistical Tools`.
- Degree of Freedom: Indicates the degree of freedom between each driver node and the Target Node in the network.
- p-value: the p-value is the probability of observing a value as extreme as the test statistic by chance.

If a dataset is associated with the network, as is the case here, the independence test, the degrees of freedom, and the p-value are also computed directly from the underlying data.

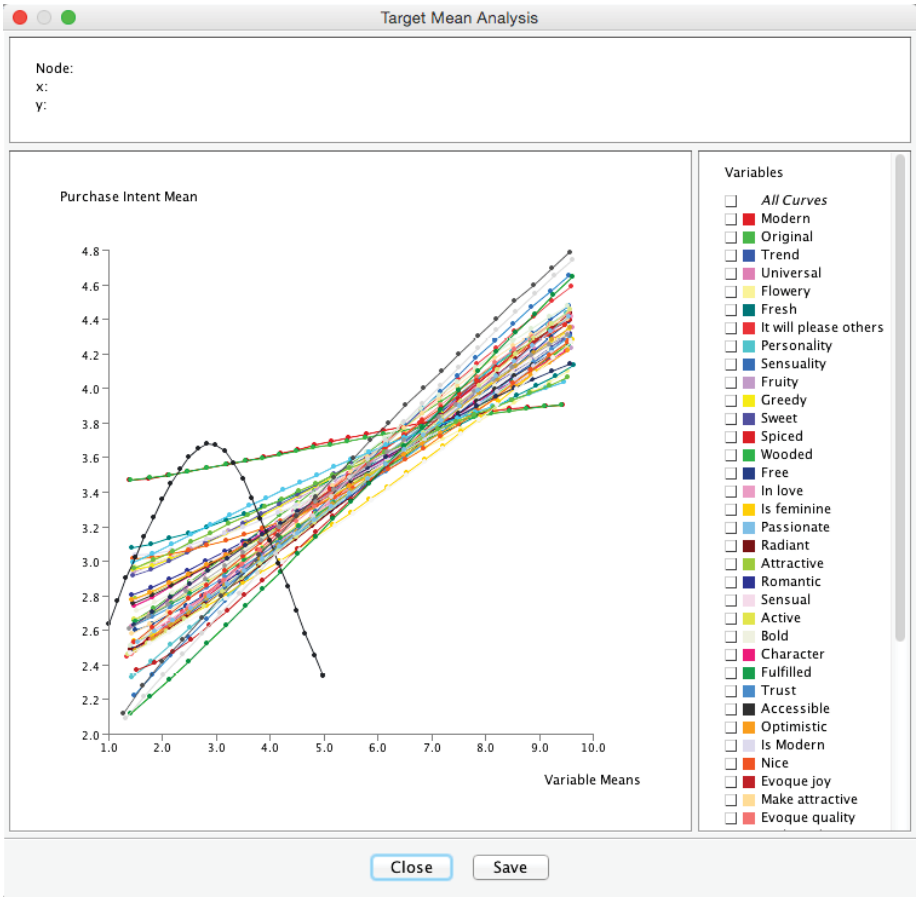
Factors versus Manifest Nodes

For overall interpretation purposes, looking at Factor-level drivers can be illuminating. Often, it provides a useful big-picture view of the domain. However, we need to consider the Manifest-level drivers to identify specific product actions. As pointed out earlier, the Factor-level drivers only exist as theoretical constructs, which cannot be directly observed in data. As a result, changing the Factor nodes requires manipulating the underlying Manifest nodes. For this reason, we now switch back our

view of the network in order to only consider the Manifest nodes in the analysis. We do that by right-clicking the Classes icon in the bottom right corner of the `Graph Panel` window. This brings up the list of all Classes, of which we only check the Class Manifest. Now, all Factors are translucent and excluded from the analysis.



We repeat both the Target Mean Analysis and the Total Effects on Target report.

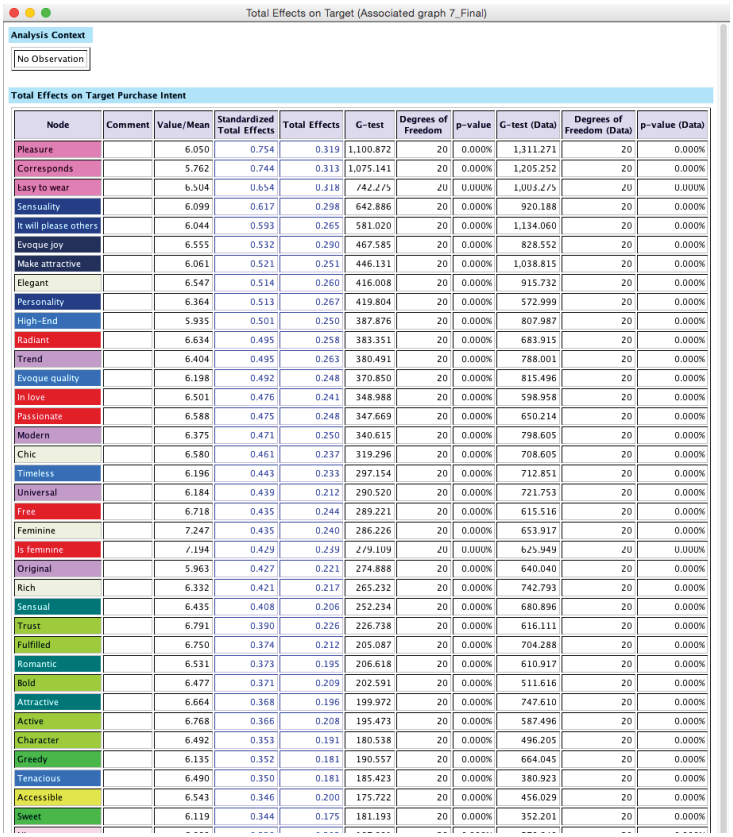


Not surprisingly, the Manifest Nodes show a similar pattern of association as the Factors. However, there is one important exception: the Manifest Node *Intensity* shows a nonlinear relationship with *Purchase Intent*. The curve for *Intensity* is shown with a gray line. Note that by hovering over a curve or a node name, BayesiaLab highlights the corresponding item in the legend or the plot.

Also, we can see that *Intensity* was recorded on a 1-5 scale rather than the 1-10 scale that applies to the other nodes. *Intensity* is a so-called "JAR" variable, i.e., a variable that has a "just-about-right" value. In the context of perfumes, this characteristic is obvious. A fragrance that is either too strong or too light is undesirable. Instead, there is a value somewhere in between that would make a fragrance most attractive. The JAR characteristic is prototypical for variables representing sensory dimensions, e.g., saltiness or sweetness.

This emphasizes the importance of visual analysis, as the nonlinearity goes unnoticed in the Total Effects on Target report. In fact, it drops almost to the bottom of the list in the report.

It turns out to be rather difficult to optimize a JAR-type variable at a population level. For example, increasing *Intensity* would reduce the number of consumers who find the fragrance too subtle. On the other hand, an increase in *Intensity* would presumably dismay some consumers who believed the original *Intensity* level to be appropriate.



Total Effects on Target (Associated graph 7_Final)

Analysis Context

No Observation

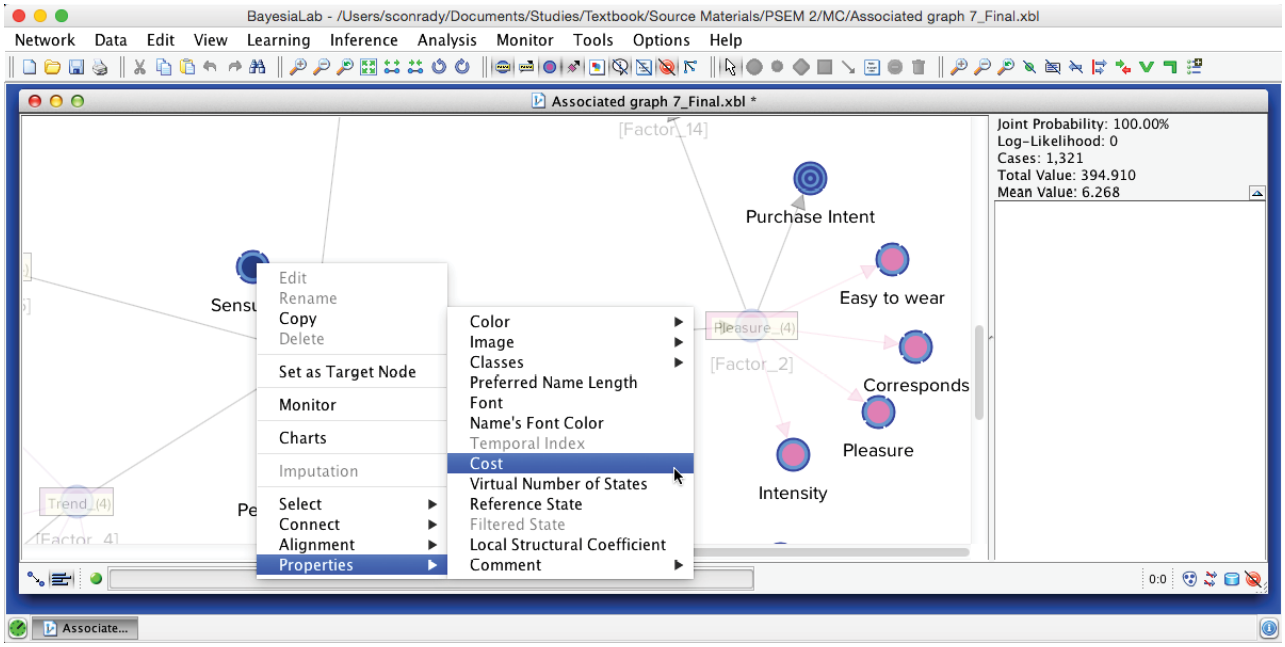
Total Effects on Target Purchase Intent

Node	Comment	Value/Mean	Standardized Total Effects	Total Effects	G-test	Degrees of Freedom	p-value	G-test (Data)	Degrees of Freedom (Data)	p-value (Data)
Pleasure		6.050	0.754	0.319	1,100.872	20	0.000%	1,311.271	20	0.000%
Corresponds		5.762	0.744	0.313	1,075.141	20	0.000%	1,205.252	20	0.000%
Easy to wear		6.504	0.654	0.318	742.275	20	0.000%	1,003.275	20	0.000%
Sensuality		6.099	0.617	0.298	642.886	20	0.000%	920.188	20	0.000%
It will please others		6.044	0.593	0.265	581.020	20	0.000%	1,134.060	20	0.000%
Evoque joy		6.555	0.532	0.290	467.585	20	0.000%	828.552	20	0.000%
Make attractive		6.061	0.521	0.251	446.131	20	0.000%	1,038.815	20	0.000%
Elegant		6.547	0.514	0.260	416.008	20	0.000%	915.732	20	0.000%
Personality		6.364	0.513	0.267	419.804	20	0.000%	572.999	20	0.000%
High-End		5.935	0.501	0.250	387.876	20	0.000%	807.987	20	0.000%
Radiant		6.634	0.495	0.258	383.351	20	0.000%	683.915	20	0.000%
Trend		6.404	0.495	0.263	380.491	20	0.000%	788.001	20	0.000%
Evoque quality		6.198	0.492	0.248	370.850	20	0.000%	815.496	20	0.000%
In love		6.501	0.476	0.241	348.988	20	0.000%	598.958	20	0.000%
Passionate		6.588	0.475	0.248	347.669	20	0.000%	650.214	20	0.000%
Modern		6.375	0.471	0.250	340.615	20	0.000%	798.605	20	0.000%
Chic		6.580	0.461	0.237	319.296	20	0.000%	708.605	20	0.000%
Timeless		6.196	0.443	0.233	297.154	20	0.000%	712.851	20	0.000%
Universal		6.184	0.439	0.212	290.520	20	0.000%	721.753	20	0.000%
Free		6.718	0.435	0.244	289.221	20	0.000%	615.516	20	0.000%
Feminine		7.247	0.435	0.240	286.226	20	0.000%	653.917	20	0.000%
Is feminine		7.194	0.429	0.239	279.109	20	0.000%	625.949	20	0.000%
Original		5.963	0.427	0.221	274.888	20	0.000%	640.040	20	0.000%
Rich		6.332	0.421	0.217	265.232	20	0.000%	742.793	20	0.000%
Sensual		6.435	0.408	0.206	252.234	20	0.000%	680.896	20	0.000%
Trust		6.791	0.390	0.226	226.738	20	0.000%	616.111	20	0.000%
Fulfilled		6.750	0.374	0.212	205.087	20	0.000%	704.288	20	0.000%
Romantic		6.531	0.373	0.195	206.618	20	0.000%	610.917	20	0.000%
Bold		6.477	0.371	0.209	202.591	20	0.000%	511.616	20	0.000%
Attractive		6.664	0.368	0.196	199.972	20	0.000%	747.610	20	0.000%
Active		6.768	0.366	0.208	195.473	20	0.000%	587.496	20	0.000%
Character		6.492	0.353	0.191	180.538	20	0.000%	496.205	20	0.000%
Greedy		6.135	0.352	0.181	190.557	20	0.000%	664.045	20	0.000%
Tenacious		6.490	0.350	0.181	185.423	20	0.000%	380.923	20	0.000%
Accessible		6.543	0.346	0.200	175.722	20	0.000%	456.029	20	0.000%
Sweet		6.119	0.344	0.175	181.193	20	0.000%	352.201	20	0.000%

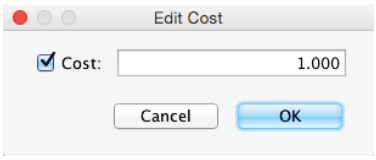
Constraints via Costs

As this driver analysis model is intended for product optimization, we need to consider any possible real-world constraints that may limit our ability to optimize any of the drivers in this domain. For instance, a perfumer may know how to change the intensity of a perfume but may not know how to directly affect the perception of "pleasure." In the original study, a number of such constraints were given.

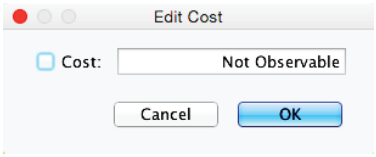
In BayesiaLab, we can conveniently encode constraints via Costs, which is a Node Property. More specifically, we can declare any node as Not Observable, which — in this context — means that they cannot be considered with regard to optimization. Costs can be set by right-clicking on an individual node and then selecting `Properties > Cost`.



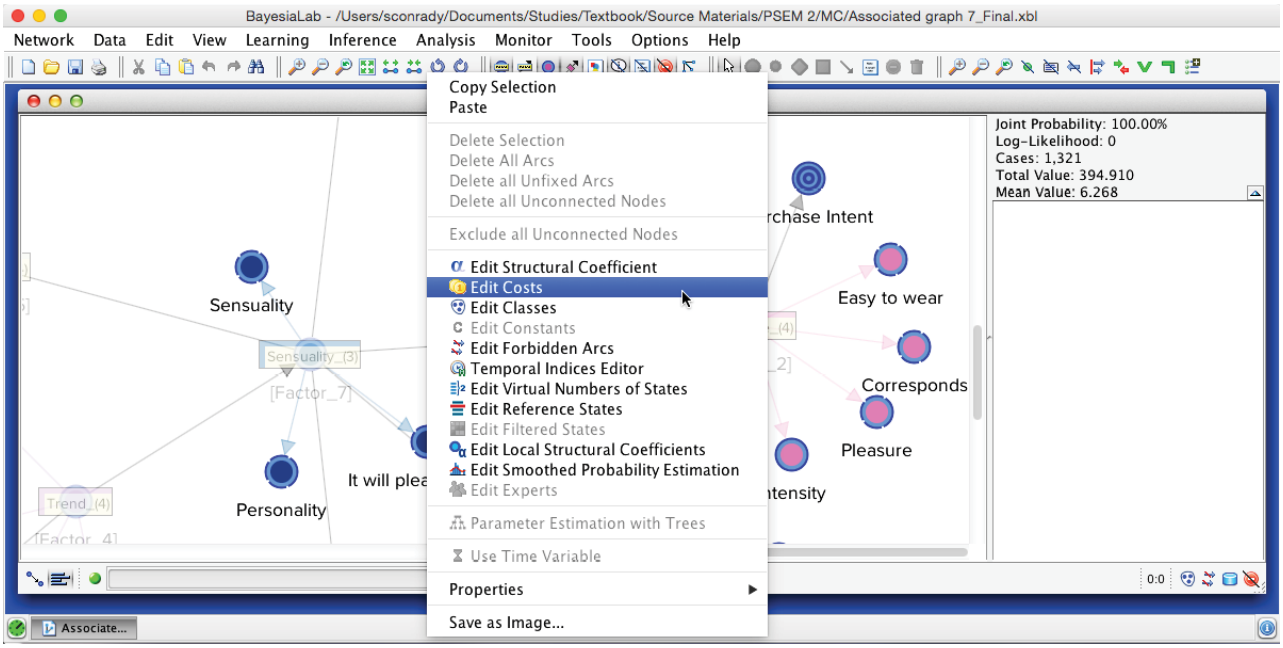
This brings up the Cost Editor for an individual node. By default, all nodes have a cost of 1.



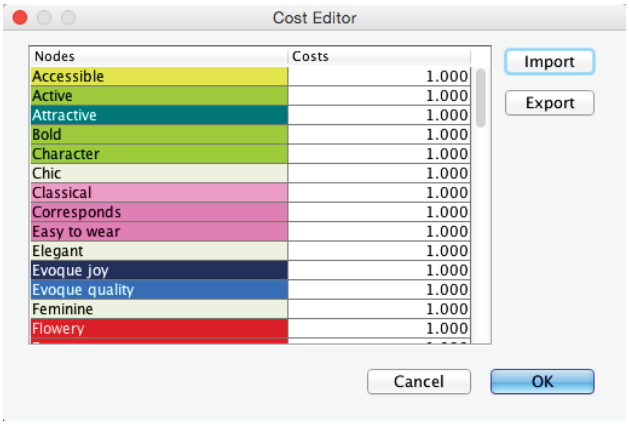
Unchecking the box `Cost` or setting a value ≤ 0 results in the node becoming Not Observable.



Alternatively, we can bring up the Cost Editor for all nodes by right-clicking on the Graph Panel and then selecting `Edit Costs` from the Context Menu.



The Cost Editor presents the default values for all nodes.

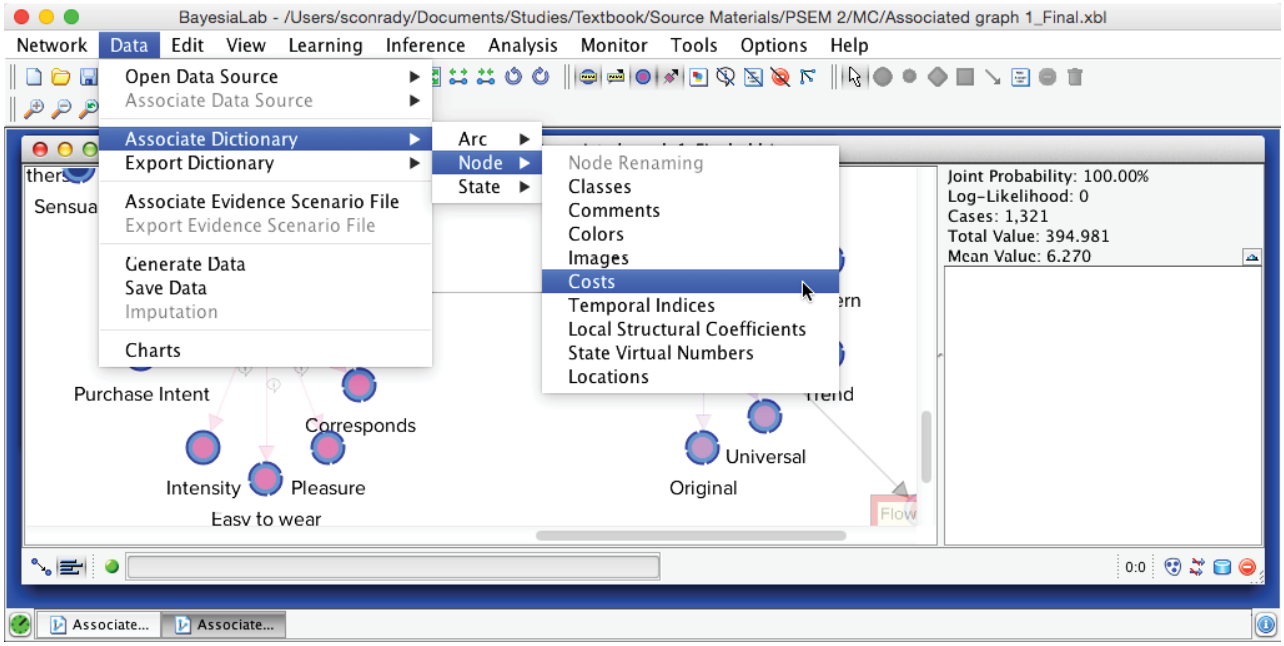


Again, setting values to zero will make nodes Not Observable. Instead of applying this setting node by node, we can import a Cost Dictionary that defines the values for each node. An excerpt from the text file is shown below. The syntax is straightforward: Not Observable is represented by 0.

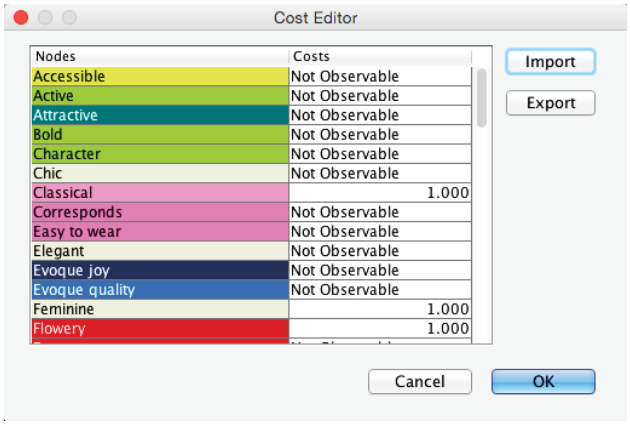
CostDrivers.txt

```
Product=0
Purchase\ Intent=1.0
Fresh=1.0
Flowery=1.0
Greedy=1.0
Spiced=1.0
Wooded=1.0
Sweet=1.0
Easy\ to\ wear=0
Rich=0
Fruity=1.0
Modern=0
Original=1.0
Feminine=1.0
Elegant=0
Tenacious=1.0
Evoque\ quality=0
Evoque\ joy=0
Make\ attractive=0
Universal=0
Trend=0
Personality=1.0
Pleasure=0
It\ will\ please\ others=0
Sensuality=0
Classical=1.0
High-End=0
Timeless=0
Corresponds=0
Attractive=0
```

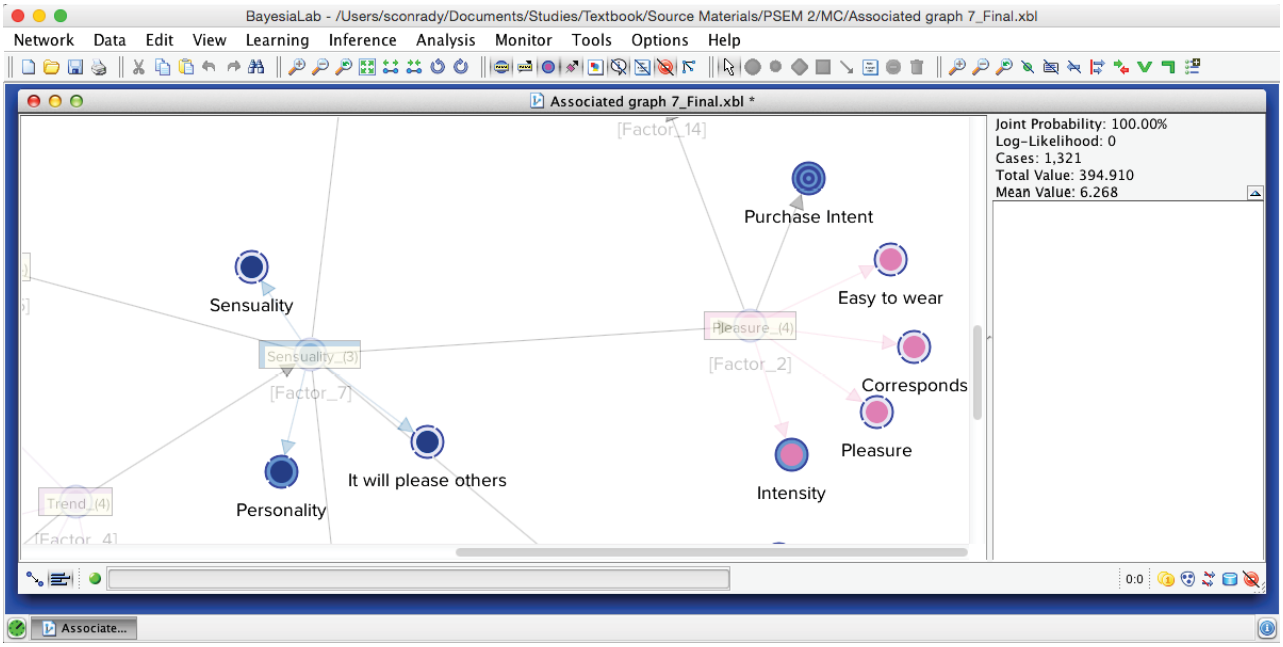
From within the Cost Editor, we can use the `Import` button to associate a Cost Dictionary. Alternatively, we can select `Menu > Data > Associate Dictionary > Node > Costs`.



Upon import, the Node Editor reflects the new values, and the presence of non-default values for costs is indicated by the Cost icon in the lower right-hand corner of the Graph Panel.



Furthermore, upon defining Costs, we can see that all Not Observable nodes are marked with a light purple background.



It is important to note that all Factors are also set to Not Observable in our example. In fact, we do have two options here:

1. The optimization can be done at the first level of the hierarchical model, i.e., using the Manifest variables;
2. The optimization can be performed at the second level of the model, i.e., using the Factors.

Most importantly, these two approaches cannot be combined as setting evidence on Factors will block information coming from Manifest variables. Formally declaring the Factors as Not Observable tells

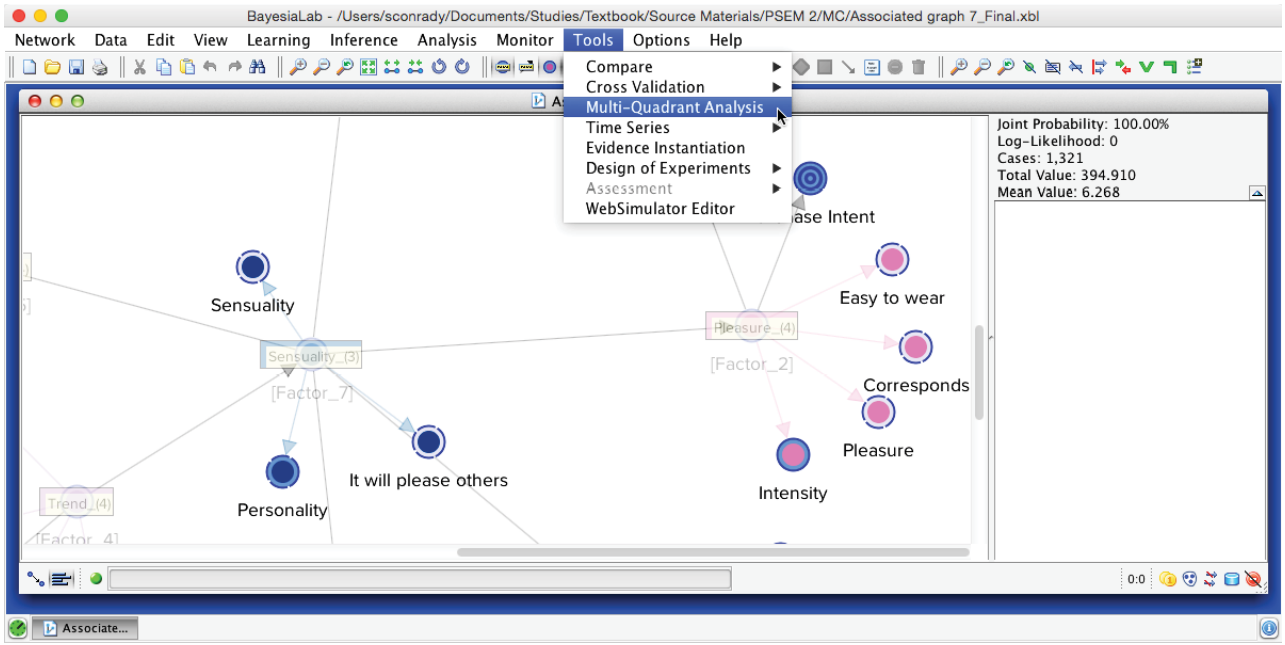
BayesiaLab to proceed with option #1. Indeed, we plan to perform optimization using the Manifest variables only.

Multi-Quadrant Analysis

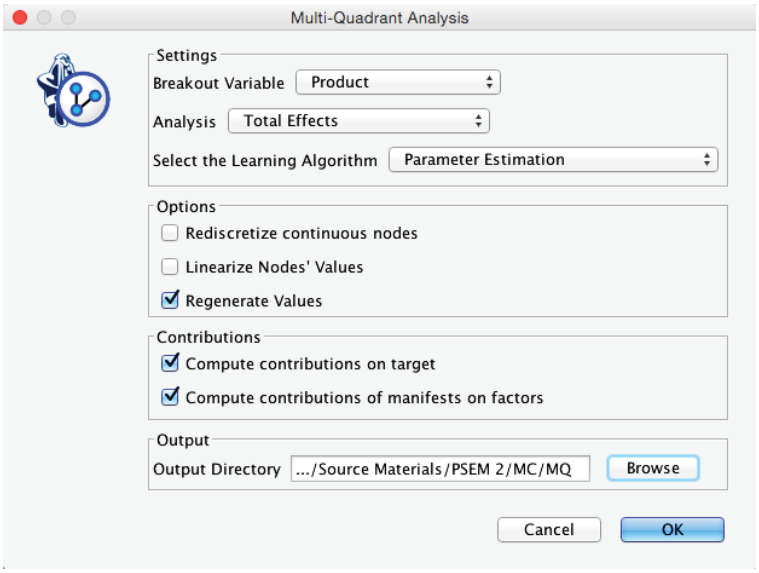
The network we have analyzed thus far modeled *Purchase Intent* as a function of perceived perfume characteristics. It is important to point out that this model represents the entire domain of all 11 tested perfumes. However, it is reasonable to speculate that different perfumes have different drivers of *Purchase Intent*. Furthermore, for purposes of product optimization, we certainly need to look at the dynamics of each product individually.

BayesiaLab assists us in this task by means of Multi-Quadrant Analysis. This function can generate new networks as a function of a Breakout Node in an existing network. This is where the node *Product* comes into play, which has been excluded all this time. Our objective is to generate a set of networks that model the drivers of *Purchase Intent* for each perfume individually, as identified by the *Product* breakout variable.

We start the Multi-Quadrant Analysis by selecting `Menu > Tools > Multi-Quadrant Analysis`.



This brings up the dialog box, in which we need to set a number of options:



Firstly, the `Breakout Variable` must be set to *Product* to indicate that we want to generate a network for each state of *Product*. For Analysis, we have several options: We choose `Total Effects` to be consistent with the earlier analysis. Regarding the `Learning Algorithm`, we select `Parameter Estimation`. This choice becomes evident once the dataset representing the "overall market" is split into 11 product-specific subsets. Now, the number of available observations per product drops to only 120. Given that most of our variables have 5 states, learning a structure with a dataset that small would be challenging.

This also explains why we used the entire dataset to learn the PSEM structure, which all the products will share. However, using Parameter Estimation will ensure that the parameters, i.e., the probability tables of each network, will be estimated based on the subsets of records associated with each state of *Product*.

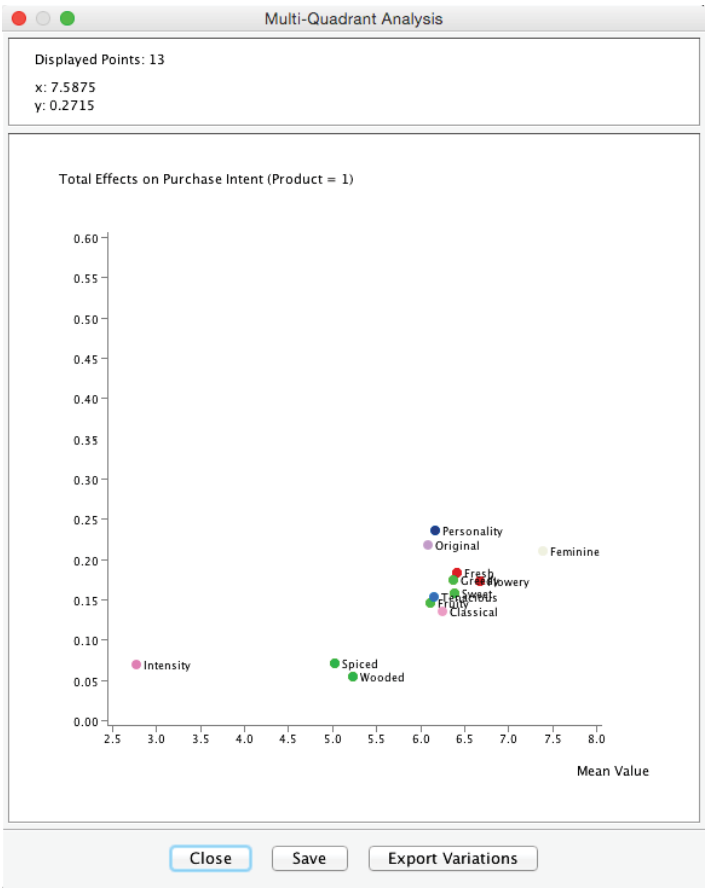
Among the Options, we check `Regenerate Values`. This recomputes, for each new network, the values associated with each state of the discretized nodes based on the respective subset of data.

There is no need to check `Rediscretize Continuous Nodes` because all discretized nodes share the same variation domain, and we required equal distance binning during the data import. However, we do recommend using this option if the variation domains are different between subsets in a study, e.g., sales volume in California versus Vermont. Without using the `Rediscretize Continuous Nodes` option, it could happen that all data points for sales in Vermont end up in the first bin, effectively transforming the variable into a constant.

Furthermore, we do not check the option for `Linearize Nodes' Values` either. This function reorders a node's states so that its states' values have a monotonically positive relationship with the values of the Target Node. Applying this transformation to the node *Intensity* would artificially increase its impact. It would incorrectly imply that it is possible to change a perfume in a way that simultaneously satisfies consumers who rated it too subtle and those who rated it too strong. Needless to say, this is impossible.

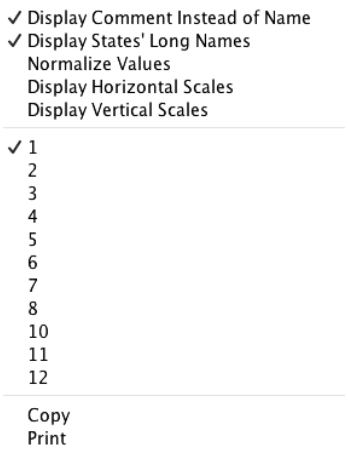
Finally, computing all `Contributions` will be helpful for interpreting each product-specific network.

Upon clicking `OK`, 11 networks are created and saved to the Output Directory defined in the dialog box. Each network is then analyzed with the specified Analysis method to produce the Multi-Quadrant Plot.

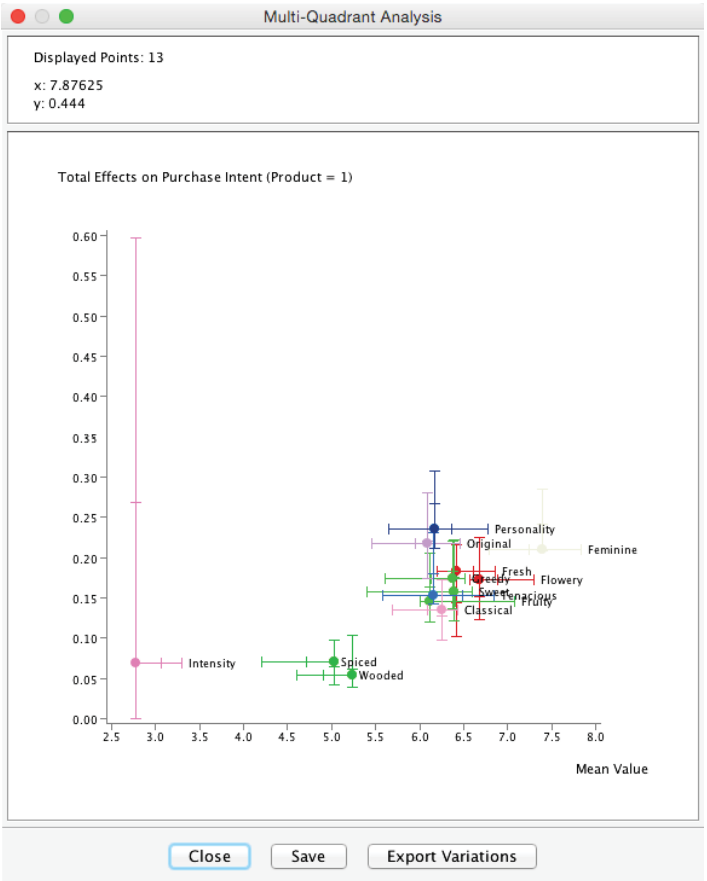


The x-value of each point indicates the mean value of the corresponding Manifest Node, as rated by those respondents who have evaluated Product 1; the position on the y-axis reflects the computed Total Effect.

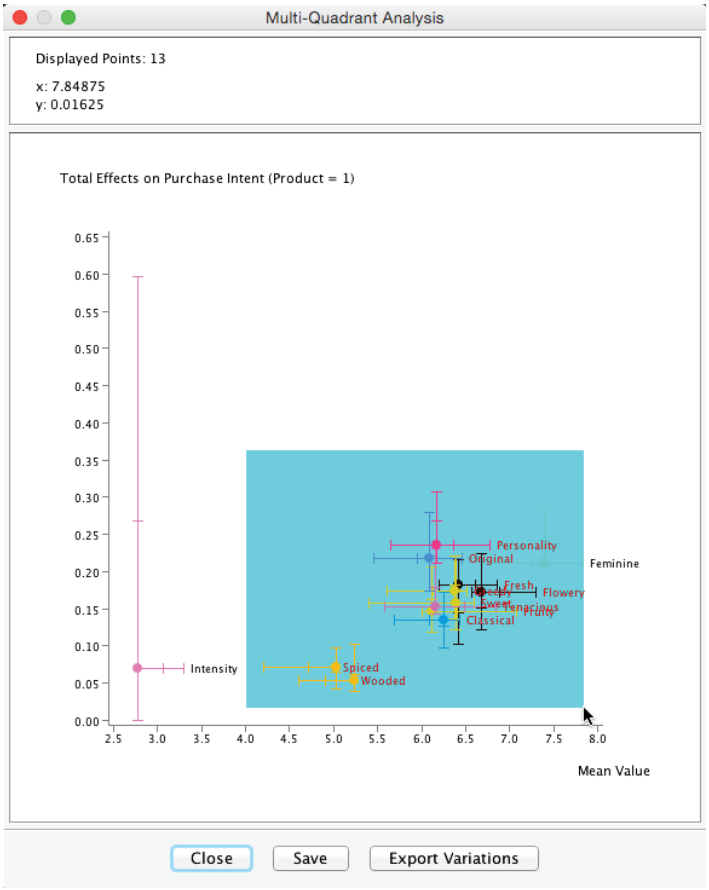
From the contextual menu, we can choose `Display Horizontal Scales` and `Display Vertical Scales`, which provide the range of positions of the other products.



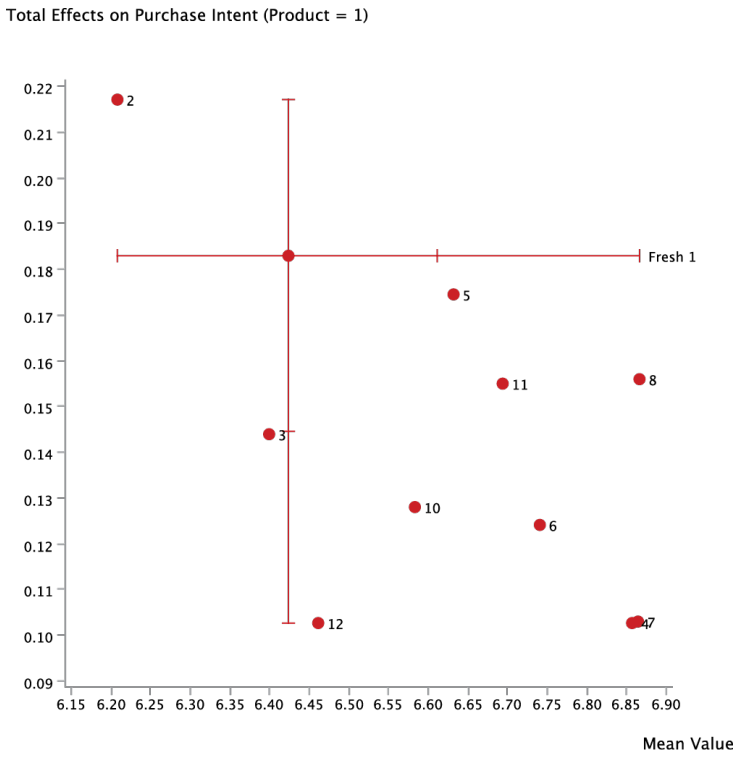
Using Horizontal Scales provides a quick overview of how the product under study is rated vis-à-vis other products. The Vertical Scales compare the importance of each dimension with respect to *Purchase Intent*. Finally, we can select the individual product to be displayed in the Multi-Quadrant Analysis window via the Contextual Menu.



Drawing a rectangle with the cursor zooms in on the specified area of the plot.



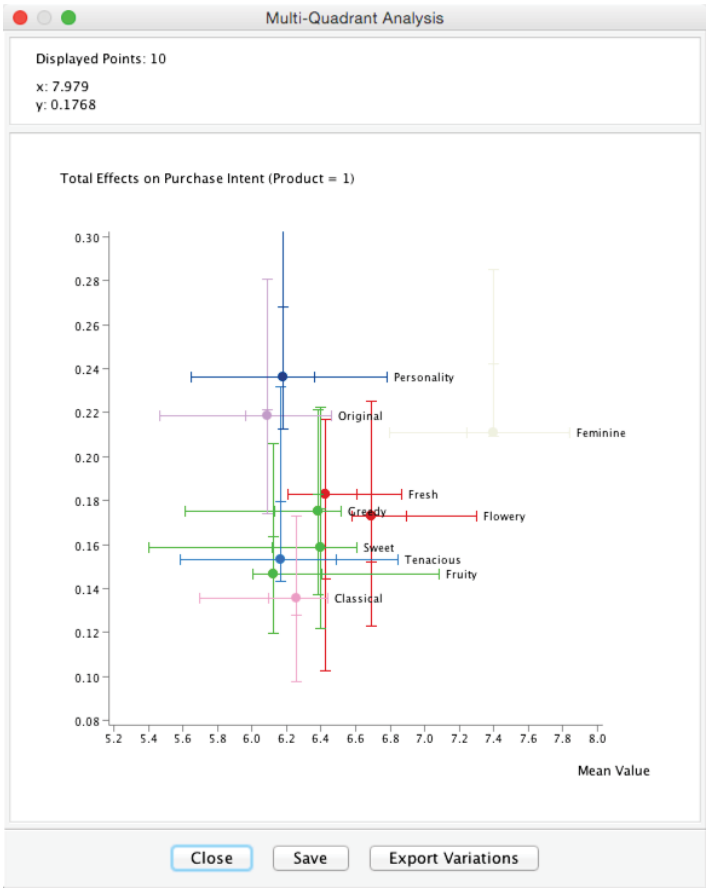
The meaning of the Horizontal Scales and Vertical Scales becomes apparent when hovering over any dot, as this brings up the position of the other (competitive) products with regard to the currently highlighted attribute.



This means, for instance, that Product 2 and Product 7 are rated lowest and highest, respectively, on the x-scale with regard to the variable *Fresh*. Regarding the Total Effect on *Purchase Intent*, Product 11 and Product 2 mark the bottom and top end, respectively (y-scale).

From a product management perspective, this suggests that for Product 1, with regard to the attribute *Fresh*, there is a large gap to the level of the best product, i.e., Product 7. So, one could interpret the variation from the status quo to the best level as "room for improvement" for Product 1.

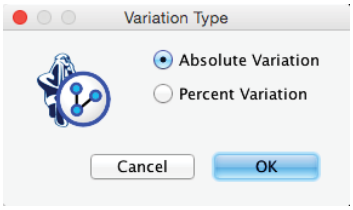
On the other hand, as we can see below, the variables *Personality*, *Original*, and *Feminine* and have a greater Total Effect on *Purchase Intent*. These relative positions will soon become relevant as we must simultaneously consider the potential for improvement and the importance of optimizing *Purchase Intent*.

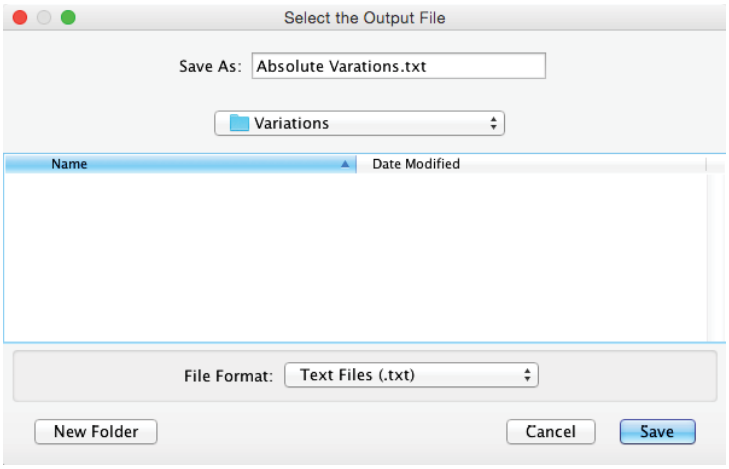


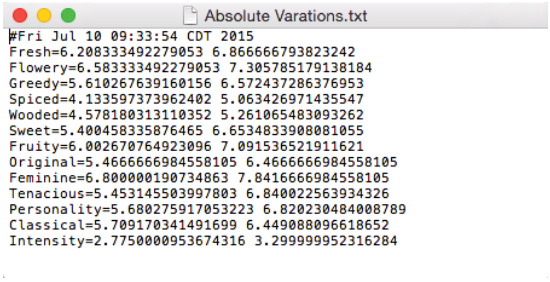
BayesiaLab's Export Variations function allows us to save the variation domain for each driver, i.e., the minimum and maximum mean values observed across all products in the study.

Knowing these variations will be useful for generating realistic scenarios for subsequent optimization. However, what do we mean by "realistic"? Ultimately, only a subject matter expert can judge how realistic a scenario is. However, a good heuristic is whether or not a certain level is achieved by any product in the market. One could argue that the existence of a certain satisfaction level for some product means that such a level is not impossible to achieve and is, therefore, "realistic."

Clicking the `Export Variations` button saves the Absolute Variations to a text file for subsequent use in optimization.

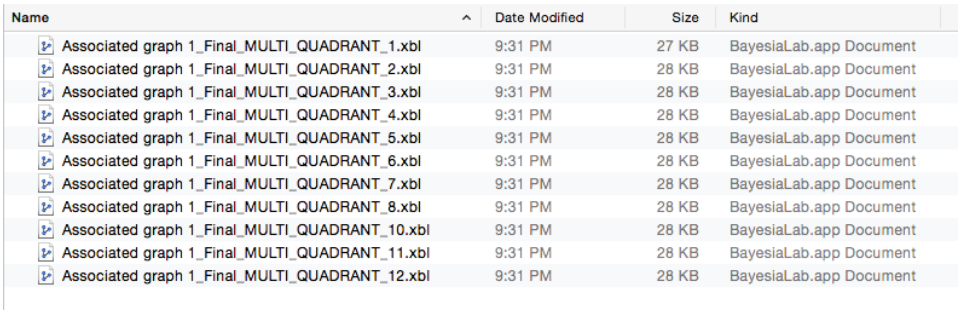




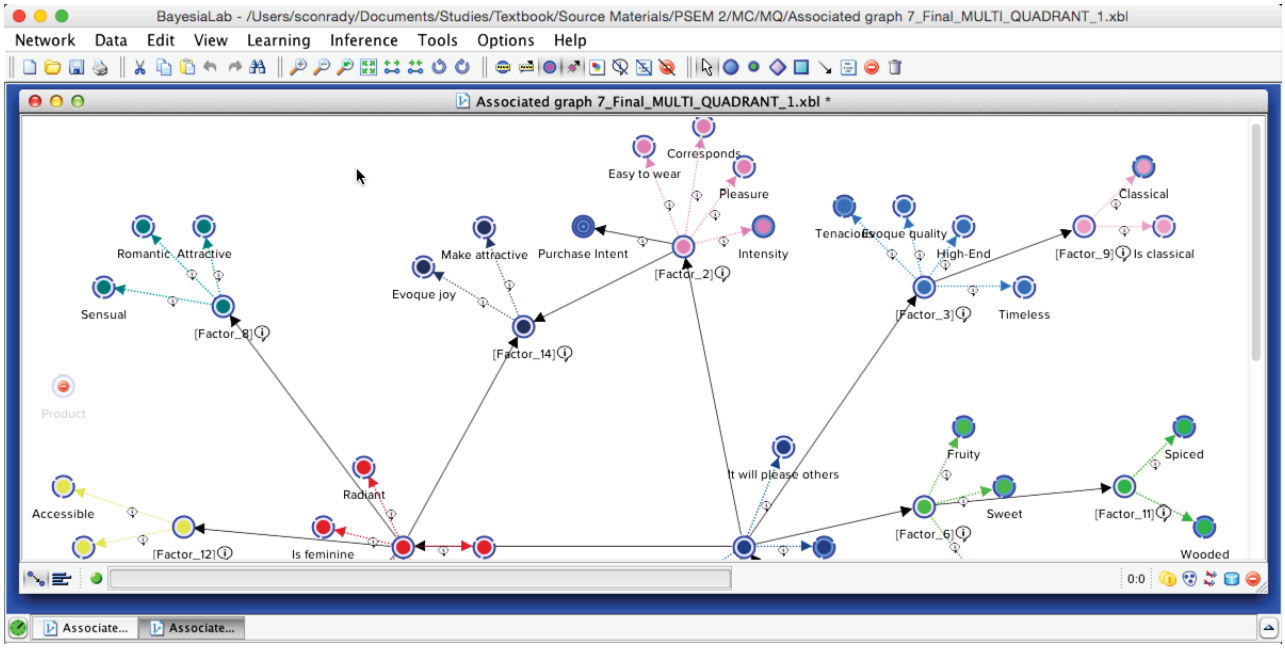


Product Optimization

In order to perform optimization for a particular product, we need to open the network for that specific product. Networks for all products were automatically generated and saved during the Multi-Quadrant Analysis, so we need to open the network for the product of interest. The suffix in the file name reflects the *Product*.



To demonstrate the optimization process, we open the file that corresponds to Product 1. Structurally, this network is identical to the network learned from the entire dataset. However, the parameters of this network were estimated only on the basis of the observations associated with Product 1.



Now we have all the elements that are necessary for optimizing the *Purchase Intent* of Product 1:

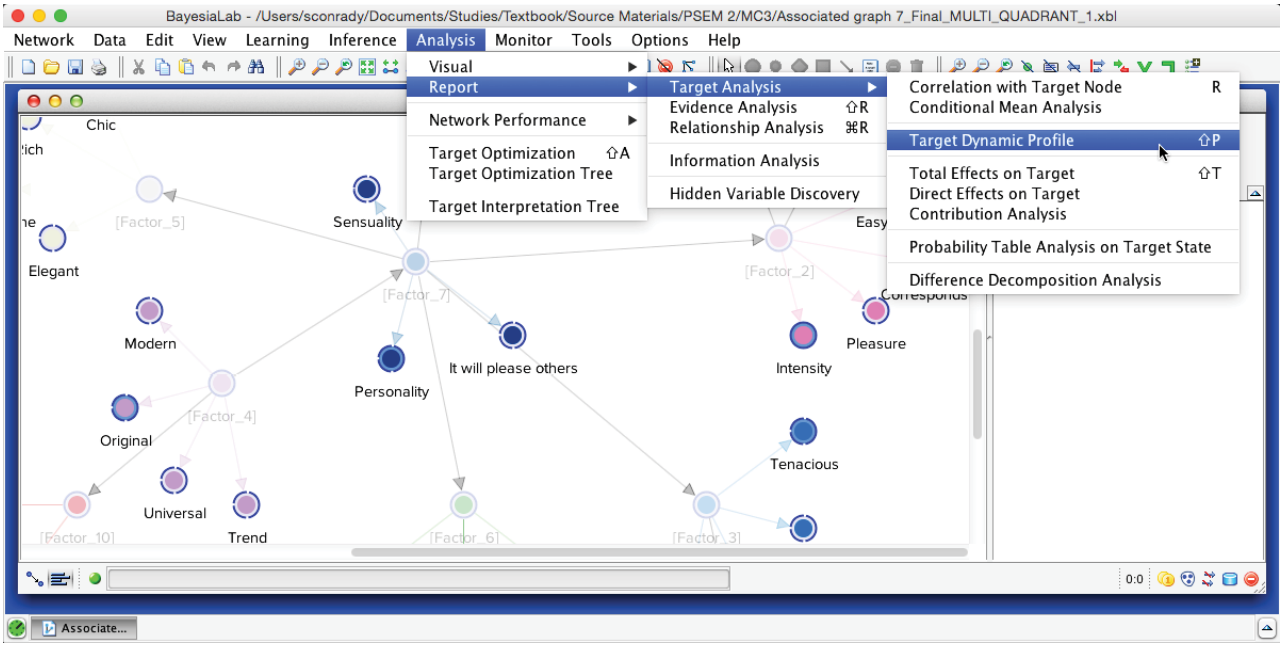
- A network that is specific to Product 1;
- A set of driver variables, selected by excluding the non-driver variables via Costs;
- Realistic scenarios, as determined by the Variation Domains of each driver variable.

With the above, we are now able to search for node values that optimize *Purchase Intent*.

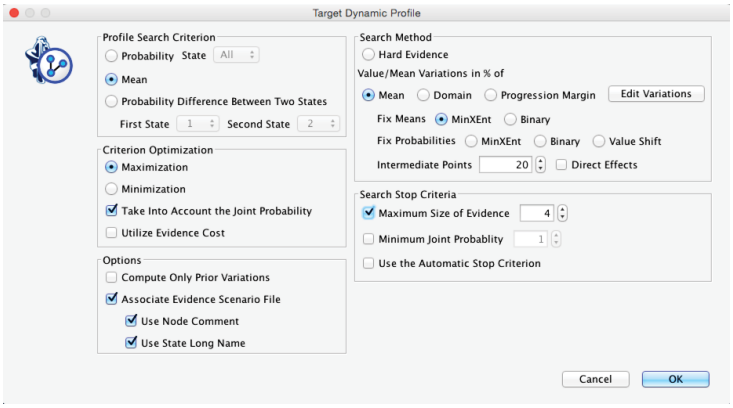
Target Dynamic Profile

Before we proceed, we need to explain what we mean by optimization. As all observations in this study are consumer perceptions, it is clear that we cannot directly manipulate them. Rather, this optimization aims to identify in which order the perfume maker should address these perceptions. Some consumer perceptions may relate to specific odoriferous compounds that a perfumer can modify; other perceptions may be influenced by marketing and branding initiatives. However, the precise mechanism of influencing consumer perceptions is not the subject of our discussion. From our perspective, the causes that could influence the perception are hidden. Thus, we have here a prototypical application of Soft Evidence, i.e., we assume that the simulated changes in the distribution of consumer perceptions originate in hidden causes (see Numerical Evidence in Chapter 7).

While BayesiaLab offers a number of optimization methods, Target Dynamic Profile is appropriate here. We start it from within Validation Mode F5 by selecting `Menu > Analysis > Report > Target Analysis > Target Dynamic Profile`.



We need to explain the many options that must be set for Target Dynamic Profile. These options will reflect our objective of pursuing realistic sets of evidence:



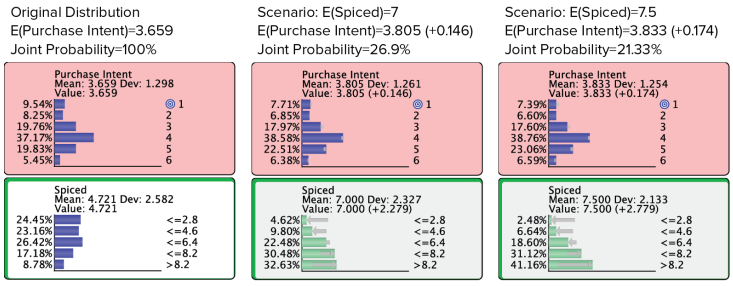
In Profile Search Criterion, we specify that we want to optimize the mean value of the Target Node as opposed to any particular state or the difference between states.

Joint Probability

Next, we specify under Criterion Optimization that the mean value of the Target Node is to be maximized. Furthermore, we check `Take Into Account` the Joint Probability. This weights any potential improvement in the mean value of the Target Node by the joint probability corresponding to the set of simulated evidence that generated this improvement. The joint probability of a simulated Evidence

Scenario will be high if its probability distribution is close to the original probability distribution observed in the consumer population: the higher the joint probability, the closer the simulated scenario to the status quo of customer perception.

In practice, checking this option means that we prefer smaller improvements with a high joint probability over larger ones with a low joint probability: $0.146 \times 26.9\% = 0.0393 > 0.174 \times 21.33\% = 0.0371$.



If all simulated values were within the constraints set in the Variation Editor, it would be better to increase the driver variable *Spiced* to a simulated value of 7 rather than 7.5, even though *Purchase Intent* would be higher for the latter value of *Spiced*. In other words, the "support" for $E(Spiced) = 7$ is greater than for $E(Spiced) = 7.5$, as more respondents are already in agreement with such a scenario. Once again, this is about pursuing achievable improvements rather than proposing pie-in-the-sky scenarios.

Costs

In this example, so far, we have only used Costs for selecting the subset of driver variables. Additionally, we can utilize Costs in the original sense of the word in the optimization process. For instance, if we had information on the typical cost of improving a specific rating by one unit, we could enter such a value as cost. This could be a dollar value, or we could set the costs to reflect the relative effort required for the same amount of change, e.g., one unit, in each driver variable. For example, a marketing manager may know that it requires twice as much effort to change the perception of *Feminine* compared to changing the perception of Sweet. If we want to quantify such efforts by using Costs, we must ensure that all variables' costs share the same scale. For instance, if some drivers are measured in dollars and others are measured in terms of time spent in hours, we will need to convert hours to dollars.

In our study, we leave all the included driver variables at a Cost of 1, i.e., we assume that it requires the same effort for the same amount of change in any driver variable. Hence, we can leave the Utilize Evidence Cost unchecked (Not Observable nodes still remain excluded as driver variables).

Compute Only Prior Variations needs to remain unchecked as well. This option would be useful if we were interested in only computing the marginal effect of drivers. We would not want any cumulative effects or conditional variations given other drivers for that purpose.

Associate Evidence Scenario will save the identified sets of evidence for subsequent evaluation.

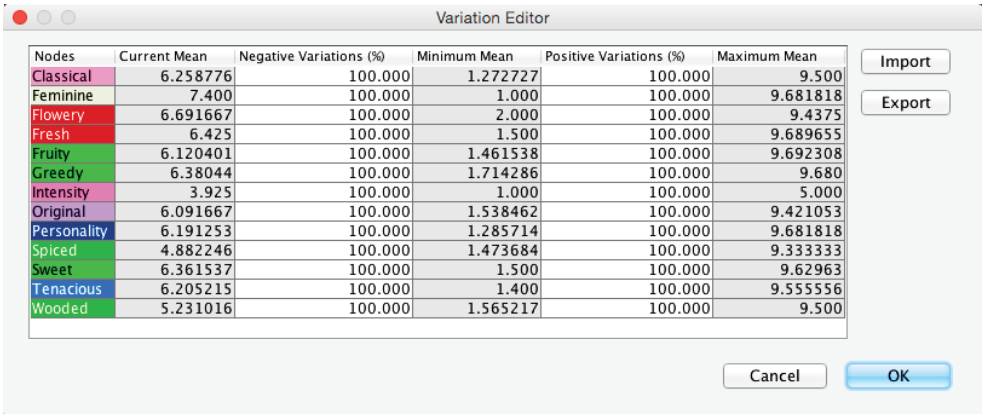
The setting of Search Methods is critically important for the optimization task. We need to define how to search for sets of evidence. Using Hard Evidence means that we would exclusively try out

sets of evidence consisting of nodes with one state set to 100%. This would imply that we simulate a condition in which all consumers perfectly agree with regard to some ratings. Needless to say, this would be utterly unrealistic. Instead, we will explore sets of evidence consisting of distributions for each node by modifying their mean values as Soft Evidence. More precisely, we use the MinXEnt method to generate such evidence (see Minimum Cross-Entropy in Chapter 7).

In this context, we reintroduce the Variations we saved earlier. We reason that the best-rated product with regard to a particular attribute represents a plausible upper limit for what any product could strive for in terms of improvement. This also means that a driver variable that has already achieved the best level will not be optimized further in this framework.

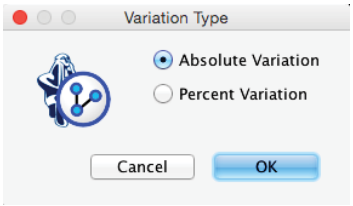
Variation Editor

Clicking on Variations brings up the Variation Editor. By default, it shows variations in the amount of ±100% of the current mean.

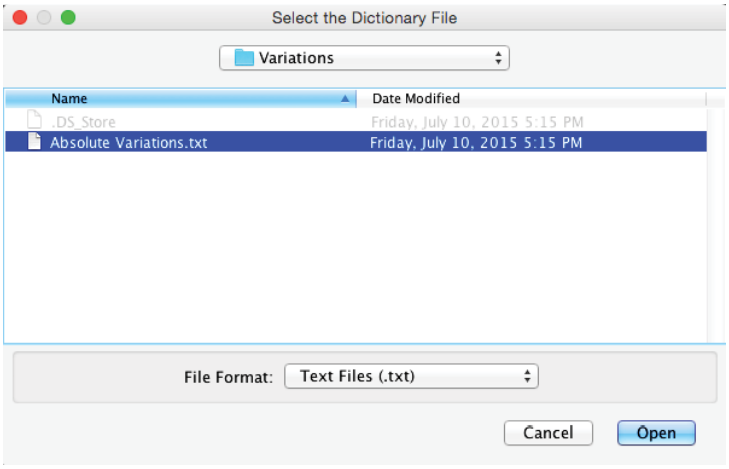


Nodes	Current Mean	Negative Variations (%)	Minimum Mean	Positive Variations (%)	Maximum Mean
Classical	6.258776	100.000	1.272727	100.000	9.500
Feminine	7.400	100.000	1.000	100.000	9.681818
Flowery	6.691667	100.000	2.000	100.000	9.4375
Fresh	6.425	100.000	1.500	100.000	9.689655
Fruity	6.120401	100.000	1.461538	100.000	9.692308
Greedy	6.38044	100.000	1.714286	100.000	9.680
Intensity	3.925	100.000	1.000	100.000	5.000
Original	6.091667	100.000	1.538462	100.000	9.421053
Personality	6.191253	100.000	1.285714	100.000	9.681818
Spiced	4.882246	100.000	1.473684	100.000	9.333333
Sweet	6.361537	100.000	1.500	100.000	9.62963
Tenacious	6.205215	100.000	1.400	100.000	9.555556
Wooded	5.231016	100.000	1.565217	100.000	9.500

To load the Variations we generated earlier through Multi-Quadrant Analysis, we click `Import` and select `Absolute Variations` from the pop-up window.

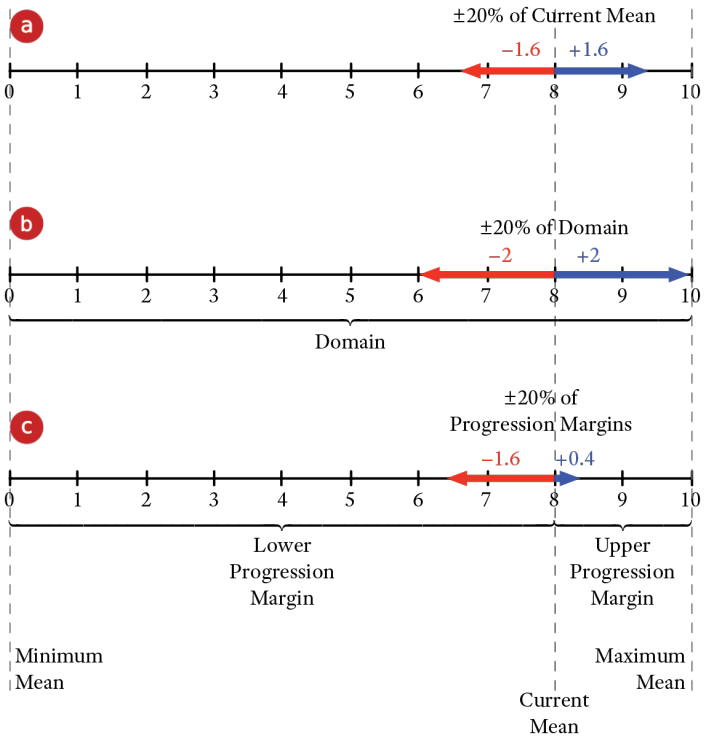


Now, we can open the previously saved file.



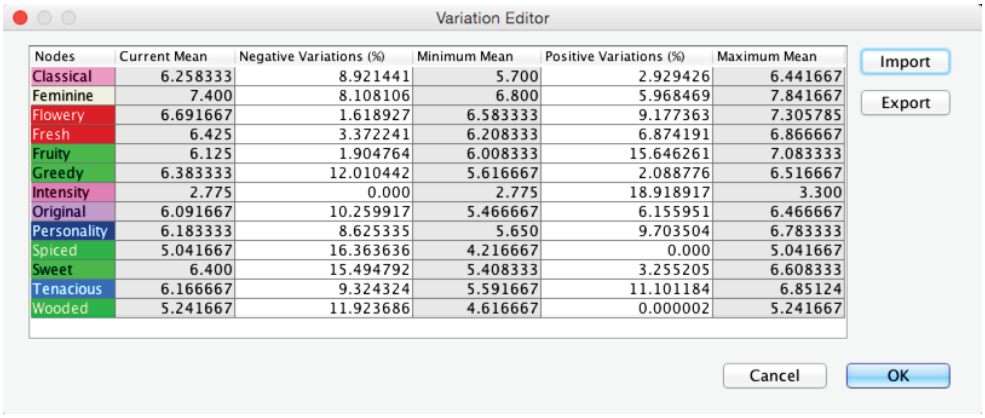
The Variation Editor now reflects the constraints. Any available expert knowledge can be applied here by entering new values for the Minimum Mean or Maximum Mean or by entering percent values for Positive Variations and Negative Variations.

Depending on the setting, the percentages are relative to (a) the Mean, (b) the Domain, or (c) the Progression Margin.



Selecting the `Progression Margin` is particularly useful as it automatically constrains the positive and negative variations in proportion to the gap from the Current Mean to the Maximum Mean and Minimum Mean values, respectively. In other words, it limits the improvement potential of a driver

variable as its value approaches the maximum. It is a practical–albeit arbitrary–approach to prevent overly optimistic optimizations.



Nodes	Current Mean	Negative Variations (%)	Minimum Mean	Positive Variations (%)	Maximum Mean
Classical	6.258333	8.921441	5.700	2.929426	6.441667
Feminine	7.400	8.108106	6.800	5.968469	7.841667
Flowery	6.691667	1.618927	6.583333	9.177363	7.305785
Fresh	6.425	3.372241	6.208333	6.874191	6.866667
Fruity	6.125	1.904764	6.008333	15.646261	7.083333
Greedy	6.383333	12.010442	5.616667	2.088776	6.516667
Intensity	2.775	0.000	2.775	18.918917	3.300
Original	6.091667	10.259917	5.466667	6.155951	6.466667
Personality	6.183333	8.625335	5.650	9.703504	6.783333
Spiced	5.041667	16.363636	4.216667	0.000	5.041667
Sweet	6.400	15.494792	5.408333	3.255205	6.608333
Tenacious	6.166667	9.324324	5.591667	11.101184	6.85124
Wooded	5.241667	11.923686	4.616667	0.000002	5.241667

Next, we select MinXEnt in the `Search Method` panel as the method for generating Soft Evidence. In terms of Intermediate Points, we set a value of 20. This means that BayesiaLab will simulate 22 values for each node, i.e., the minimum and maximum plus 20 intermediate values, all within the constraints set by the variations. This is particularly useful in the presence of non-linear effects.

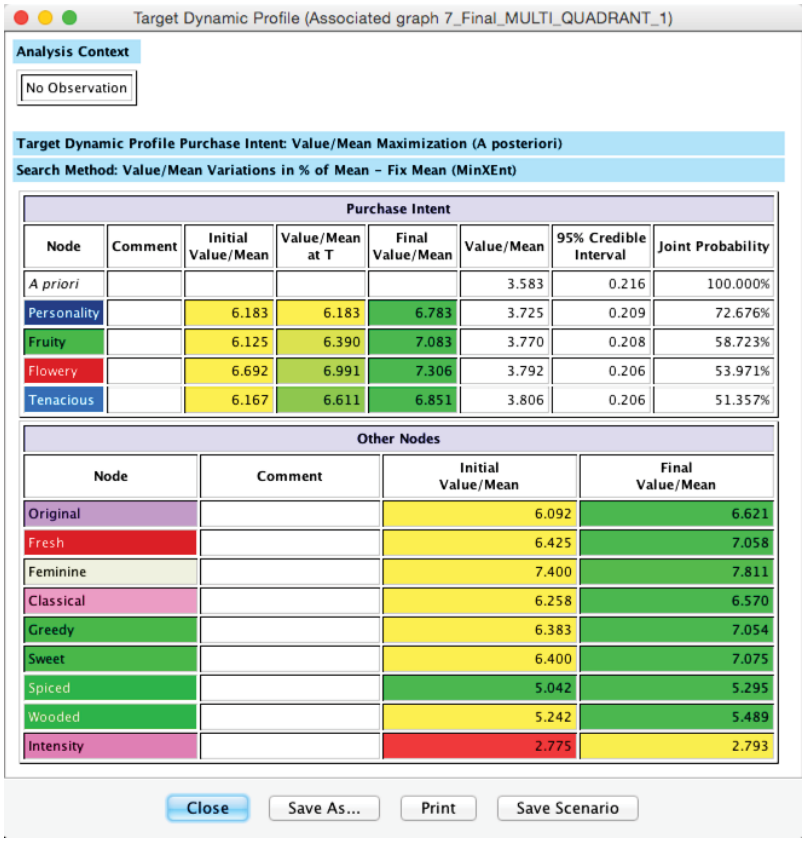
Within the `Search Stop Criteria` panel, the Maximum Size of Evidence specifies the maximum number of driver variables recommended as part of the optimization policy. Real-world considerations once again drive this setting. Although one could wish to bring all variables to their ideal level, a decision-maker may recognize that it is not plausible to pursue anything beyond the top 4 variables.

Alternatively, we can choose to terminate the optimization process once the joint probability of the simulated evidence drops below the specified Minimum Joint Probability.

The final option, Use the Automatic Stop Criterion, leaves it up BayesiaLab to determine whether adding further evidence provides a worthwhile improvement for the Target Node.

Optimization Results

Once the optimization process concludes, we obtain a report window that contains a list of priorities: *Personality*, *Fruity*, *Flowery*, and *Tenacious*.



Target Dynamic Profile (Associated graph 7_Final_MULTI_QUADRANT_1)

Analysis Context

No Observation

Target Dynamic Profile Purchase Intent: Value/Mean Maximization (A posteriori)

Search Method: Value/Mean Variations in % of Mean - Fix Mean (MinXEnt)

Purchase Intent

Node	Comment	Initial Value/Mean	Value/Mean at T	Final Value/Mean	Value/Mean	95% Credible Interval	Joint Probability
A priori					3.583	0.216	100.000%
Personality		6.183	6.183	6.783	3.725	0.209	72.676%
Fruity		6.125	6.390	7.083	3.770	0.208	58.723%
Flowery		6.692	6.991	7.306	3.792	0.206	53.971%
Tenacious		6.167	6.611	6.851	3.806	0.206	51.357%

Other Nodes

Node	Comment	Initial Value/Mean	Final Value/Mean
Original		6.092	6.621
Fresh		6.425	7.058
Feminine		7.400	7.811
Classical		6.258	6.570
Greedy		6.383	7.054
Sweet		6.400	7.075
Spiced		5.042	5.295
Wooded		5.242	5.489
Intensity		2.775	2.793

Close | Save As... | Print | Save Scenario

To explain the items in the report, we present a simplified and annotated version of the report below. Note that this report can be saved in HTML format for subsequent editing as a spreadsheet.

Analysis Context	
No Observation	No other evidence is set

Search Method: Value/Mean Variations in % of Mean -- Fix Mean (MinXEnt)					
Target Dynamic Profile Purchase Intent: Value/Mean Maximization (A posteriori)					
Purchase Intent					
Node	**Initial Value/Mean**	**Value/Mean at T**	**Final Value/Mean**	**Value/Mean**	**Joint Probability**
				Initial value of Purchase Intent, prior to optimization. ▼	Initial joint probability is 100%. ▼
			A priori ▶	3.583	100.00%
Personality	6.183	6.183	6.783	3.725	72.68%
▲ Most important node	▲ Initial value of ***Intensity,*** prior to optimization		▲ Optimal value of ***Intensity,*** within given constraints.	▲ Value of ***Purchase Intent*** after ***Personality*** is set to optimal value. This means that after setting ***Personality*** to the optimal value, ***Purchase Intent*** increases from 3.583 to 3.725.	▲ New joint probability after setting ***Personality.*** This means that 72.68% of the observations already meet this condition.
Fruity	6.125	6.39	7.083	3.77	58.72%
▲ 2nd most important node	▲ Initial value of ***Fruity,*** prior to optimization.	▲ Value of ***Fruity,*** after ***Personality*** is set to optimal value.	▲ Optimal value of ***Fruity,*** within given constraints.	▲ Value of ***Purchase Intent*** , after ***Personality*** and ***Fruity*** are set to optimal values.	▲ New joint probability after setting ***Personality.***
Flowery	6.692	6.991	7.306	3.792	53.97%
▲ 3rd most important node	▲ Initial value of ***Flowery,*** prior to optimization.	▲ Value of ***Flowery,*** after ***Personality*** and ***Fruity*** are set to optimal value.	▲ Optimal value of ***Flowery,*** within given constraints.	▲ Value of ***Purchase Intent,*** after ***Personality, Fruity,*** and ***Flowery*** are set to optimal values.	▲ New joint probability after setting ***Flowery.***
Tenacious	6.167	6.611	6.851	3.806	51.36%
▲ 4th most important node	▲ Initial value of ***Tenacious*** , prior to optimization.	▲ Value of ***Tenacious*** , after ***Personality, Fruity, and Flowery*** are set to optimal values.	▲ Optimal value of ***Tenacious*** , within given constraints.	▲ Value of ***Purchase Intent*** , after ***Personality, Fruity, Flowery,*** and ***Tenacious*** are set to optimal values.	▲ New joint probability after setting ***Tenacious.***
				▲ This means that after applying all four listed measures, an increase of 0.223 would be observed for ***Purchase Intent.***	

Other Nodes		
Node	**Initial Value/Mean**	**Final Value/Mean**
Original	6.092	6.621
	▲ Initial value of ***Original*** , prior to optimization.	▲ Final value of ***Original*** , after setting the top---4 nodes to the optimal levels.
Fresh	6.425	7.058
	▲ Initial value of ***Fresh,*** prior to optimization.	▲ Final value of ***Fresh,*** after setting the top---4 nodes to the optimal levels.
Feminine	7.4	7.811
	▲ Initial value of ***Feminine*** prior to optimization.	▲ Final value of ***Feminine,*** after setting the top---4 nodes to the optimal levels.

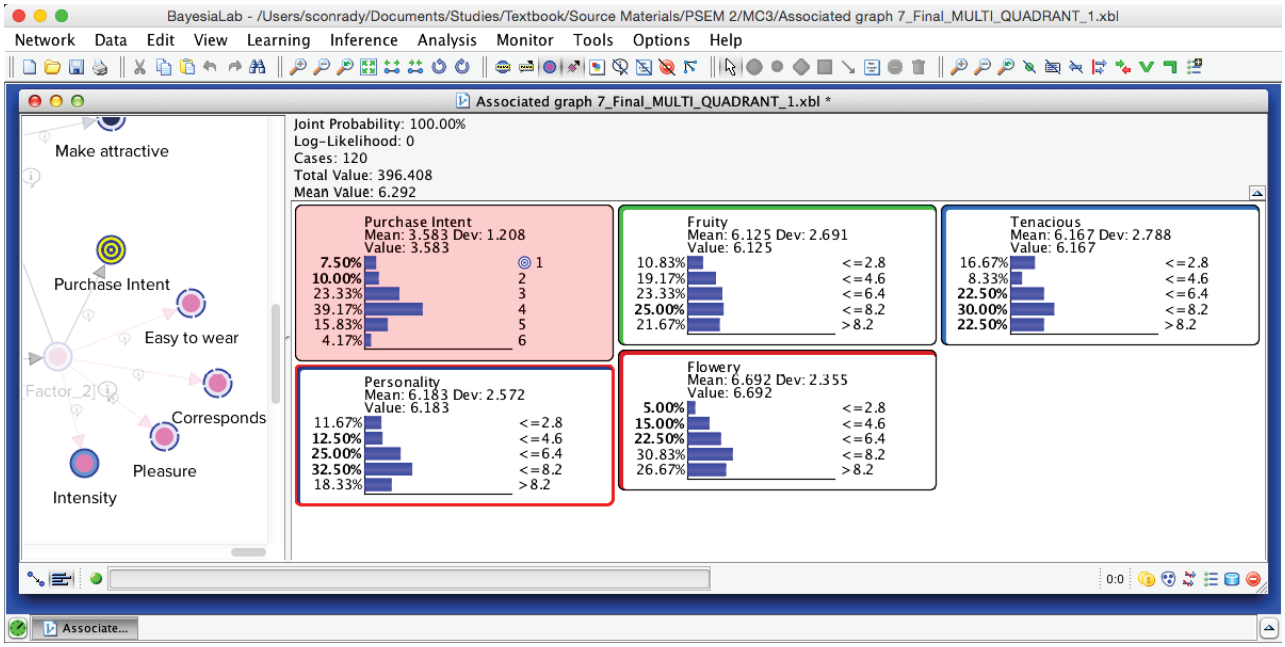
Collateral Effects

Even though ***Other Nodes*** do not directly belong to the recommended group of top drivers, the predicted level of Purchase Intent can only be reached with certainty if the ***Other Nodes*** can be set to these levels.

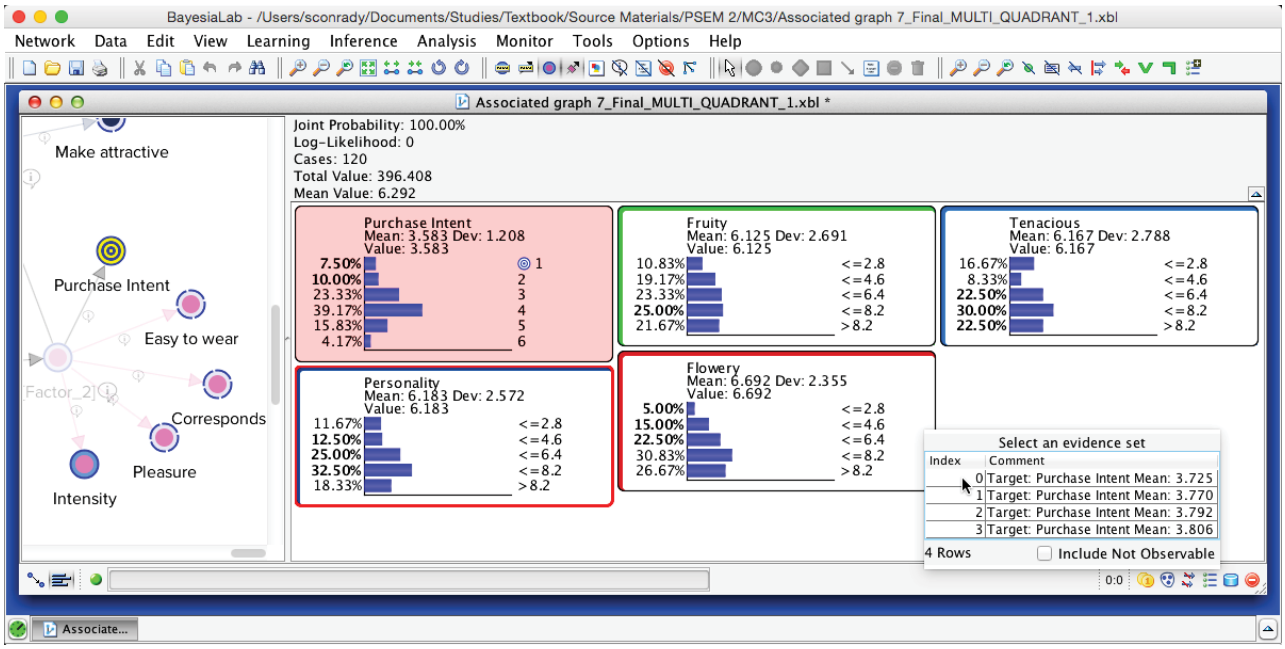
Most importantly, the Value/Mean column shows the successive improvement upon implementing each policy. From initially 3.58, the *Purchase Intent* improves to 3.86, which may seem like a fairly small step. However, the importance lies in the fact that this improvement is not based on Utopian thinking but rather on modest changes in consumer perception, well within the range of competitive performance.

Evidence Scenarios

As an alternative to interpreting the static report, we can examine each element in the list of priorities. To do so, we bring up all the Monitors of the nodes identified for optimization.

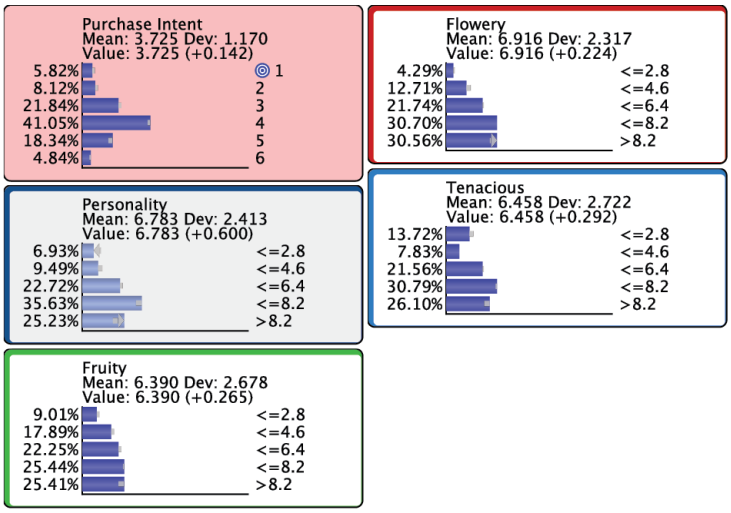


Then, we retrieve the individual steps by right-clicking on the Evidence Scenario icon in the lower right-hand corner of the main window.

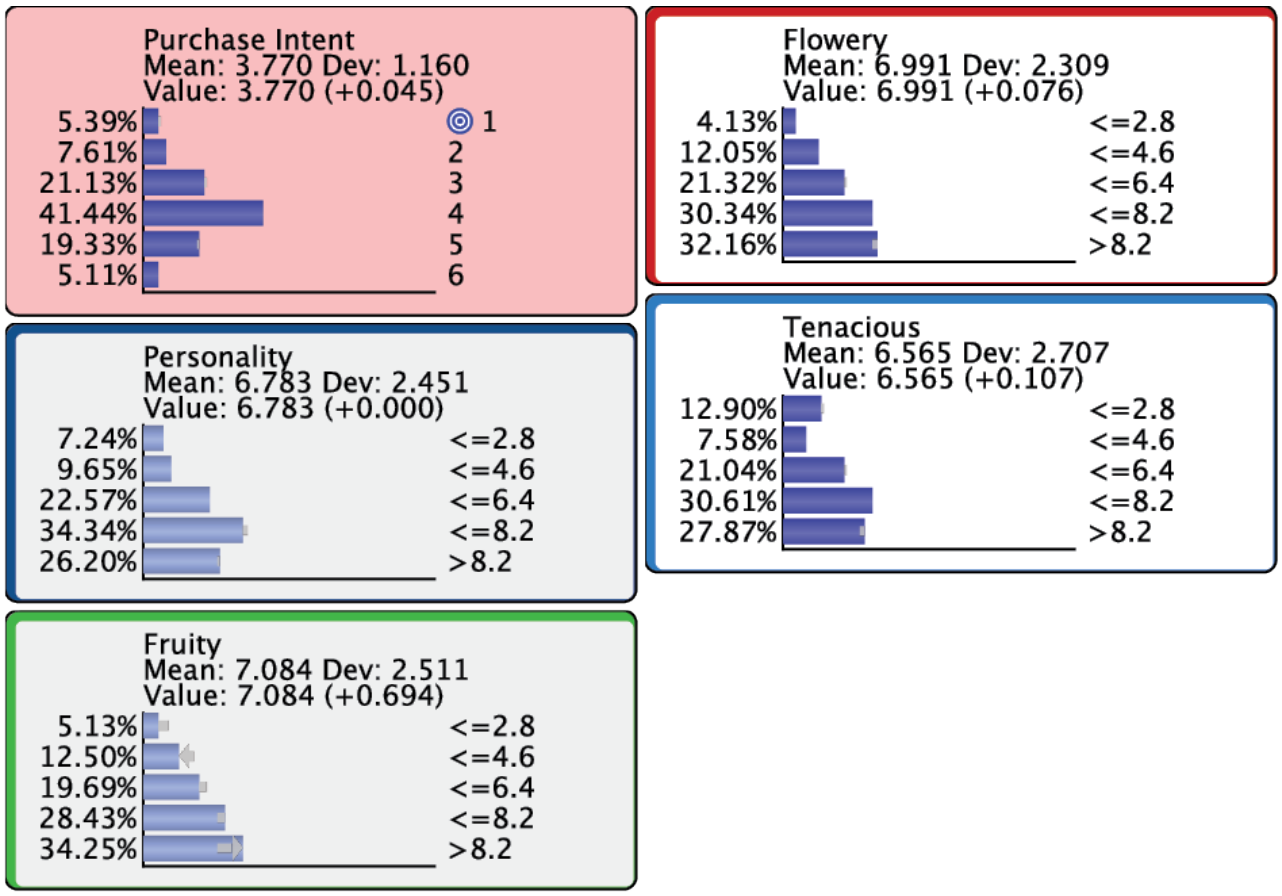


Selecting the first row in the table (Index=0) sets the evidence corresponding to the first priority, i.e., *Personality*. We can now see that the evidence we have set is a distribution rather than a single value. The small gray arrows indicate how the distribution of the evidence and the distributions of *Purchase*

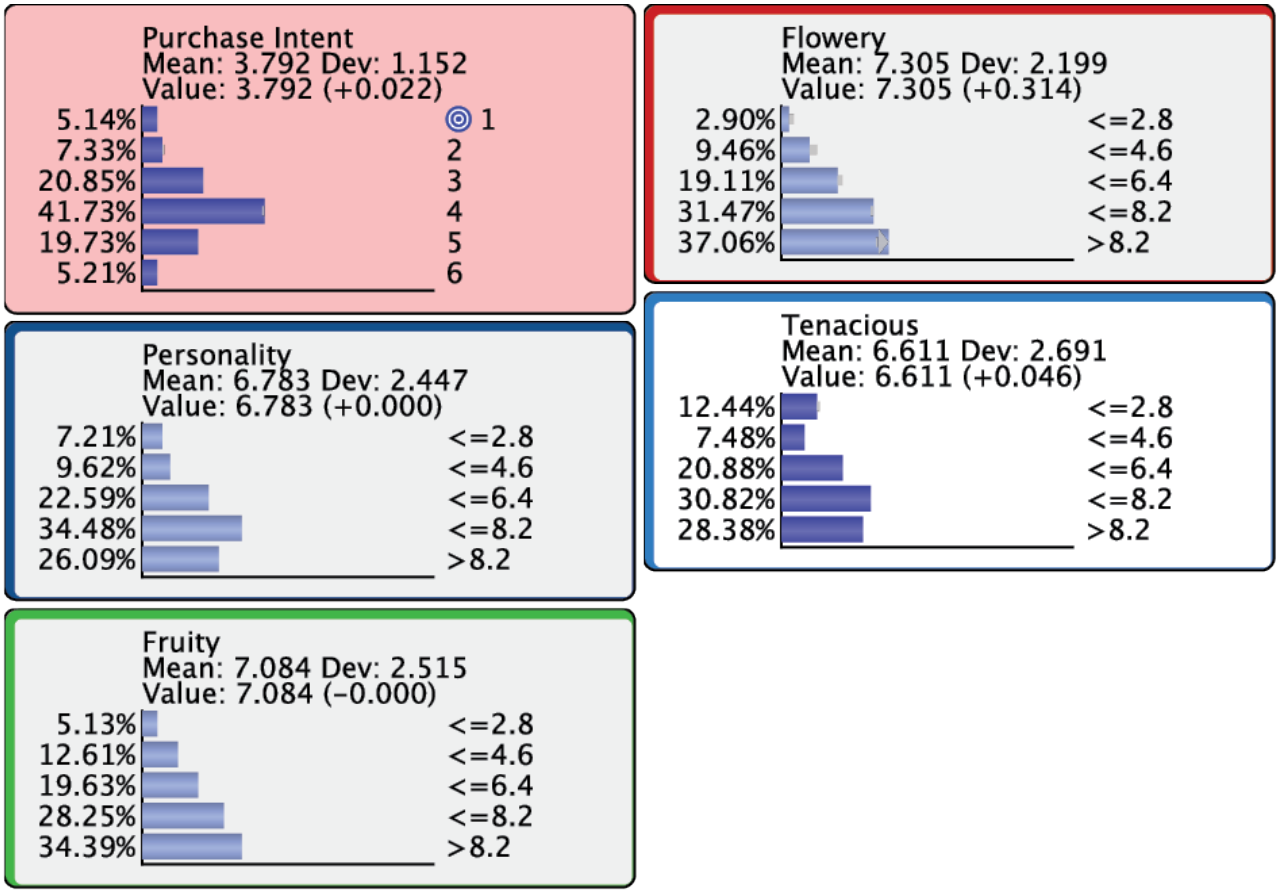
Intent, *Fruity*, *Flowery*, and *Tenacious*, are all different from their prior marginal distributions. The changes to the *Fruity*, *Flowery*, and *Tenacious* correspond to what is shown in the report in the column Value/Mean at T.

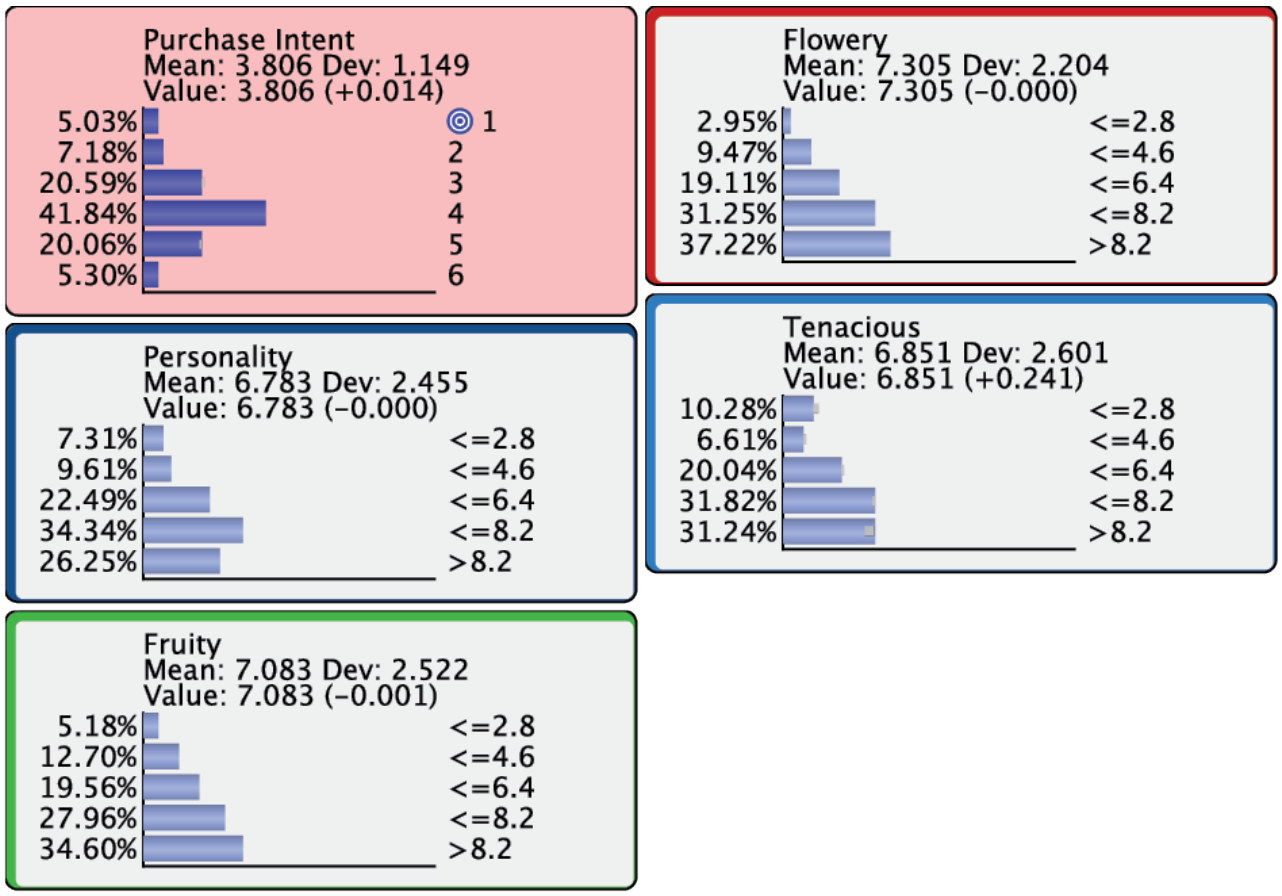


By selecting Index=1 we introduce a second set of evidence, i.e., the optimized distribution for *Personality*.



Continuing with Index 2 and 3, we see that the improvements to *Purchase Intent* become smaller.

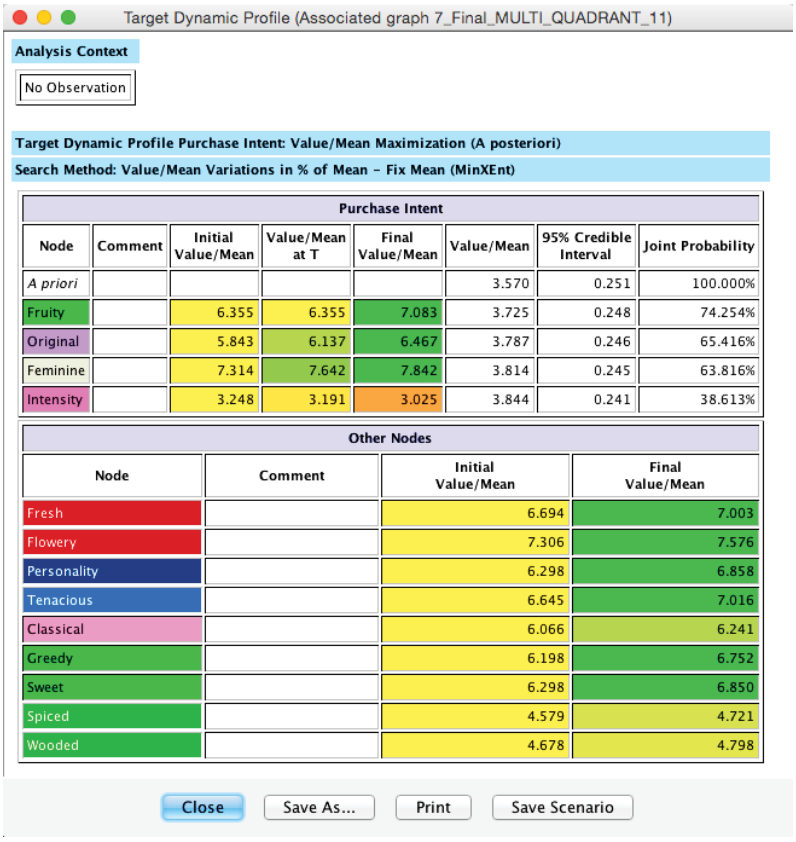

Purchase Intent
Mean: 3.792 Dev: 1.152
Value: 3.792 (+0.022)
5.14% 1
7.33% 2
20.85% 3
41.73% 4
19.73% 5
5.21% 6
Flowery
Mean: 7.305 Dev: 2.199
Value: 7.305 (+0.314)
2.90% <=2.8
9.46% <=4.6
19.11% <=6.4
31.47% <=8.2
37.06% >8.2
Personality
Mean: 6.783 Dev: 2.447
Value: 6.783 (+0.000)
7.21% <=2.8
9.62% <=4.6
22.59% <=6.4
34.48% <=8.2
26.09% >8.2
Tenacious
Mean: 6.611 Dev: 2.691
Value: 6.611 (+0.046)
12.44% <=2.8
7.48% <=4.6
20.88% <=6.4
30.82% <=8.2
28.38% >8.2
Fruity
Mean: 7.084 Dev: 2.515
Value: 7.084 (-0.000)
5.13% <=2.8
12.61% <=4.6
19.63% <=6.4
28.25% <=8.2
34.39% >8.2



Bringing up all the remaining nodes would bring up any "collateral" changes due to setting multiple pieces of evidence.

The results tell us that for Product 1, a higher consumer rating of *Personality* would be associated with a higher *Purchase Intent*. Improving the perception of *Personality* might be a task for the marketing and advertising team. Similarly, a better consumer rating of *Fruity* would also be associated with greater *Purchase Intent*. A product manager could then interpret this and request a change to some ingredients. Our model tells us that if such changes in consumer ratings were brought about in the proposed order, a higher *Purchase Intent* would potentially be observed.

While we have only presented the results for Product 1, we want to highlight that the priorities are different for each product, even though they all share the same underlying PSEM structure. The recommendations from the Target Dynamic Profile of Product 11 are shown below.



Target Dynamic Profile (Associated graph 7_Final_MULTI_QUADRANT_11)

Analysis Context

No Observation

Target Dynamic Profile Purchase Intent: Value/Mean Maximization (A posteriori)

Search Method: Value/Mean Variations in % of Mean – Fix Mean (MinXEnt)

Purchase Intent

Node	Comment	Initial Value/Mean	Value/Mean at T	Final Value/Mean	Value/Mean	95% Credible Interval	Joint Probability
A priori					3.570	0.251	100.000%
Fruity		6.355	6.355	7.083	3.725	0.248	74.254%
Original		5.843	6.137	6.467	3.787	0.246	65.416%
Feminine		7.314	7.642	7.842	3.814	0.245	63.816%
Intensity		3.248	3.191	3.025	3.844	0.241	38.613%

Other Nodes

Node	Comment	Initial Value/Mean	Final Value/Mean
Fresh		6.694	7.003
Flowery		7.306	7.576
Personality		6.298	6.858
Tenacious		6.645	7.016
Classical		6.066	6.241
Greedy		6.198	6.752
Sweet		6.298	6.850
Spiced		4.579	4.721
Wooded		4.678	4.798

Close | Save As... | Print | Save Scenario

This is an interesting example as it identifies that the JAR-type variable *Intensity* needs to be lowered to optimize *Purchase Intent* for Product 11.

It is important to reiterate that the sets of evidence we apply are not direct interventions in this domain. Hence, we are not performing causal inference. Instead, the sets of evidence we found help us prioritize our course of action for product and marketing initiatives.

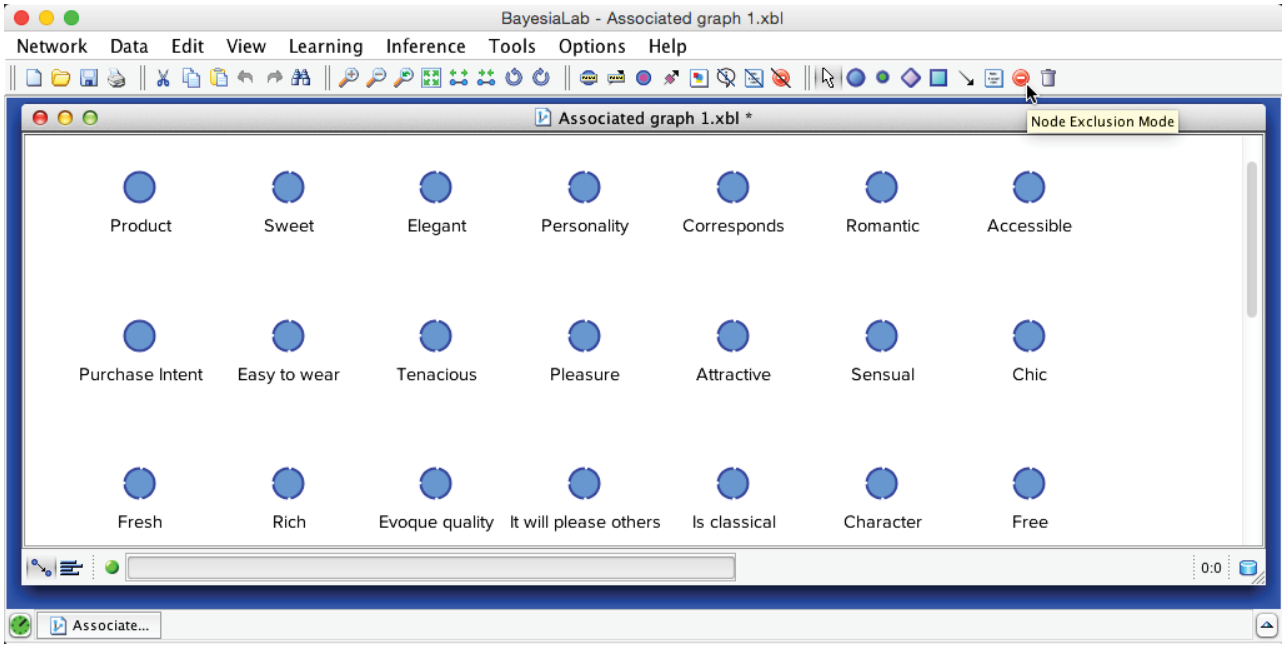
Summary

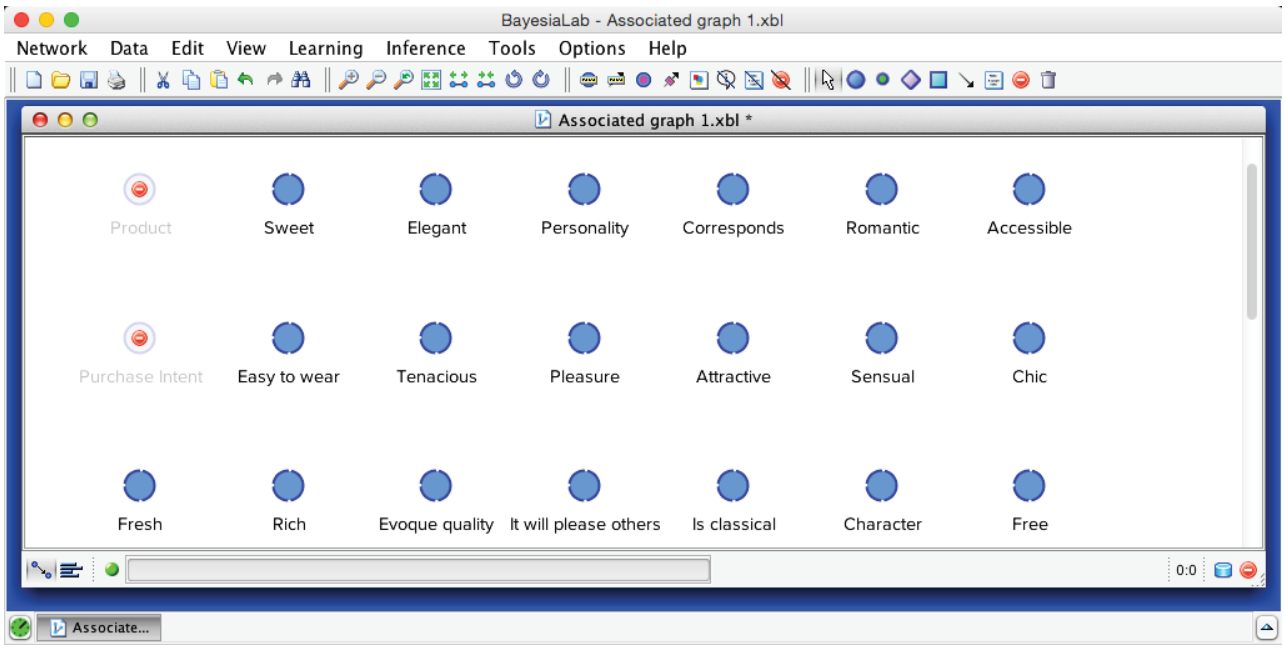
We presented a complete workflow that generates a Probabilistic Structural Equation Model for key driver analysis and product optimization. The Bayesian networks paradigm turned out to be a practical platform for developing the model and its subsequent analysis all the way through optimization. With all the steps contained in BayesiaLab, we have a single, continuous line of reasoning from raw survey data to a final order of priorities for action.

Step 1: Unsupervised Learning

As a first step, we need to exclude the node *Purchase Intent*, which will later serve as our Target Node. We do not want this node to become part of the structure that we will subsequently use for discovering hidden concepts. Likewise, we need to exclude the node *Product*, as it does not contain consumer feedback to be evaluated.

We can exclude nodes by selecting the `Node Exclusion Mode` and then clicking on the to-be-excluded node.





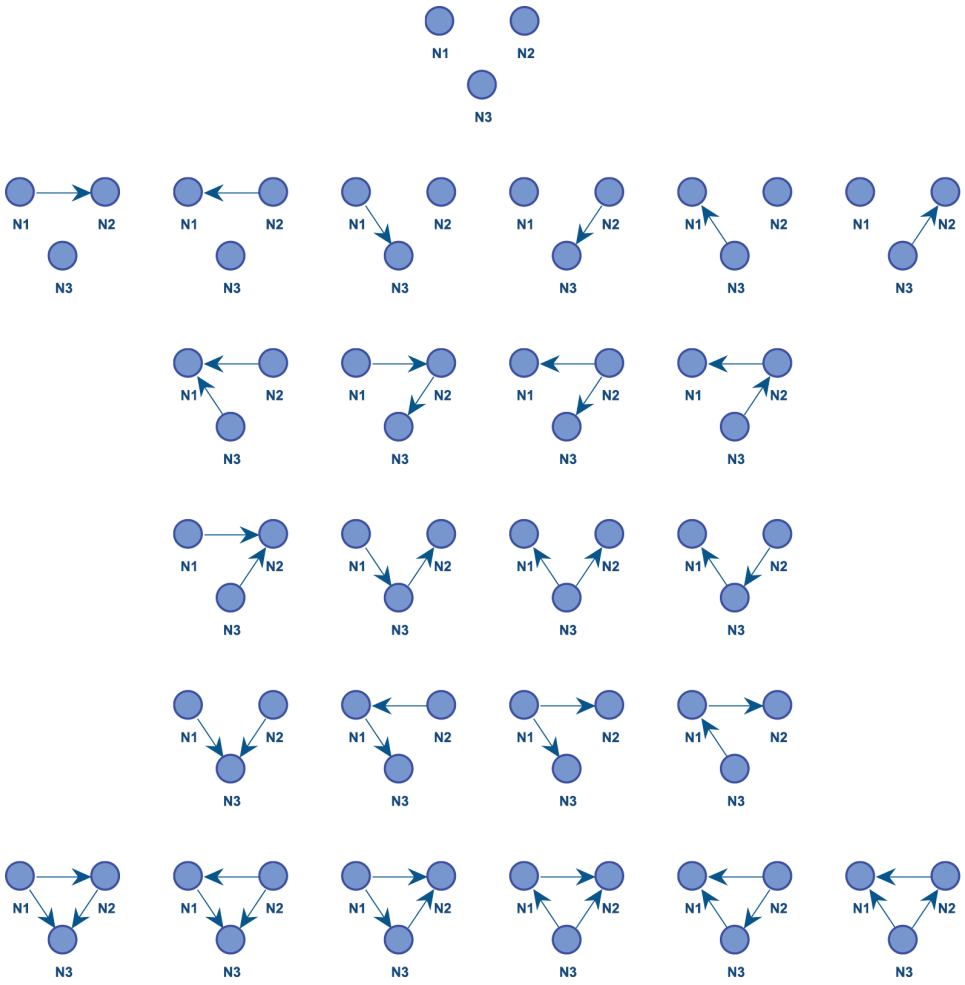
The upcoming series of steps is crucial. We now need to prepare a robust network on which we can later perform the clustering process. Given the importance, we recommend going through the full

range of Unsupervised Learning algorithms and comparing the performance of each resulting network structure to select the best structure.

The objective is to increase our chances of finding the optimal network for our purposes. Given that the number of possible networks grows super-exponentially with the number of nodes, this is a significant challenge.

Number of Nodes	Number of Possible Networks
1	1
2	3
3	25
4	543
5	29,281
6	3.7815×10^6
7	1.13878×10^9
8	7.83702×10^11
9	1.21344×10^15
10	4.1751×10^18
...	...
47	8.98454×10^376

It may not be immediately apparent how such an astronomical number of networks could be possible. The illustration below shows how 3 nodes can be combined in 25 ways to form a network.



Needless to say, generating all 9×10^376 networks–based on 47 nodes–and then selecting the best one is completely intractable (for reference, it is estimated that there are between 10^78 to 10^82 atoms in the known, observable universe). So, an exhaustive search would only be feasible for a small subset of nodes.

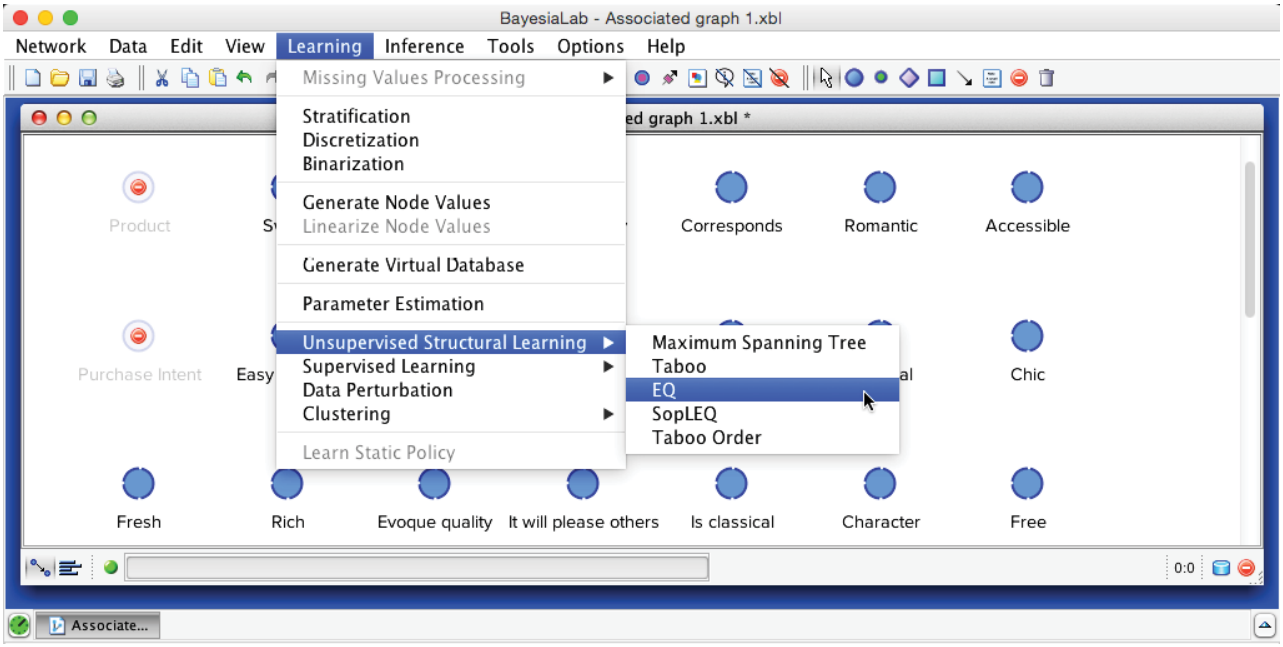
As a result, we have to use heuristic search algorithms to explore a small part of this huge space in order to find a local optimum. However, a heuristic search algorithm does not guarantee to find the global optimum. This is why BayesiaLab offers a range of distinct learning algorithms, which all use different search spaces and search strategies:

- Bayesian networks for Maximum Weight Spanning Tree and Taboo.
- Essential Graphs for EQ and SopLEQ, i.e., graphs with edges and arcs representing classes of equivalent networks.
- Order of nodes for Taboo Order.

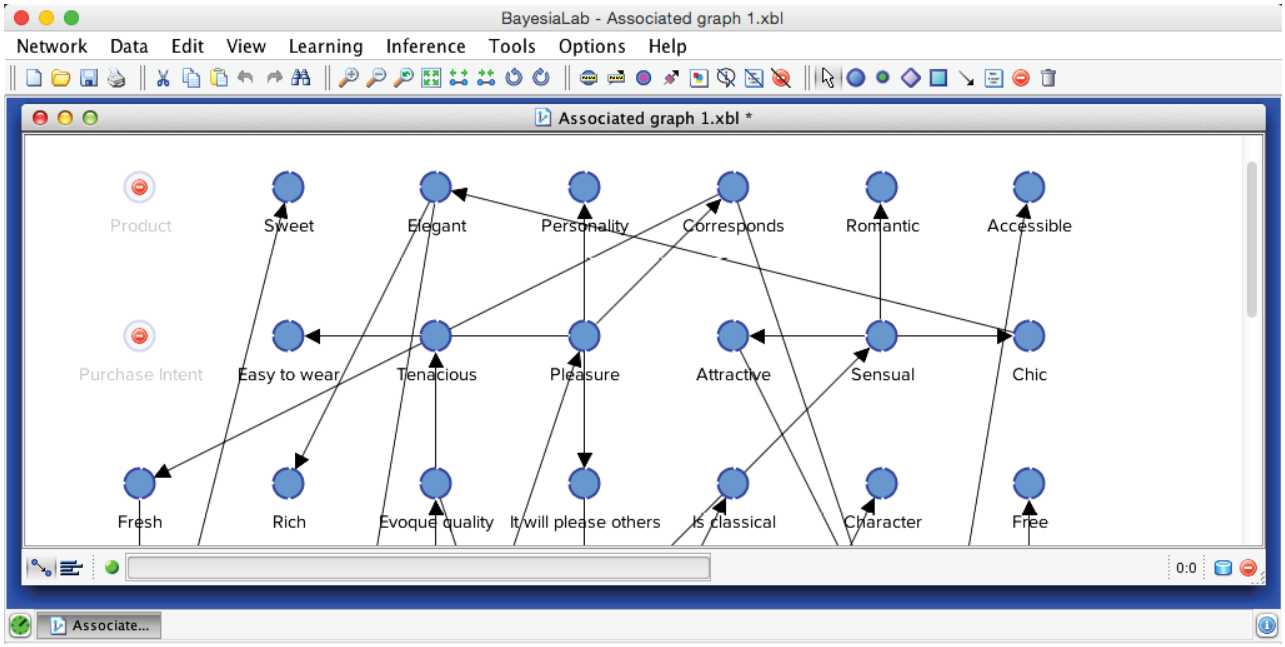
This diversity increases the probability of finding a solution close to the global optimum. Given adequate time and resources for learning, we recommend employing the algorithms in the following sequence to find the best solution:

- Maximum Weight Spanning Tree + Taboo
- Taboo ("from scratch")
- EQ ("from scratch") + Taboo
- SopLEQ + Taboo
- Taboo Order + Taboo

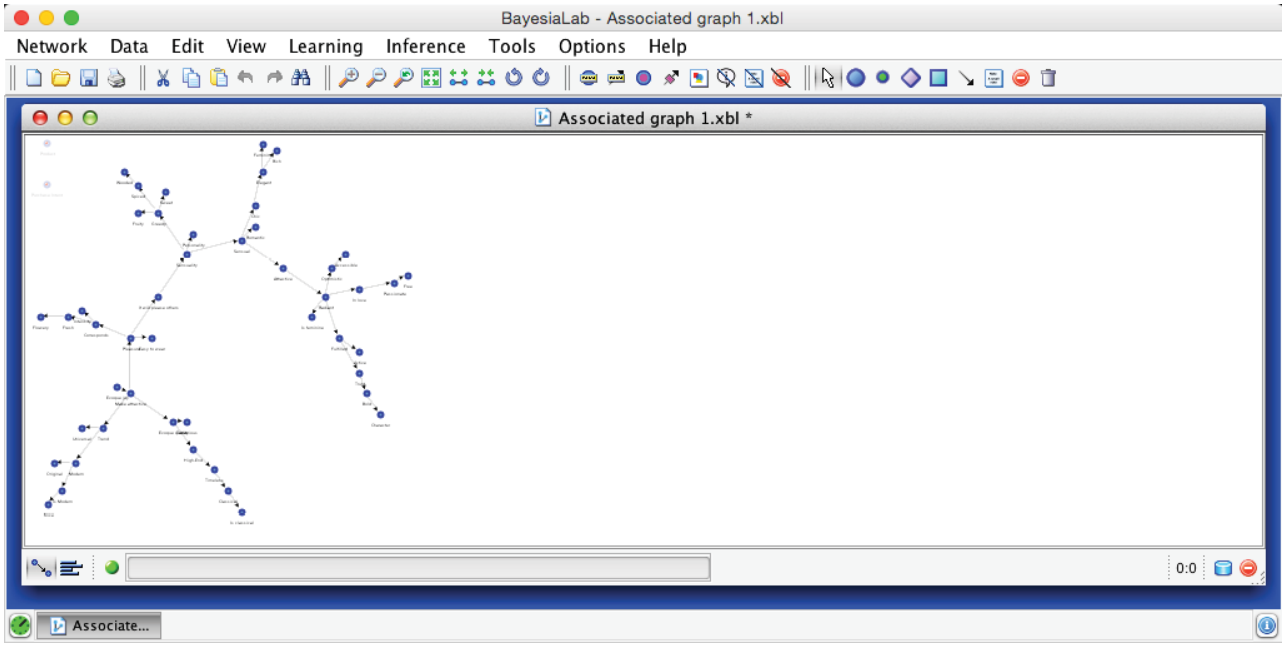
However, to keep the presentation compact, we only illustrate the learning steps for the EQ algorithm: `Menu > Learning > Unsupervised Structural Learning > EQ`.



The network generated by the EQ algorithm is shown below.



Pressing P and then clicking the Best-Fit icon, provides an interpretable view of the network. Additionally, rotating the network graph with the Rotate Left and Rotate Right buttons can help set a suitable view.



We now need to assess the quality of this network. Each of BayesiaLab's Unsupervised Learning algorithms uses the Minimum Description Length (MDL) Score –internally–as the measure to optimize

while searching for the best possible network. However, we can also employ the MDL Score for explicitly rating the quality of a network.

Minimum Description Length

The Minimum Description Length (MDL) Score is a two-component score, which has to be minimized to obtain the best solution. It has been used traditionally in the artificial intelligence community for estimating the number of bits required for representing (1) a "model" and (2) "data given this model."

In our machine-learning application, the "model" is a Bayesian network consisting of a graph and probability tables. The second term is the log-likelihood of the data given the model, which is inversely proportional to the probability of the observations (data) given the Bayesian network (model). More formally, we write this as:

$$MDL(B, D) = \alpha \times DL(B) + DL(D \mid B)$$

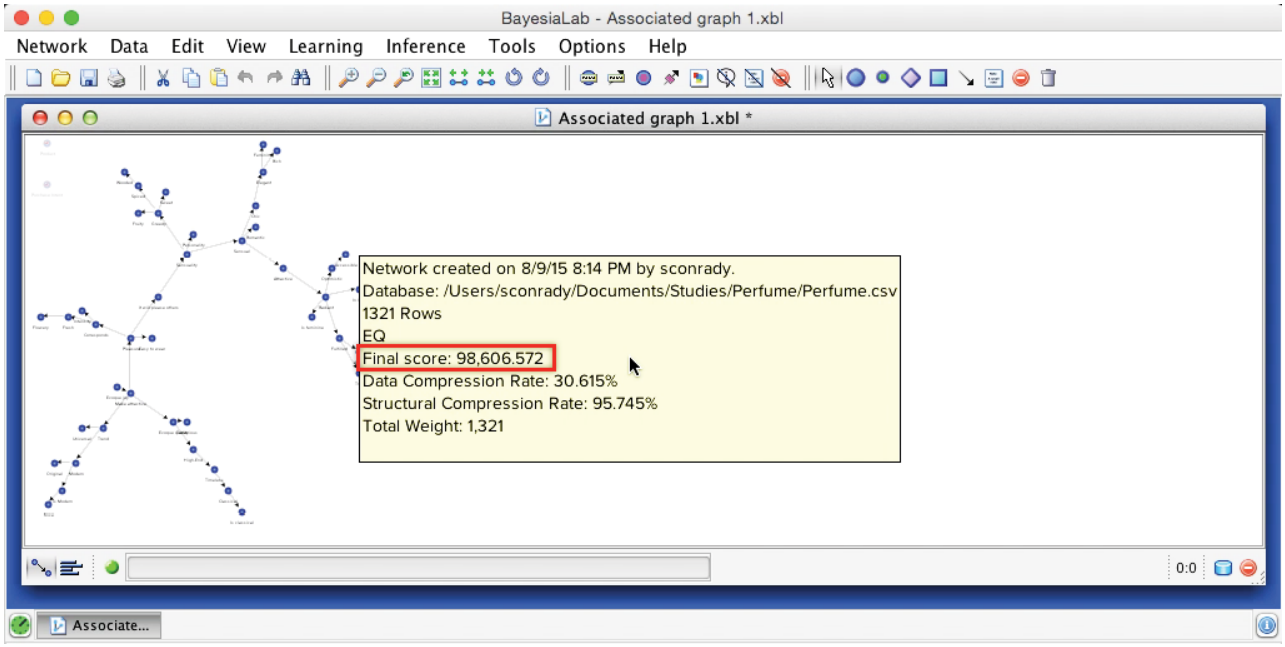
- αrepresents BayesiaLab's Structural Coefficient (the default value is 1), a parameter that permits changing the weight of the structural part of the MDL Score (the lower the value of α, the greater the complexity of the resulting networks),
- $DL(B)$ is the number of bits to represent the Bayesian network B (graph and probabilities), and
- $DL(D \mid B)$ is the number of bits representing the dataset D given the Bayesian network B (likelihood of the data given the Bayesian network).

The minimum value for the first term, $DL(B)$, is obtained with the simplest structure, i.e., the fully unconnected network, in which all variables are stated as independent. The minimum value for the second term, $DL(D \mid B)$, is obtained with the fully connected network, i.e., a network corresponding to the analytical form of the joint probability distribution, in which no structural independencies are stated.

Thus, minimizing this score consists of finding the best trade-off between both terms. For a learning algorithm that starts with an unconnected network, the objective is to add a link for representing a probabilistic relationship if, and only if, this relationship reduces the log-likelihood of the data, i.e., $DL(D \mid B)$ by a large enough amount to compensate for the increase in the size of the network representation, i.e., $DL(B)$.

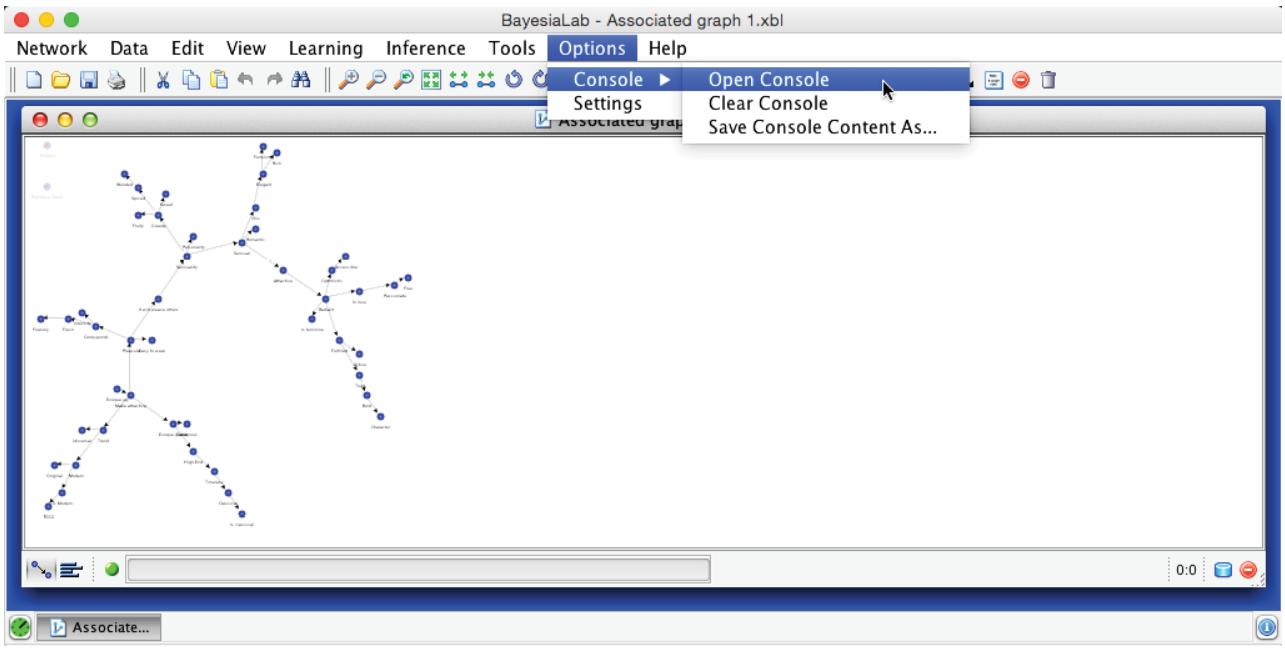
MDL Score Comparison

We now use the MDL Score to compare the results of all learning algorithms. We can look up the MDL Score of the current network by pressing W while hovering the cursor over the Graph Panel. This brings up an info box that reports a number of measures, including the MDL Score, which is displayed here as the "Final Score."

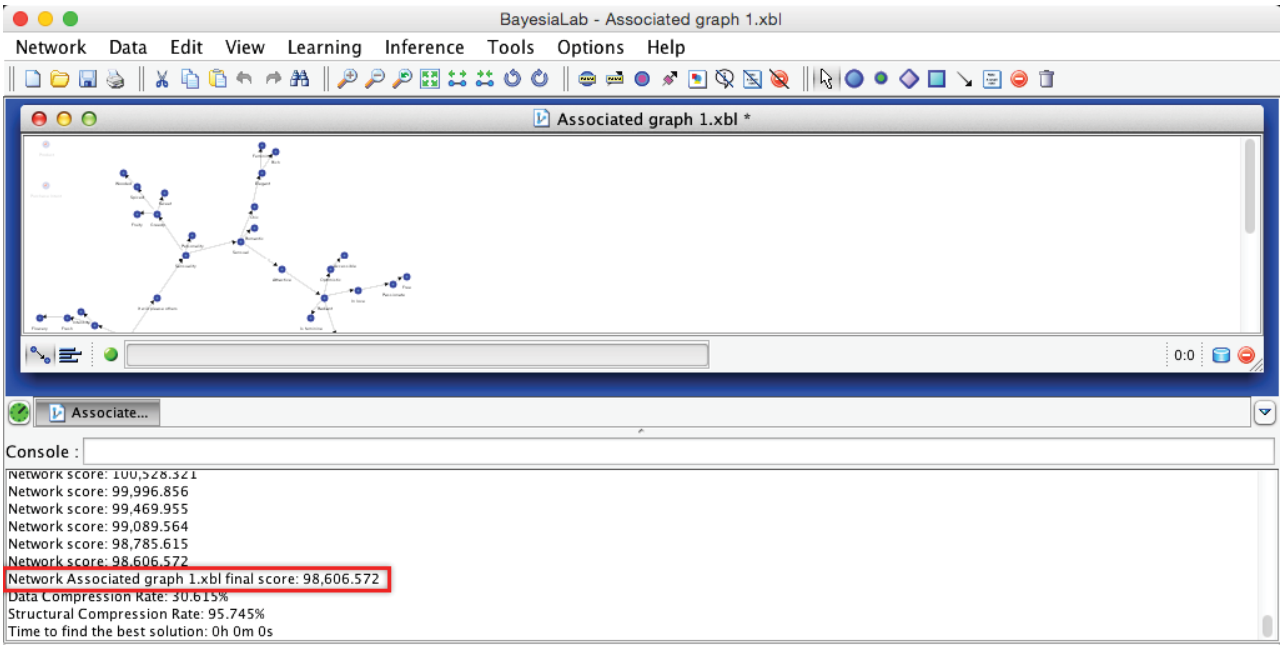


> The MDL score can only be compared for networks with precisely the same representation of all variables, i.e., with the same discretization thresholds and the same data.

Alternatively, we can open up the Console via `Menu > Options > Console > Open Console`:



The Console maintains a kind of "log" that keeps track of the learning progress by recording the MDL Score (or "Network Score") at each step of the learning process. Here, the "Final Score" marks the MDL Score of the current network, which is what we need to select the network with the lowest value.

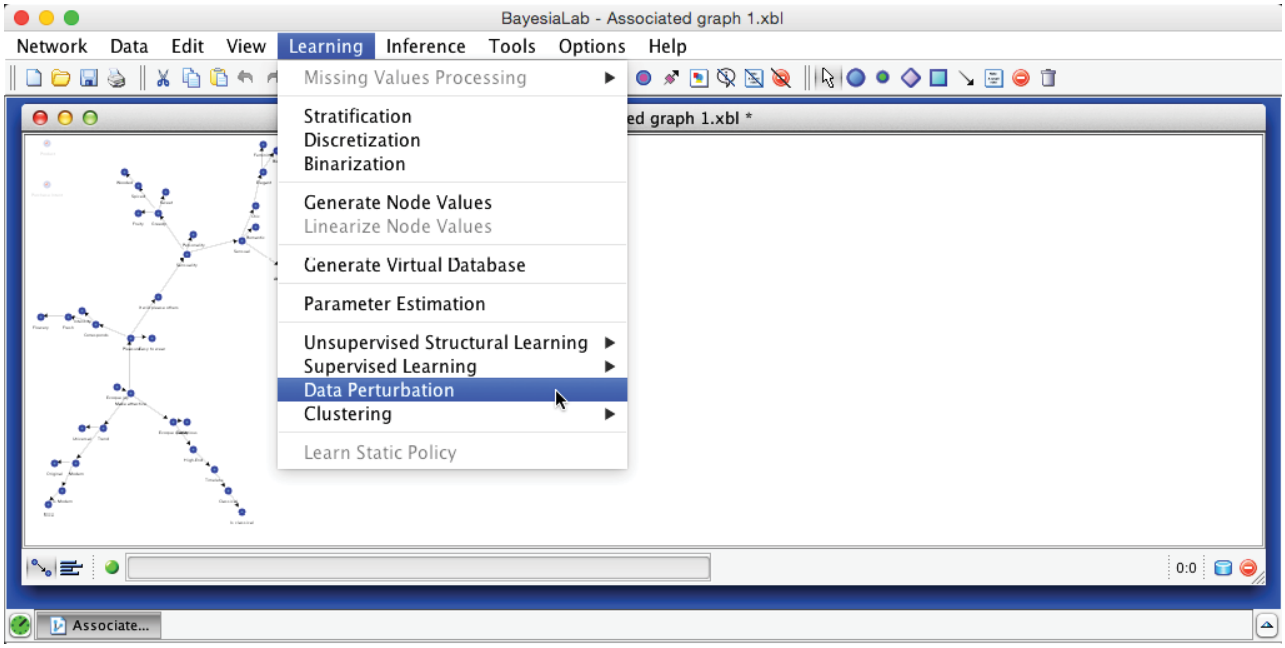


The EQ algorithm produces a network with an MDL Score of 98,606.572. As it turns out, this performance is on par with all the other algorithms we considered, although we skip presenting the details in this chapter. Given this result, we can proceed with the EQ-learned network to the next step.

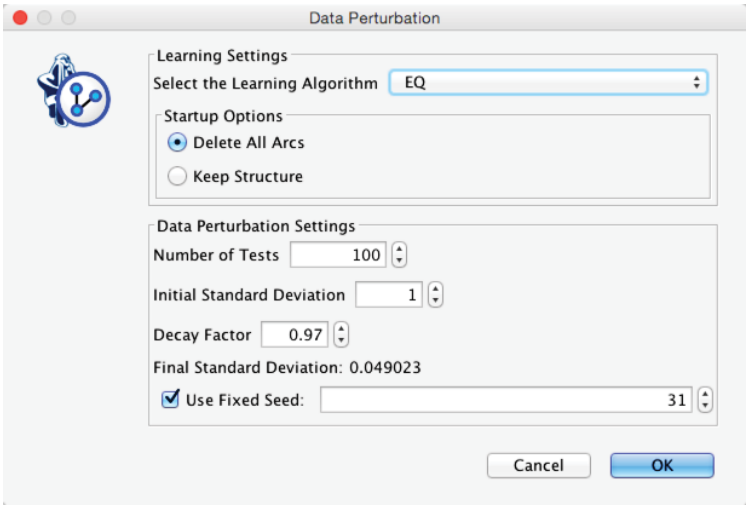
Data Perturbation

As a further safeguard against utilizing a sub-optimal network, BayesiaLab offers Data Perturbation, which is an algorithm that adds random noise (from within the interval [-1,1]) to the weight of each observation in the dataset.

In the context of our learning task, Data Perturbation can help us escape from local minima, which we could have encountered during learning. We start this algorithm by selecting `Learning > Data Perturbation`.



For Data Perturbation, we need to set a number of parameters. The additive noise is always generated from a Normal distribution with a mean of 0, but we must set the Initial Standard Deviation. A Decay Factor defines the exponential attenuation of the standard deviation with each iteration.



Upon completing Data Perturbation, we see the newly learned network in the Graph Panel. Once again, we can retrieve the score by pressing W while hovering with the cursor over the Graph Panel or by looking it up in the Console. The score remains unchanged at 98,606.572. We can now be reasonably confident that we have found the optimal network given the original choice of discretization, i.e., the most compact representation of the joint probability distribution defined by the 47 manifest variables.

On this basis, we now switch to the Validation Mode F5. Instead of examining individual nodes, however, we proceed directly to Variable Clustering.

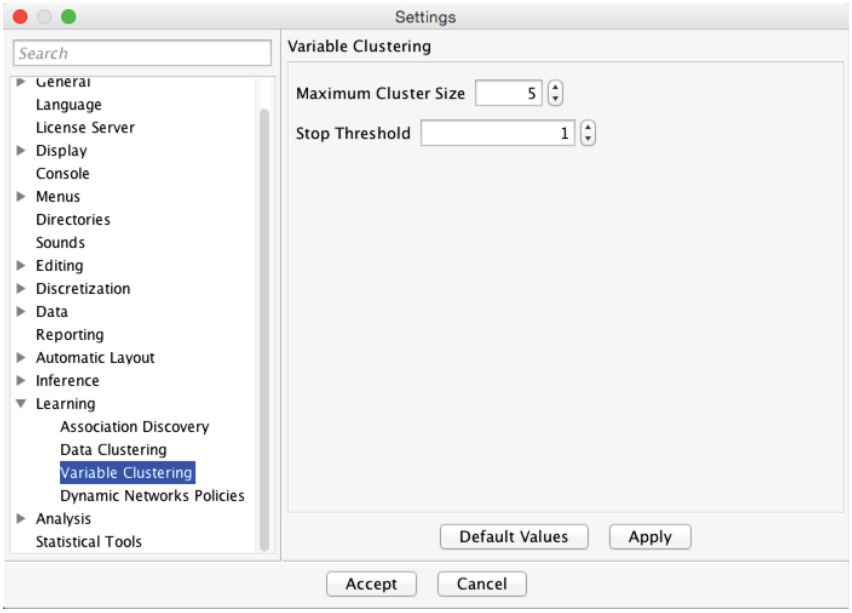
Step 2: Variable Clustering

BayesiaLab's Variable Clustering is a hierarchical agglomerative clustering algorithm that uses Arc Force (i.e., the Kullback-Leibler Divergence) for computing the distance between nodes.

At the start of Variable Clustering, each manifest variable is treated as a distinct cluster. The clustering algorithm proceeds iteratively by merging the "closest" clusters into a new cluster. Two criteria are used for determining the number of clusters:

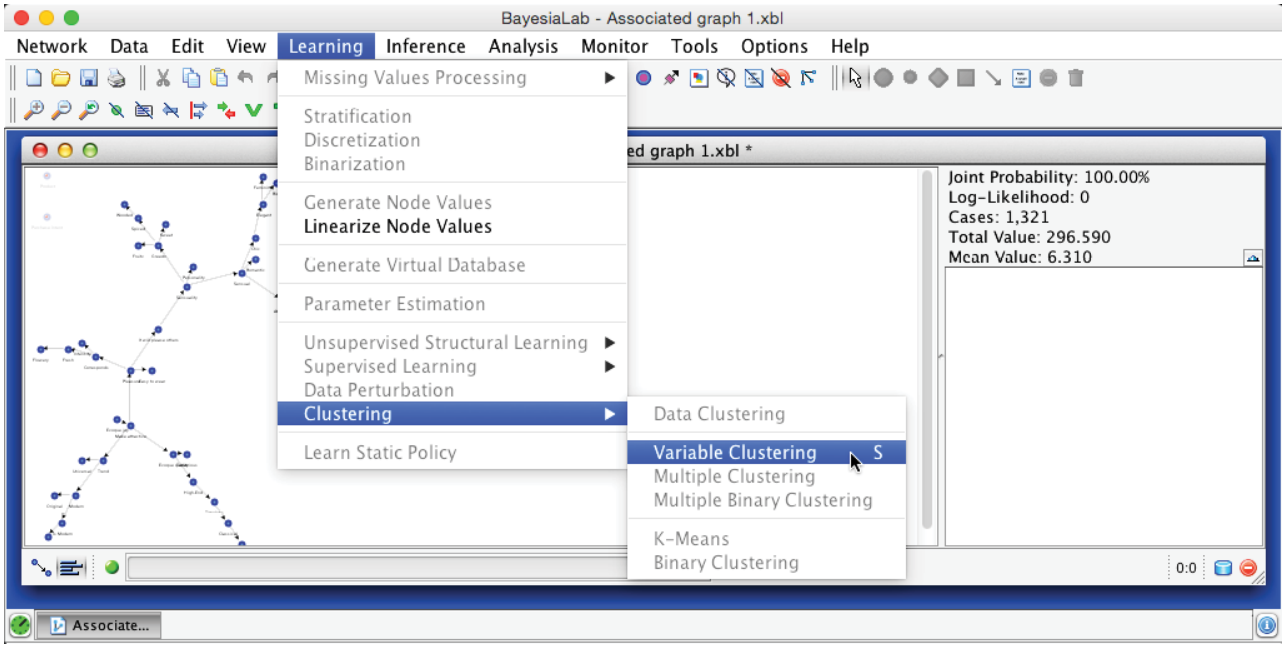
- Stop Threshold: a minimum Arc Force value below which clusters are not merged (a kind of significance threshold).
- Maximum Cluster Size: the maximum number of variables per cluster.

These criteria can be set via `Menu > Options > Settings > Learning > Variable Clustering`:

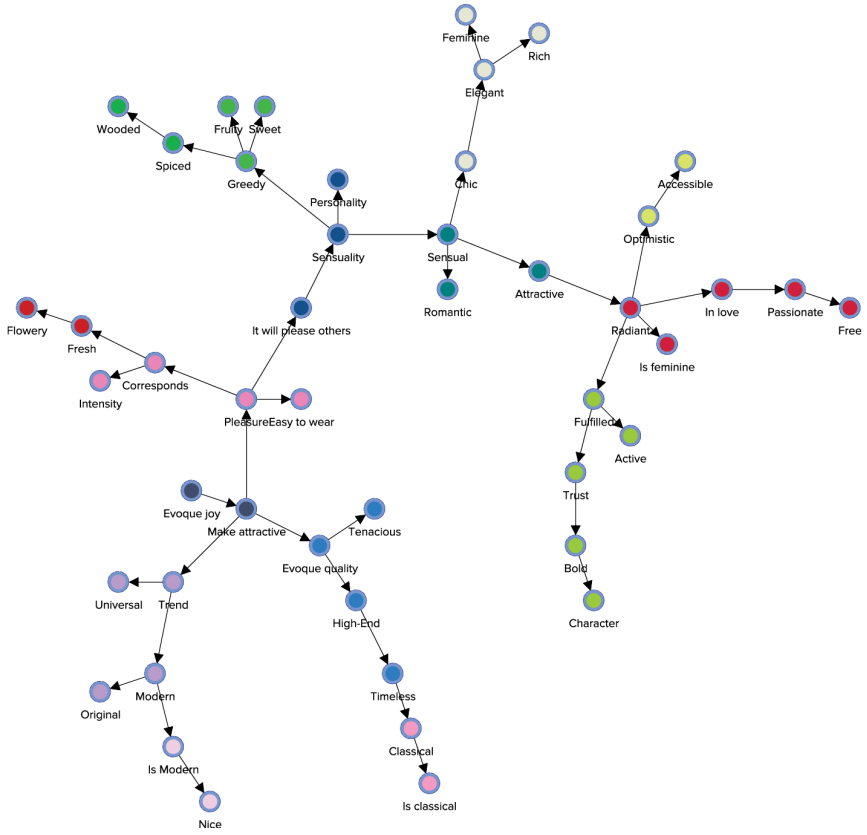


We do not advise changing the Stop Threshold, but Maximum Cluster Size is more subjective. For building PSEMs, we recommend a value between 5 and 7, for reasons that will become clear when we show how latent variables are generated. If, however, the goal of Variable Clustering is dimensionality reduction, we suggest increasing the Maximum Cluster Size to a much higher value, thus effectively eliminating it as a constraint.

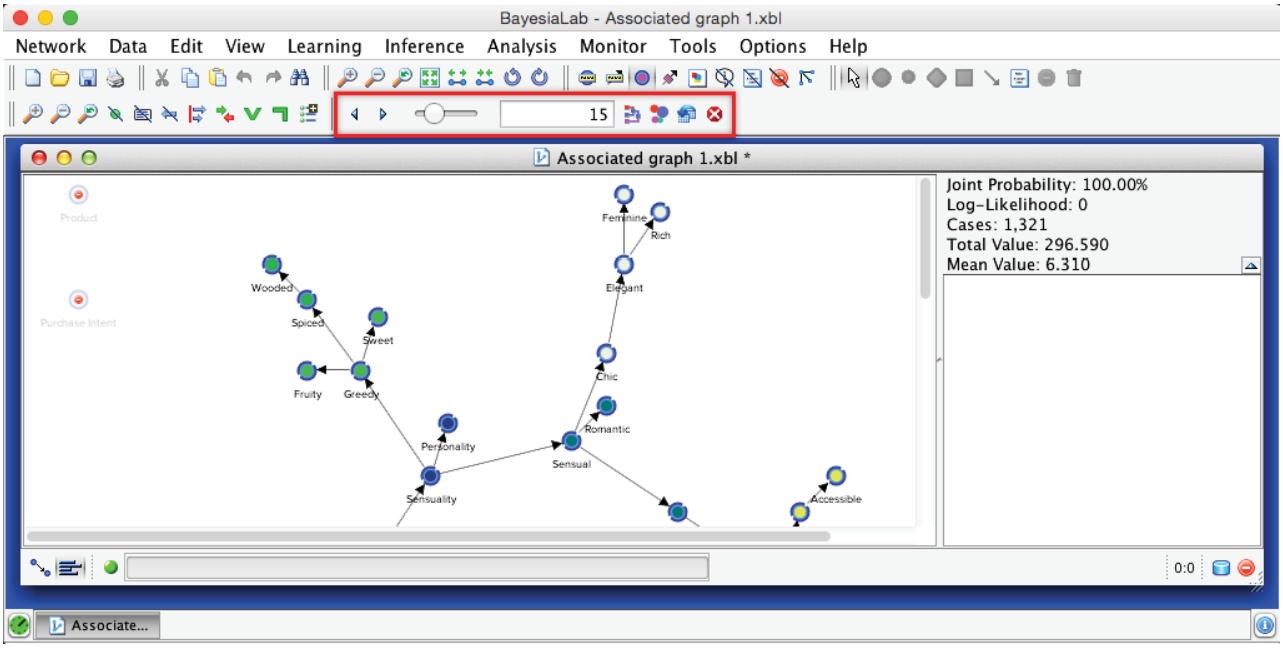
The Variable Clustering algorithm can be started via Main `Menu > Learning > Clustering > Variable Clustering` or by using the shortcut S.



In this example, BayesiaLab identified 15 clusters, and each node is now color-coded according to its cluster membership. The following image shows the standalone graph–outside the `BayesiaLab` window for better legibility.

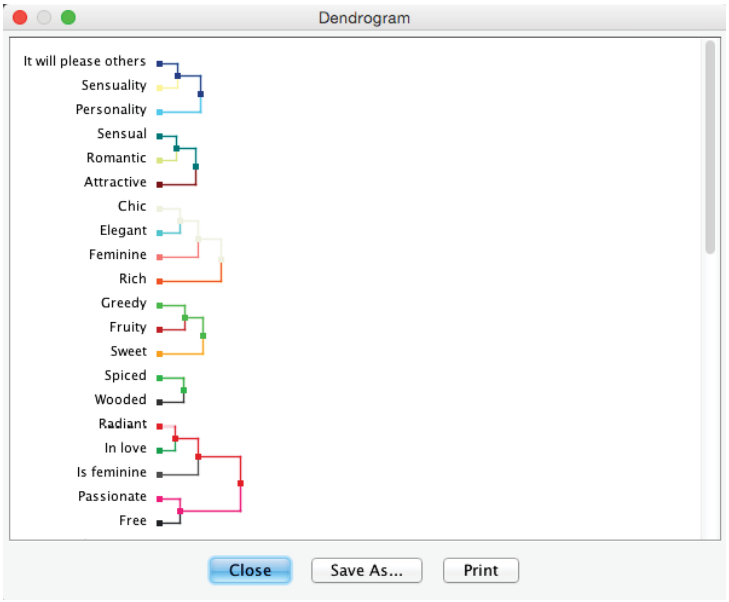


BayesiaLab offers several tools for examining and editing the proposed cluster structure. They are accessible from an extended menu bar (highlighted in the screenshot below).

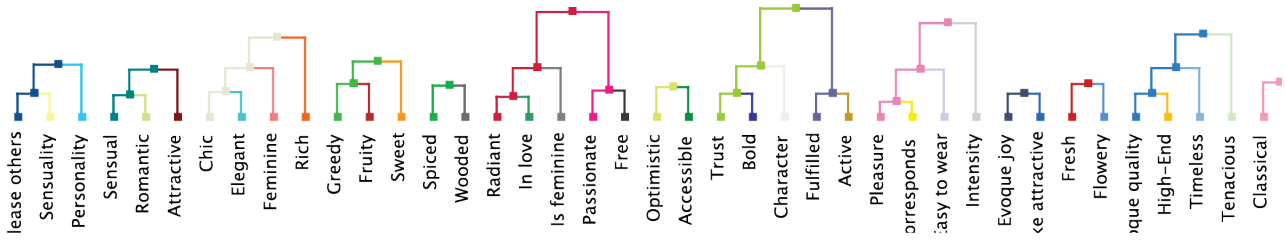


Dendrogram

The Dendrogram allows us to review the linkage of nodes within variable clusters. It can be activated via the corresponding icon in the extended menu. The lengths of the branches in the Dendrogram are proportional to the Arc Force between clusters.

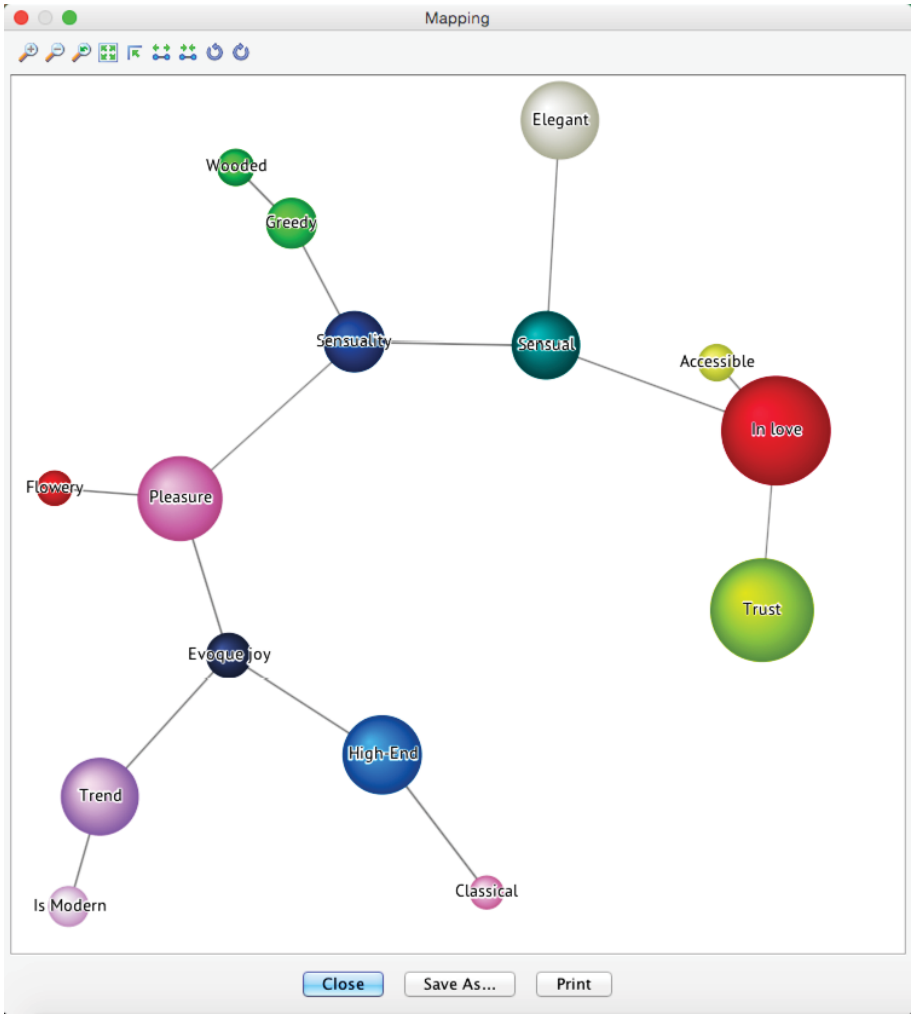


Also, the Dendrogram can be copied directly as a vector or bitmap graphic by right-clicking on it. Alternatively, it can be exported in various formats via the Save As... button. As such, it can be imported into documents and presentations. This ability to copy and paste graphics applies to most graphs, plots, and charts in BayesiaLab.

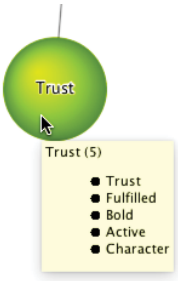


Cluster Mapping

Mapping offers an intuitive approach to examining the just-discovered cluster structure as an alternative to Dendrogram. It can be activated via the `Mapping` button in the menu bar.

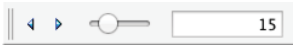


By hovering over any cluster "bubbles" with the cursor, BayesiaLab displays a list of all manifest nodes connected to that particular cluster. Each list of manifest variables is sorted according to the intra-cluster Node Force. This also explains the names displayed on the clusters. By default, each cluster takes on the name of the strongest manifest variable.



Number of Clusters

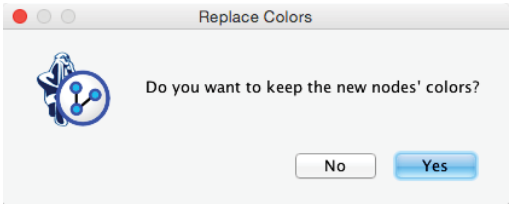
As explained earlier, BayesiaLab uses two criteria to determine the default number of clusters. We can change this number via the selector in the menu bar.



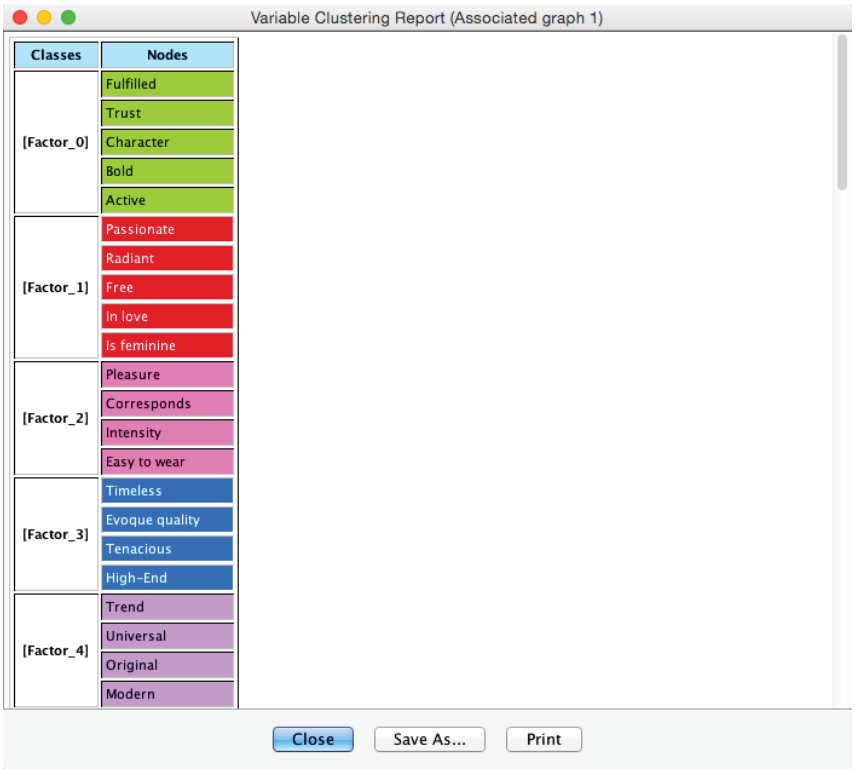
The Dendrogram and the `Mapping` view respond dynamically to any changes to the number of clusters.

Cluster Validation

The result of the Variable Clustering algorithm is purely descriptive. Once the question regarding the number of clusters is settled, we need to formally confirm our choice by clicking the `Validate Clustering` button in the toolbar. Only then can we trigger the creation of one Class per Cluster. At that time, all nodes become associated with unique Classes named $Factor_i$, with i representing the identifier of the factor. Additionally, we are prompted to confirm that we wish to keep the node colors generated during clustering.



The Clusters are now saved, and the color-coding is formally associated with the nodes. A Clustering Report provides a formal summary of the new Factors and their associated Manifest variables.

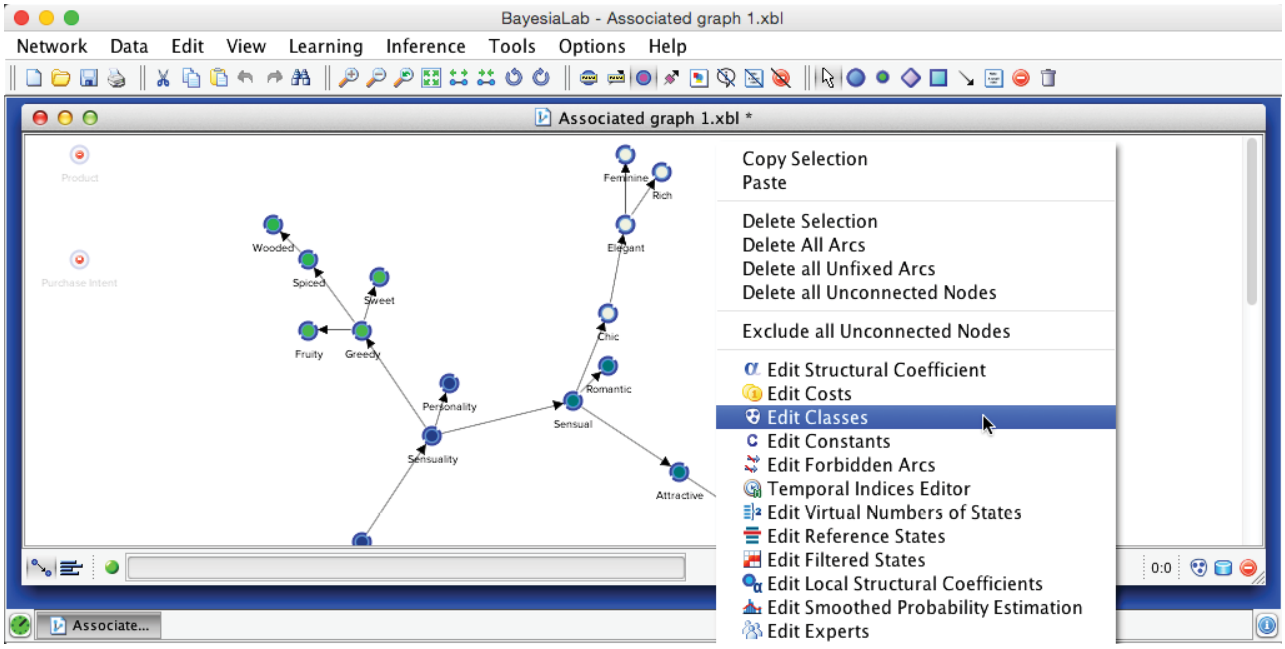


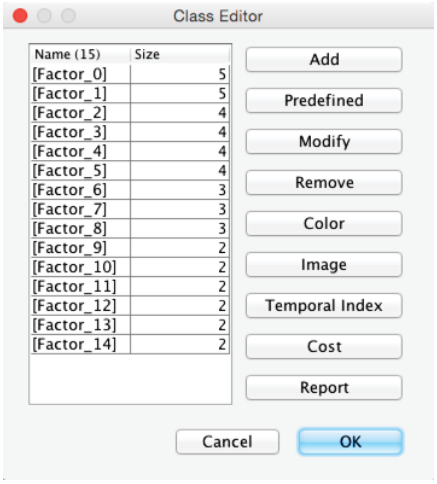
Note that we use the following terms interchangeably: "derived concept," "unobserved latent variable," "hidden cause," and "extracted factor."

The Classes icon in the lower right-hand corner of the window confirms that Classes have been created corresponding to the Factors. This concludes Step 2, and we formally close Variable Clustering via the stop icon on the extended toolbar.

Editing Factors

Beyond our choice with regard to the number of Clusters, we also have the option of using our domain knowledge to modify which Manifest Nodes belong to specific Factors. This can be done by right-clicking on the Graph Panel and selecting Edit Classes, and then Modify from the Contextual Menu. Alternatively, we can click the Classes icon . In our example, however, we show the Class Editor just for reference, as we keep all the original variable assignments in place.

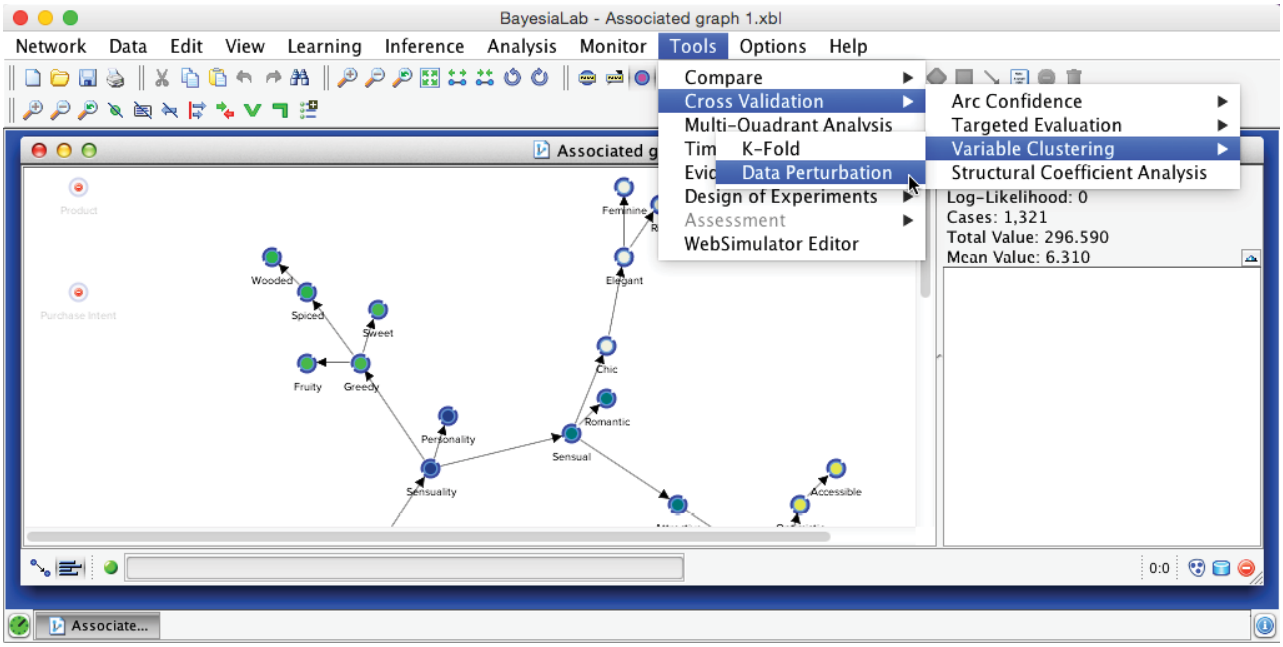




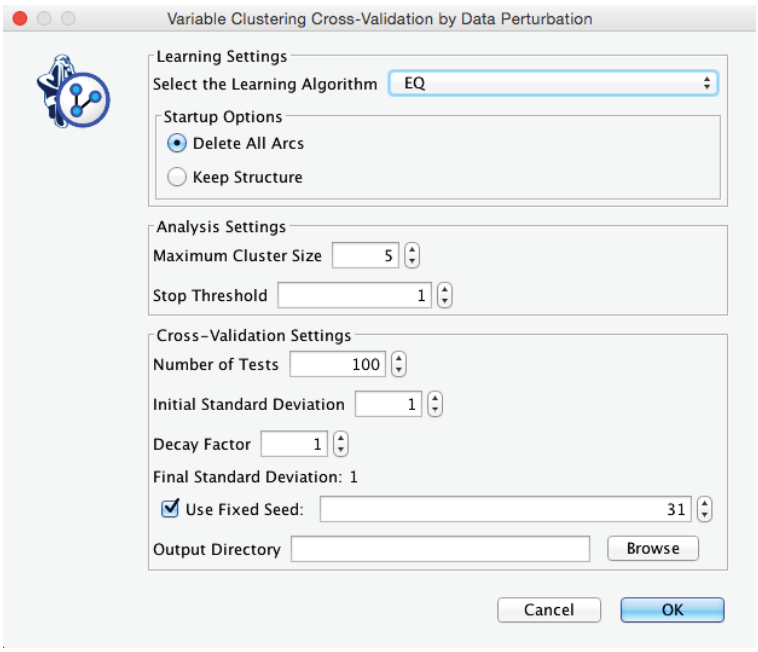
Cluster Cross-Validation

We now examine the robustness of the identified factors, i.e., how these factors respond to changes in sampling. This is particularly important for studies that are regularly repeated with new data, e.g., annual customer satisfaction surveys. Inevitably, survey samples are going to be different between the years. As a result, machine learning will probably discover somewhat different structures each time and, therefore, identify different clusters of nodes. Therefore, it is important to distinguish between a sampling artifact and a substantive change in the joint probability distribution. In the context of our example, the latter would reveal a structural change in consumer behavior.

We start the validation process via `Menu > Tools > Cross-Validation > Variable Clustering > Data Perturbation`.

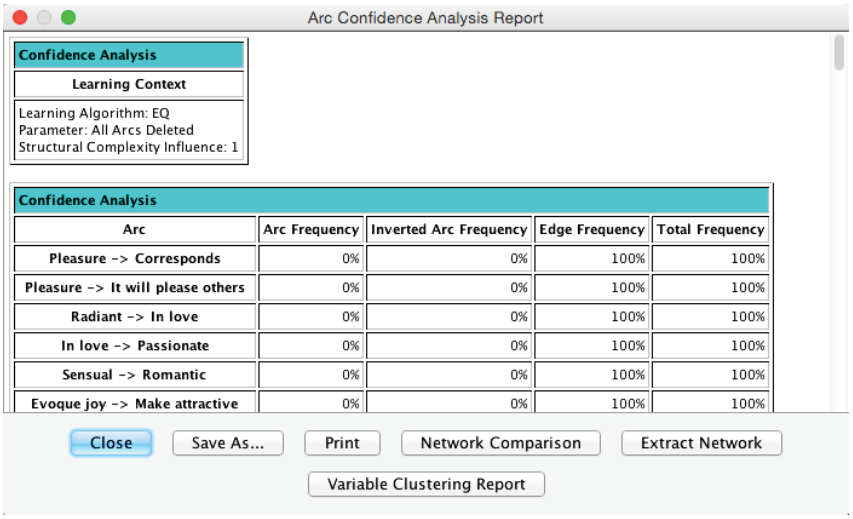


This brings up the dialogue box shown below.



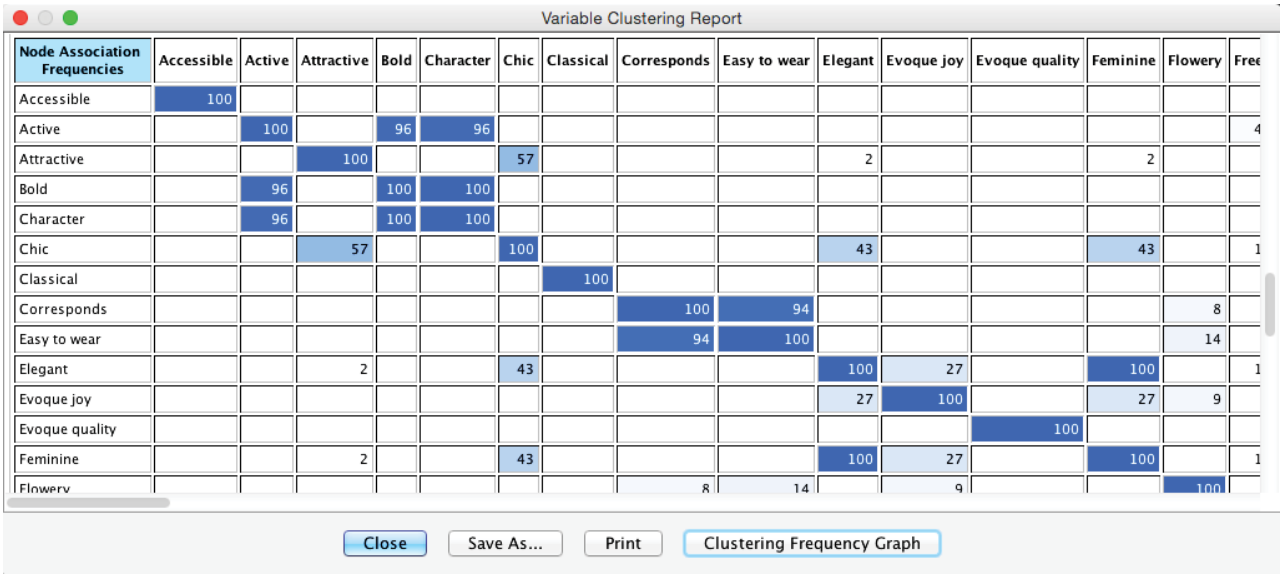
These settings specify that BayesiaLab will learn 100 networks with EQ and perform Variable Clustering on each of them, all while maintaining the constraint of a maximum of 5 nodes per cluster

without any attenuation of the perturbation. Upon completion, we obtain a report panel, from which we initially select `Variable Clustering Report`.



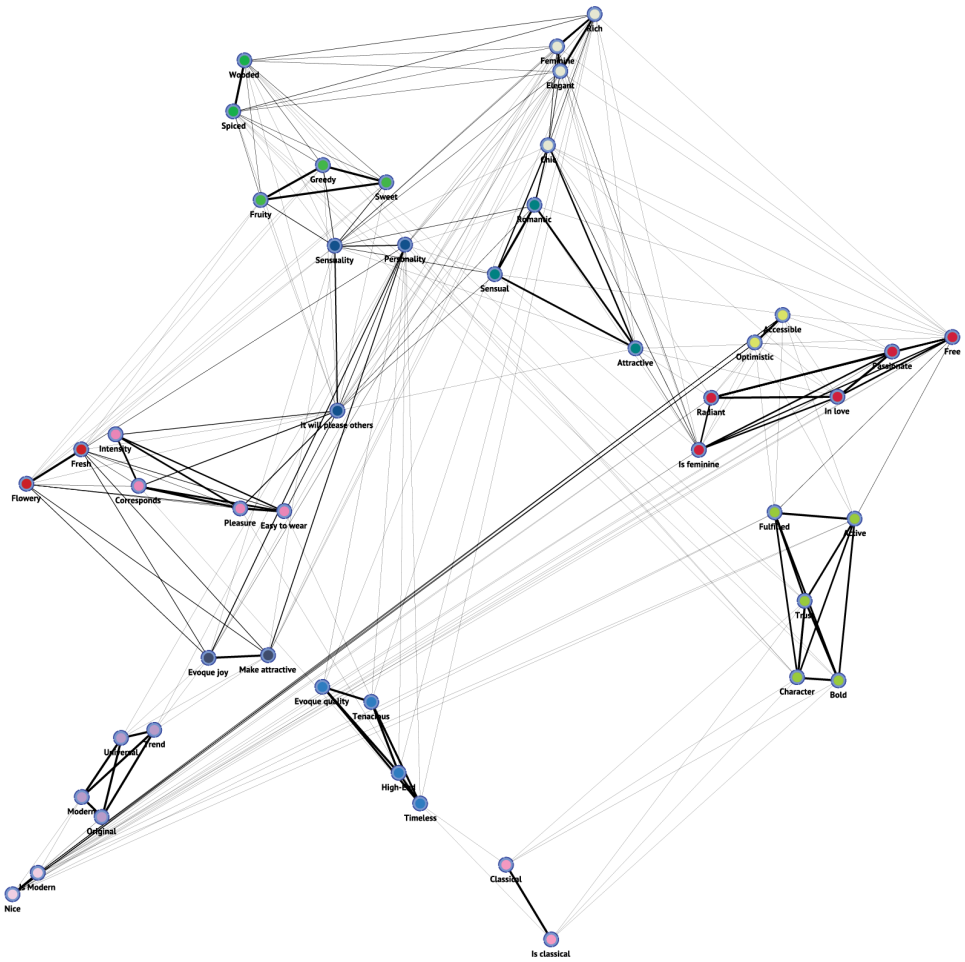
The Variable Clustering Report consists primarily of two large tables. The first table in the report shows the cluster membership of each node in each network (only the first 12 columns are shown). Here, thanks to the colors, we can easily detect whether nodes remain clustered together between iterations or whether they "break up."

The second table shows how frequently individual nodes are clustered together.



Node Association Frequencies	Accessible	Active	Attractive	Bold	Character	Chic	Classical	Corresponds	Easy to wear	Elegant	Evoque joy	Evoque quality	Feminine	Flowery	Free
Accessible	100														
Active		100		96	96										4
Attractive			100			57				2			2		
Bold		96		100	100										
Character		96		100	100										
Chic			57			100				43			43		1
Classical							100								
Corresponds								100	94					8	
Easy to wear								94	100					14	
Elegant			2			43				100	27		100		1
Evoque joy										27	100		27	9	
Evoque quality												100			
Feminine			2			43				100	27		100		1
Flowery								8	14		9			100	

Clicking the `Clustering Frequency Graph` provides yet another visualization of the clustering patterns. The thickness of the lines is proportional to the frequency of nodes being in the same Cluster. Equally important for interpretation is the absence of lines between nodes. For instance, the absence of a line between *Flowery* and Modern says that they have never been clustered together in any of the 100 samples. If they were to cluster together in future iteration with new survey data, it would probably reflect a structural change in the market rather than a data sampling artifact.



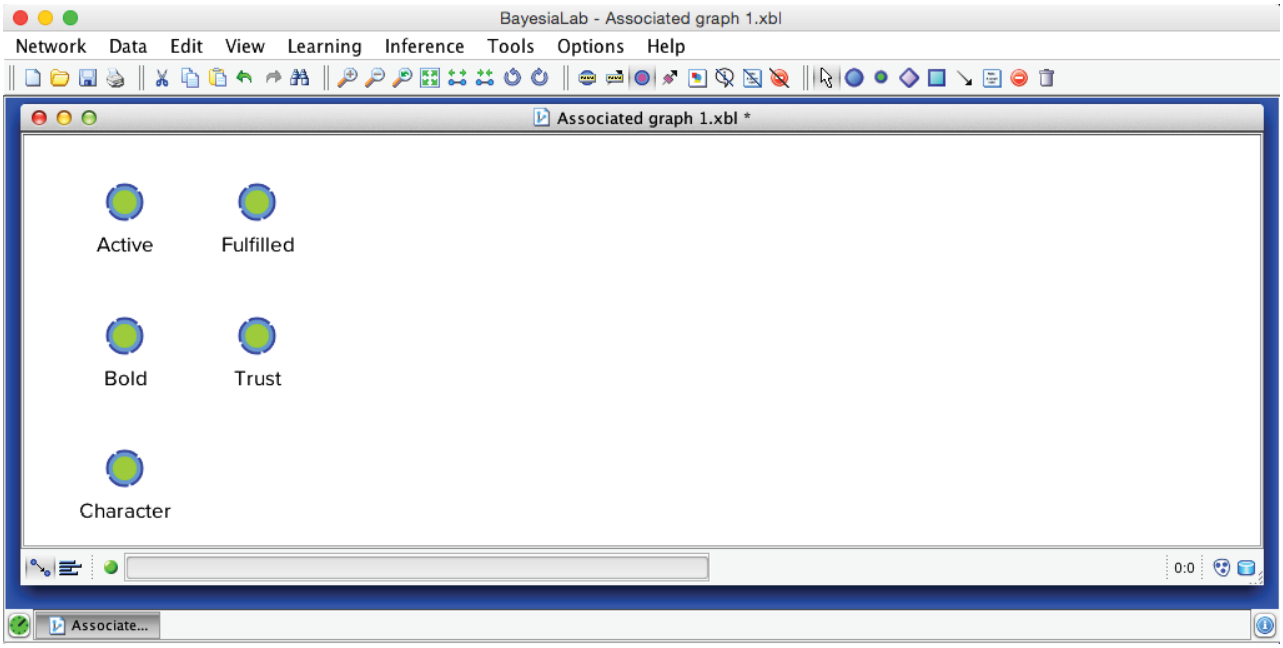
Step 3: Multiple Clustering

The Cluster Cross-Validation was merely a review, and it did not change the factors we confirmed when we clicked the `Validate Clustering` button. Although we have defined these Factors in terms of Classes of Manifest variables, we still need to create the corresponding latent variables via Multiple Clustering. This process creates one discrete Factor for each Cluster of variables by performing Data Clustering on each subset of clustered manifest variables.

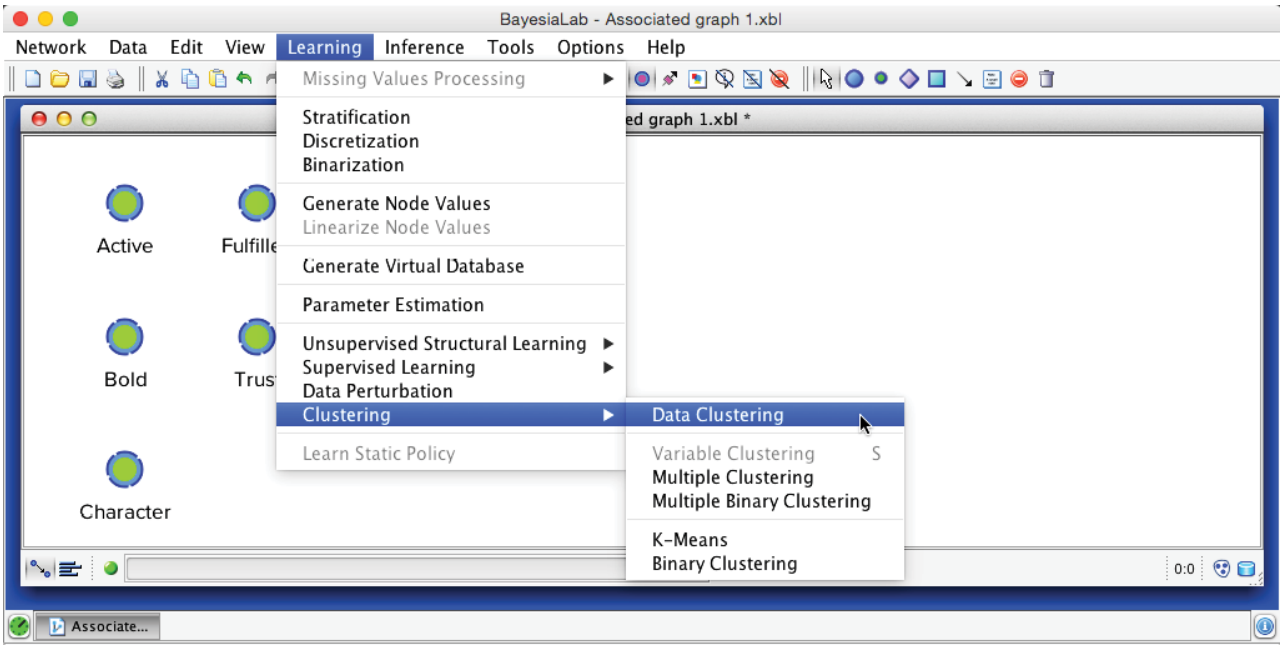
In traditional statistics, deriving such latent variables or factors is typically performed by means of Factor Analysis, e.g., Principal Components Analysis (PCA).

Data Clustering

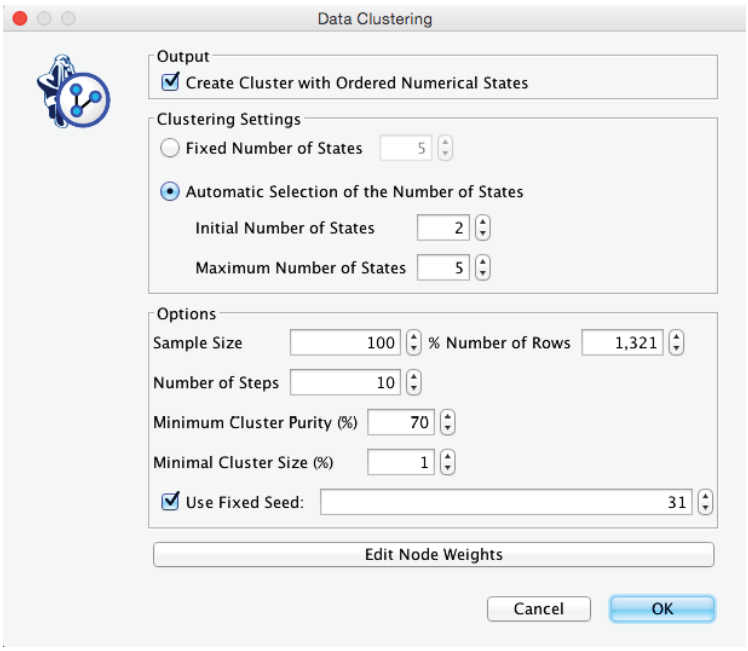
Before we run this automatically across all factors with the Multiple Clustering algorithm, we will demonstrate the process on a single cluster of nodes, namely the nodes associated with Factor_0: *Active*, *Bold*, *Character*, *Fulfilled*, and *Trust*. We simply delete all other nodes and arcs and save this subset of nodes as a new, separate XBL file.



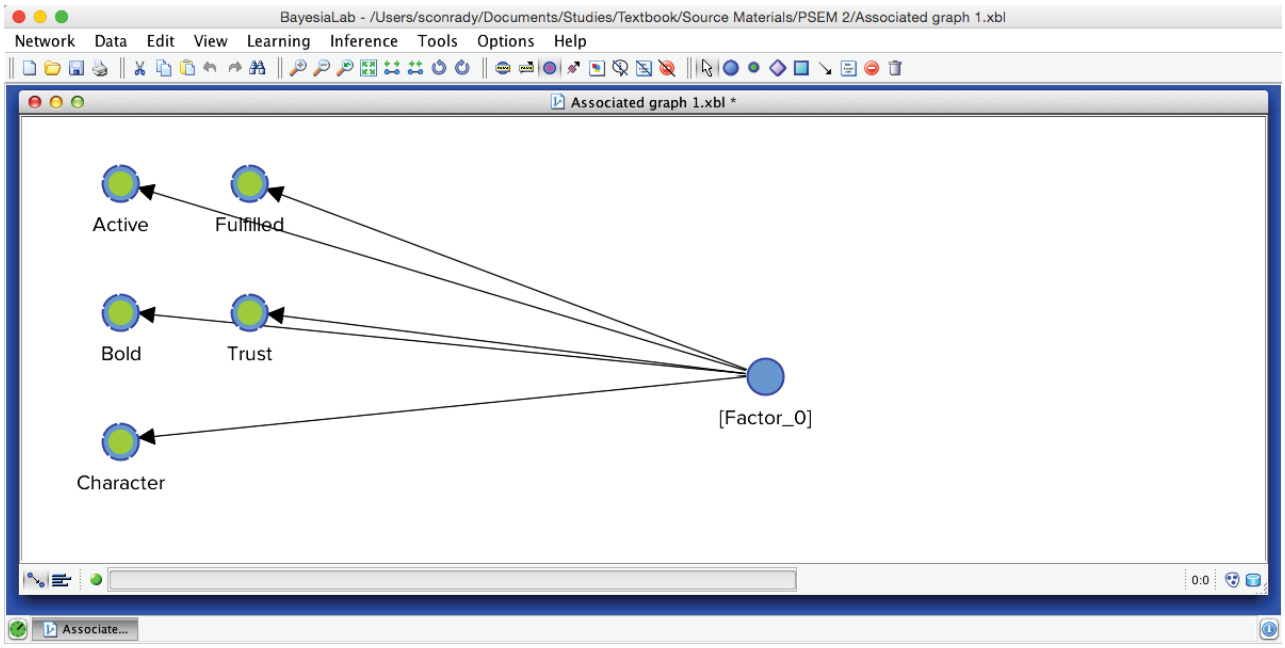
The objective of BayesiaLab's Data Clustering algorithm is to create a node that compactly represents the joint probability distribution defined by the variables of interest. We start Data Clustering via `Menu > Learning > Clustering > Data Clustering`. Unless we select a subset, Data Clustering will be applied to all nodes.



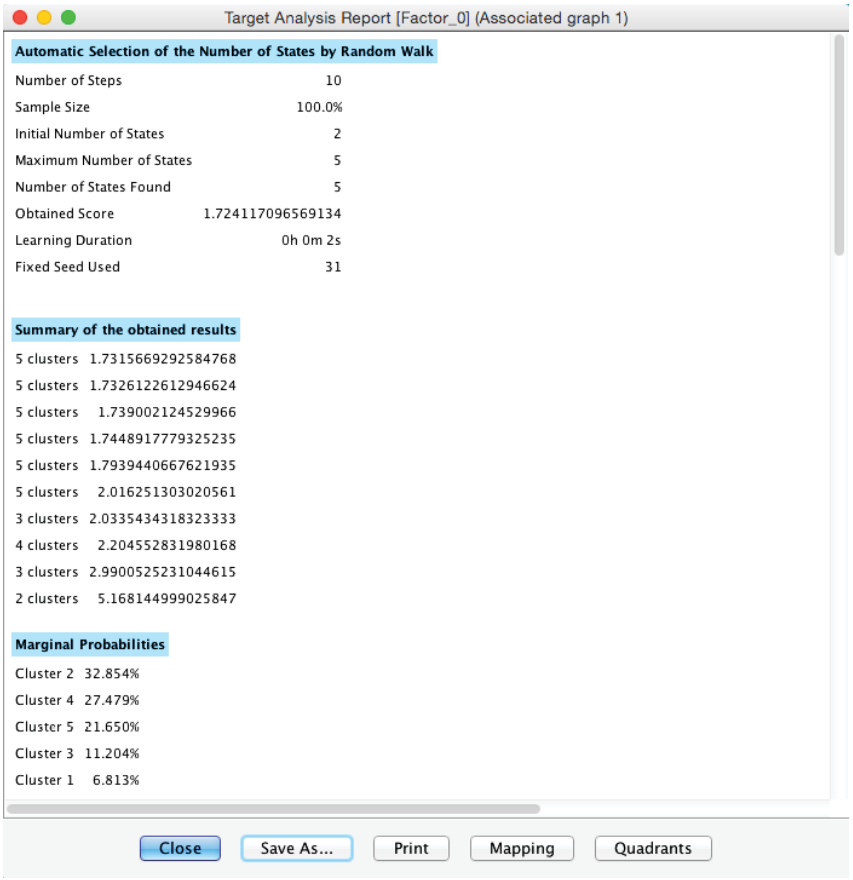
In the `Data Clustering` dialog box, we set the options as shown below. Among the settings, we need to point out that we leave the number of states of the to-be-induced factor open; we only set a range, e.g., 2-5. This means we let BayesiaLab determine the optimal number of states for representing the joint probability distribution.



Upon completing the clustering process, we obtain a graph with newly induced [Factor_0] being connected to all its associated manifest variables.



Furthermore, BayesiaLab produces a window that contains a comprehensive Clustering Report.



Given its size, we show it as two separate tables below.

Automatic Selection of the Number of States by Random Walk	
Number of Steps	10
Sample Size	100.00%
Initial Number of States	2
Maximum Number of States	5
Number of States Found	5
Obtained Score	1.724117097
Learning Duration	0h 0m 2s
Fixed Seed Used	31

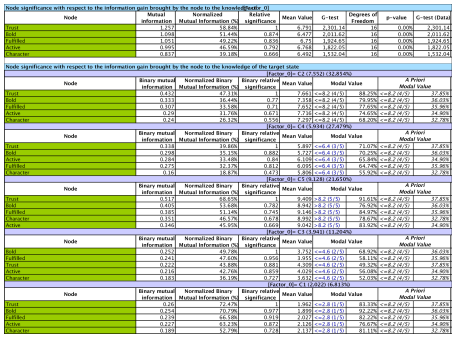
Summary of the obtained results	
5 clusters	1.731566929
5 clusters	1.732612261
5 clusters	1.739002125
5 clusters	1.744891778
5 clusters	1.793944067
5 clusters	2.016251303
3 clusters	2.033543432
4 clusters	2.204552832
3 clusters	2.990052523
2 clusters	5.168144999

Marginal Probabilities	
Cluster 2	32.85%
Cluster 4	27.48%
Cluster 5	21.65%
Cluster 3	11.20%
Cluster 1	6.81%

Clustering Average Purity: 93.656%

Cluster	Purity	Neighborhood
Cluster 1	96.84%	Cluster 3 2.997% Cluster 5 0.111%
Cluster 5	96.51%	Cluster 2 3.444%
Cluster 2	92.97%	Cluster 4 4.699% Cluster 5 2.331%
Cluster 3	92.14%	Cluster 4 5.894% Cluster 1 1.965%
Cluster 4	92.06%	Cluster 2 5.360% Cluster 3 2.436%

Performance Indices
Contingency Table Fit 84.882%
Deviance 1,046.581
Hypercube Cells Per State 530.512



Relative Significance

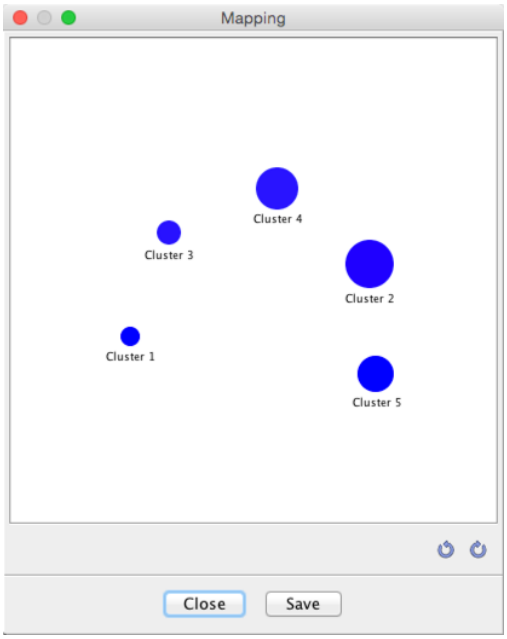
In the second part of the report, the variables are sorted by Relative Significance with respect to the Target Node, which is [Factor_0].

$$RS_i = \frac{I(M_i, F)}{\max_j I(M_j, F)}$$

where M_i represents the i^{th} manifest variable, and F represents the factor variable. The function $I(\cdot)$ computes the Mutual Information.

Mapping

From the window that contains the report, we can also produce a Mapping of the Clusters.

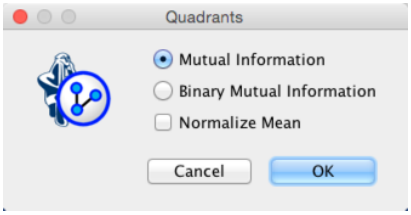


This graph displays three properties of the identified Cluster States (Cluster 1-Cluster 5) within the new Factor node, [Factor_0]:

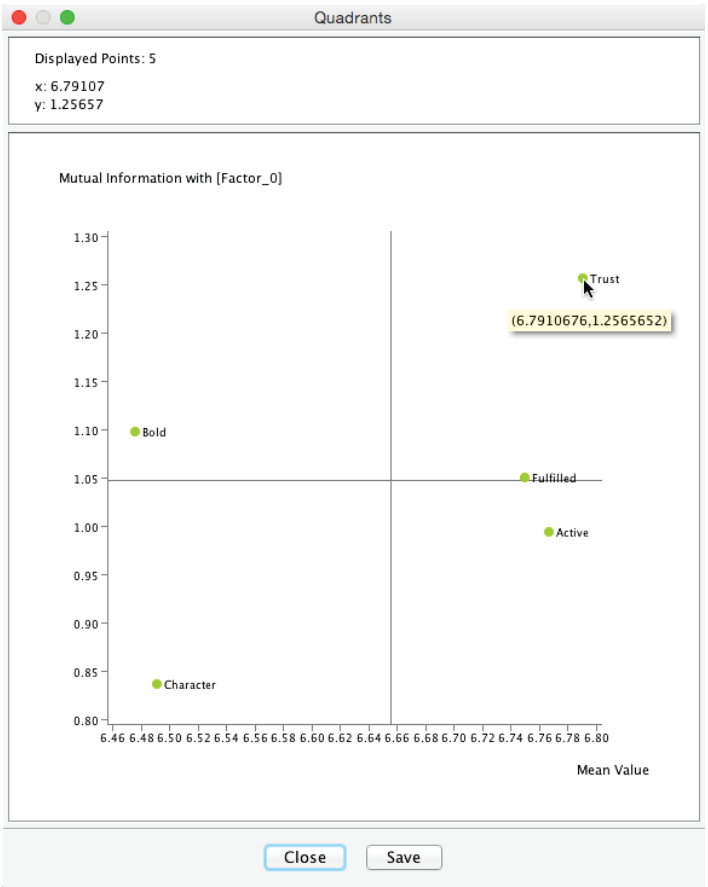
- The saturation of the blue represents the purity of the Cluster States: the higher the purity, the higher the saturation of the color. Here, all purities are in the 90%+ range, which is why they are all deep blue.
- The sizes represent the respective marginal probabilities of the Cluster States. We will see this distribution again once we open the Monitor of the new factor node.
- The distance between any two clusters is proportional to the neighborhood of the clusters.

Quadrants

Clicking the `Quadrants` button in the report window brings up the options for graphically displaying the node's relative importance with regard to the induced [Factor_0].



For our example, we select Mutual Information. Furthermore, we do not need to normalize the means as all values of the Manifest nodes in [Factor_0] are recorded on the same scale.



This Quadrant Plot highlights two measures that are relevant for interpretation:

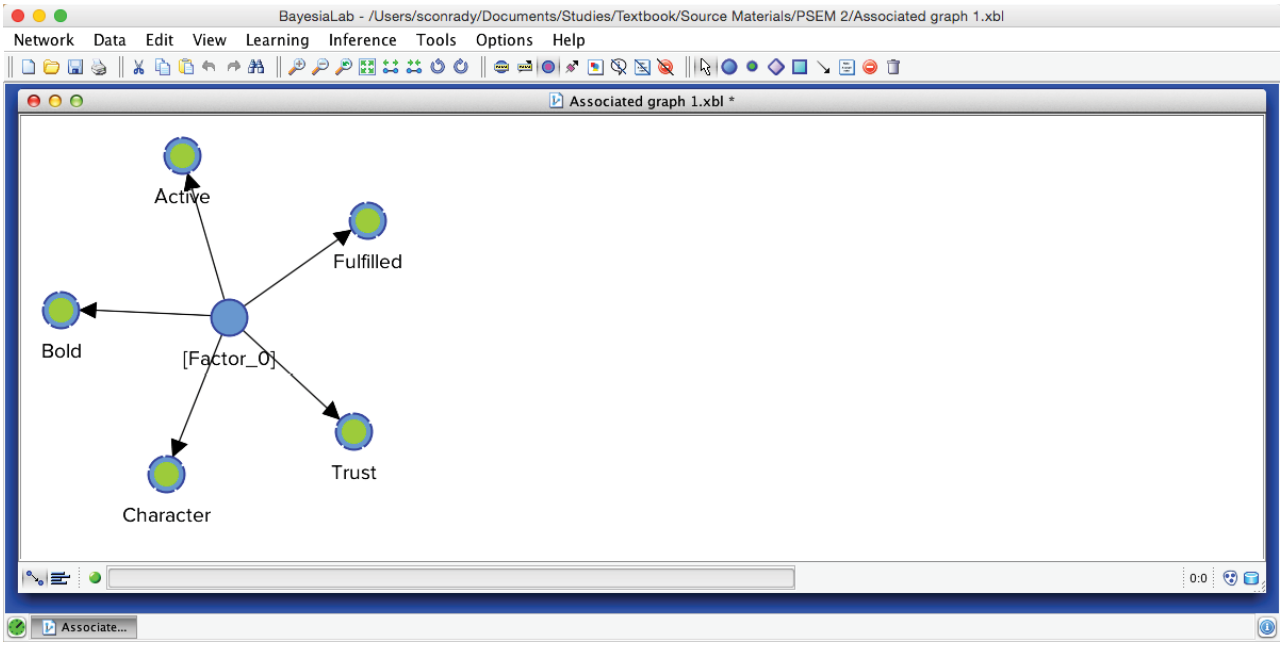
- Mutual Information on the y-axis, i.e., the importance of each Manifest variable with regard to [Factor_0].

- The mean value of each Manifest variable on the x-axis.

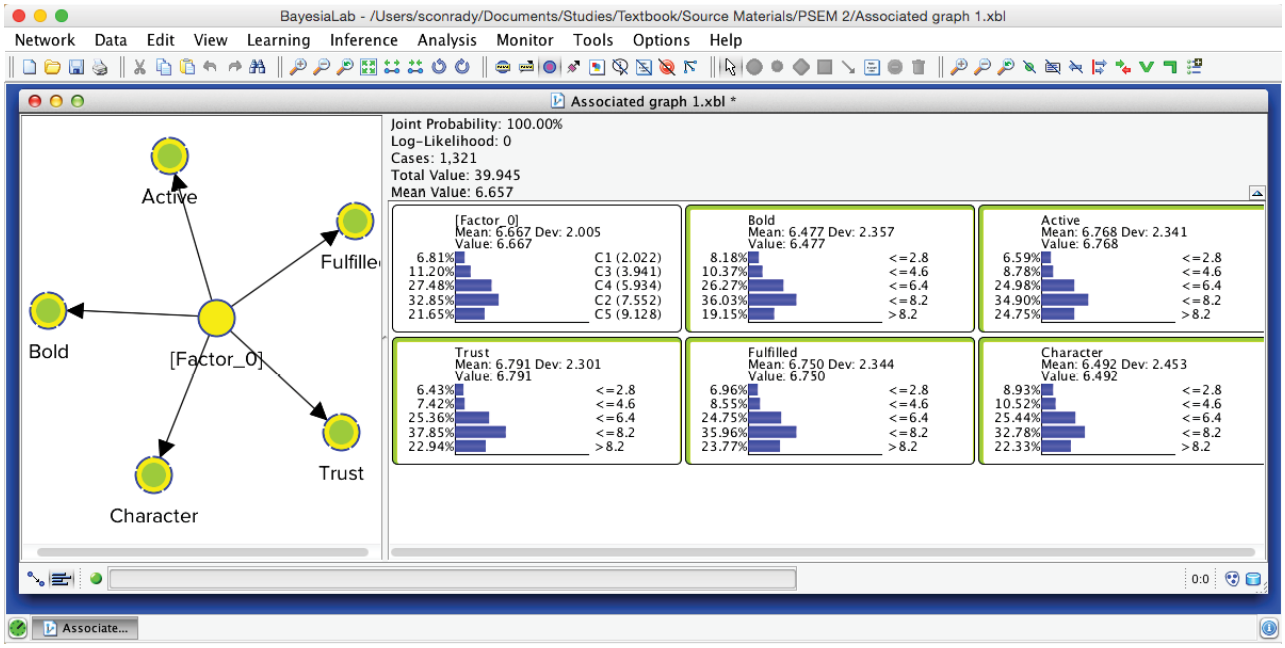
This plot shows that the most important variable is *Trust* with $I(Trust, Factor_0) = 1.26$. It is also the variable with the highest expected satisfaction level, i.e., $E(Trust) = 6.79$.

When hovering with the cursor over the plot, the upper panel of the `Quadrant Plot` window returns the exact coordinates of the respective point, i.e., Mutual Information and Mean Value in this example.

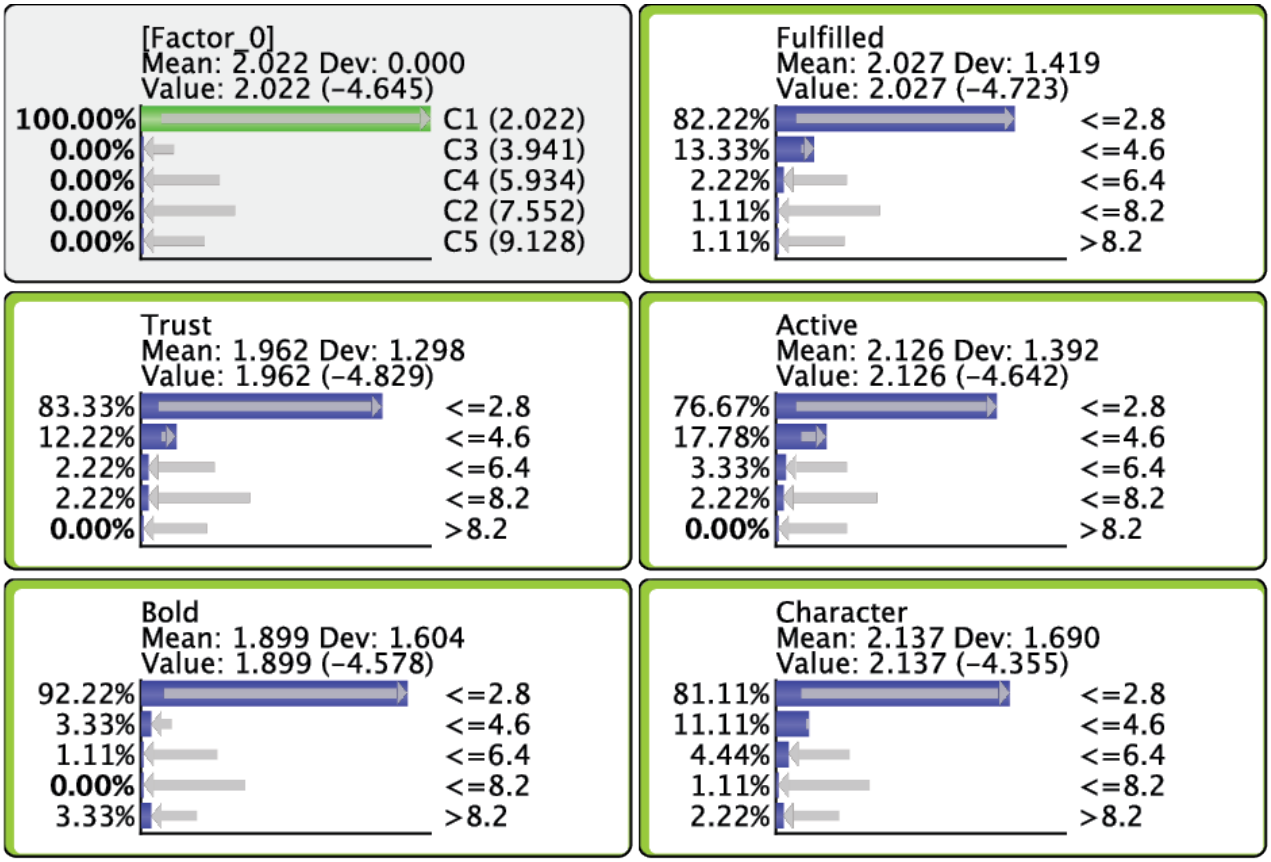
We return to the Graph Panel after closing the Quadrant Plot and the report window. It shows the newly induced [Factor_0] directly connected to all its associated Manifest variables. Applying the Automatic Layout `P` produces a suitable view of the network produced by the Data Clustering process.



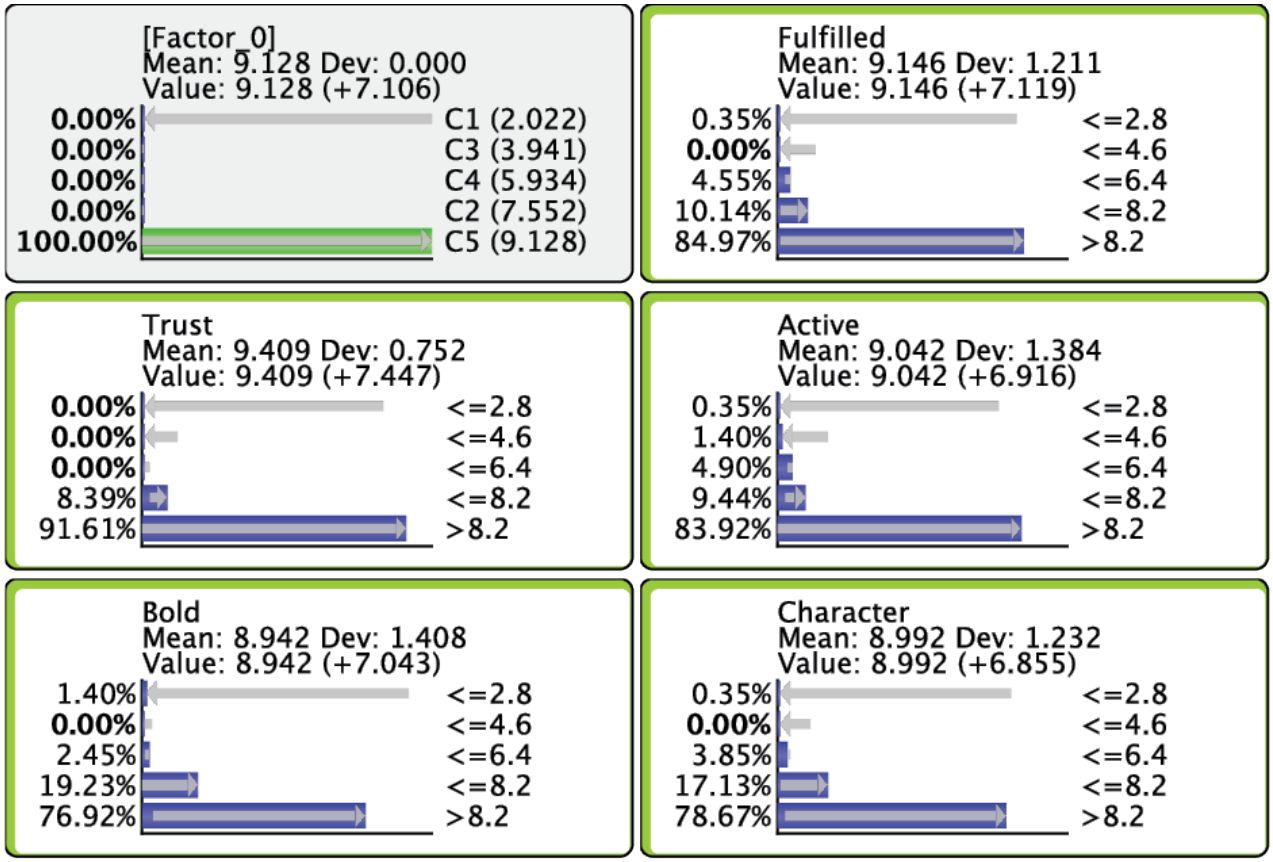
After switching to the Validation Mode `F5`, we open the Monitors for all nodes. We can see five Cluster States for [Factor_0], labeled C1 through C5, as well as their marginal distribution. This distribution was previously represented as the "bubble size" of Clusters 1-5.



In the Monitor of *Factor_0*, we see that the name of each Cluster State carries a value shown in parentheses, e.g., C1 (2.022). This value is the weighted average of the associated Manifest variables given the state C1, where the weight of each variable is its Relative Significance with respect to *Factor_0*. That means that given the state C1 of *Factor_0*, the weighted mean value of *Trust*, *Bold*, *Fulfilled*, *Active*, and *Character* is 2.022. This becomes more apparent when we actually set *Factor_0* to state C1.

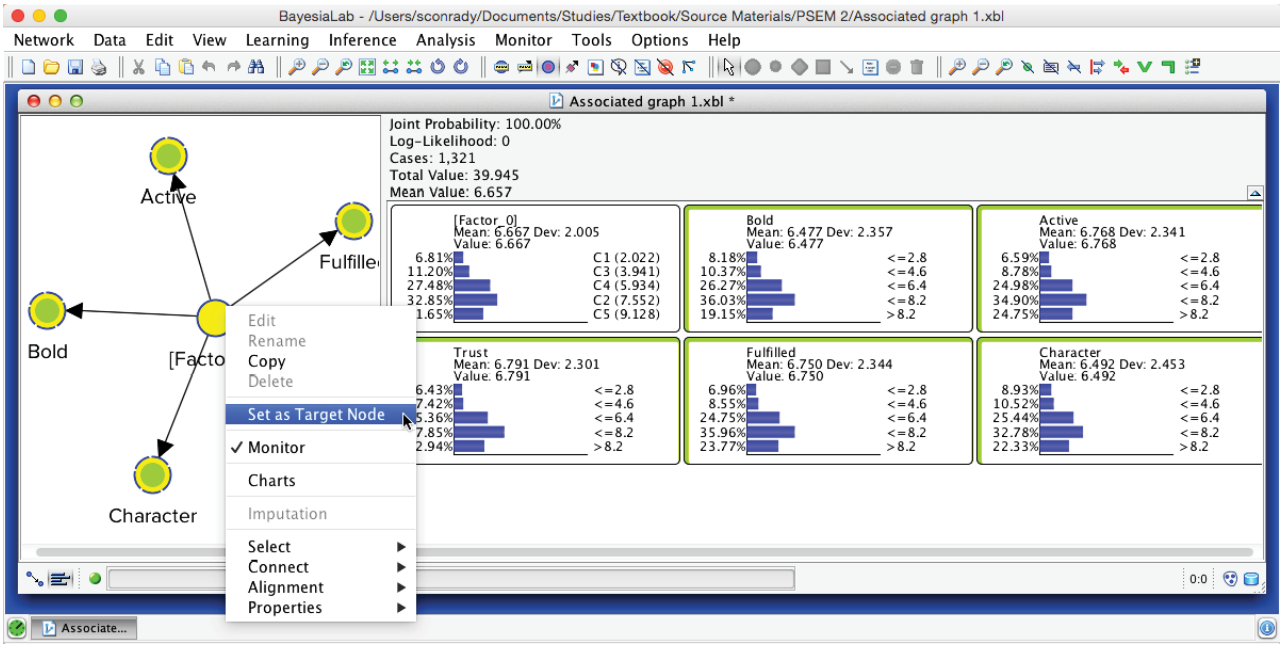


Given that all the associated Manifest variables share the same satisfaction level scale, the values of the created states can also be interpreted as satisfaction levels. Cluster State C1 summarizes the "low" ratings across the Manifest nodes. Conversely, C5 represents mostly the "high" ratings; the other states cover everything else in between.

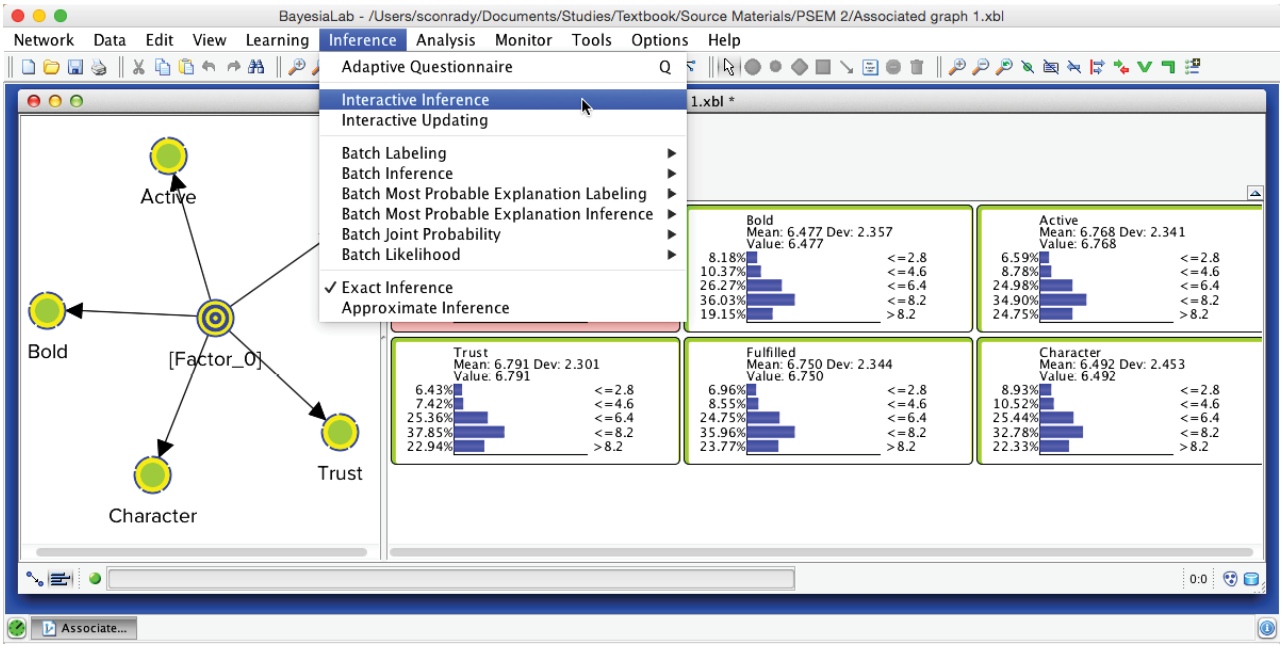


It is important to understand that each underlying record was assigned a specific state of [Factor_0]. In other words, the hidden variable is no longer hidden. It has been added to the dataset and imputed for all respondents. The imputation is done via Maximum Likelihood: given the satisfaction levels observed for each of the 5 Manifest variables, the state with the highest posterior probability is assigned to the respondent.

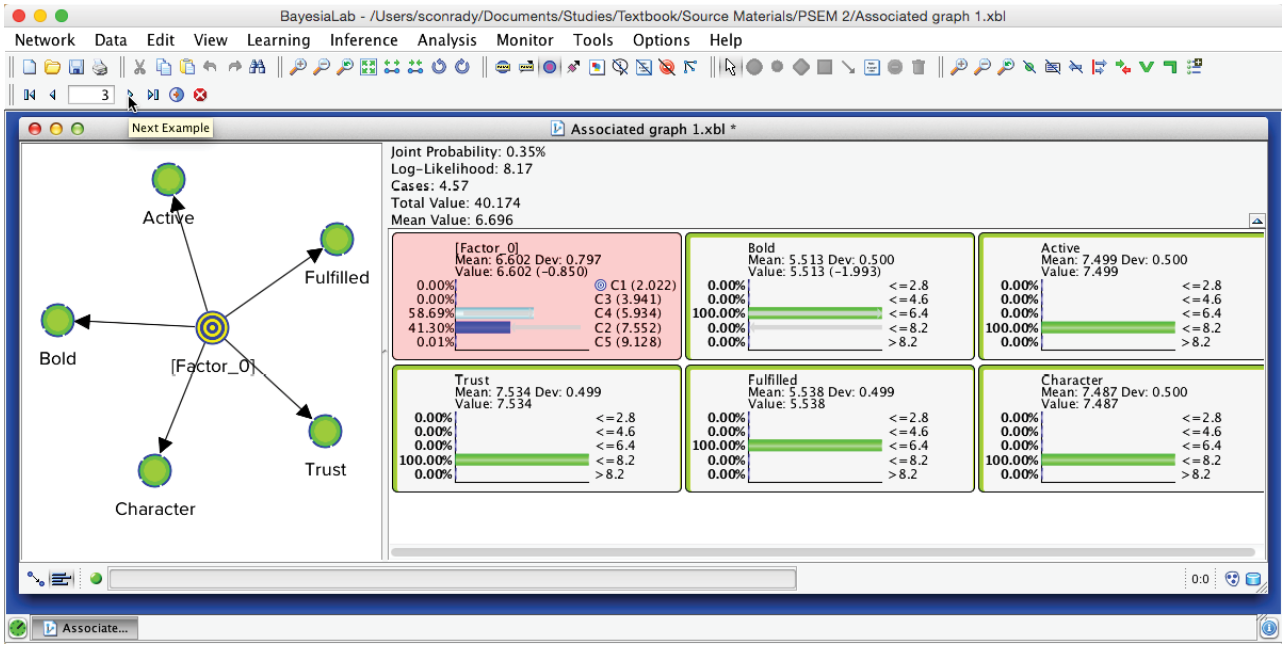
We can easily verify this by scrolling through each record in the dataset. To do so, we first set [Factor_0] as Target Node by right-clicking on it and selecting `Set as Target Node` from the Contextual Menu. Note that the Monitor corresponding to the Target Node turns red.



Then, we select `Menu > Inference > Interactive Inference`.

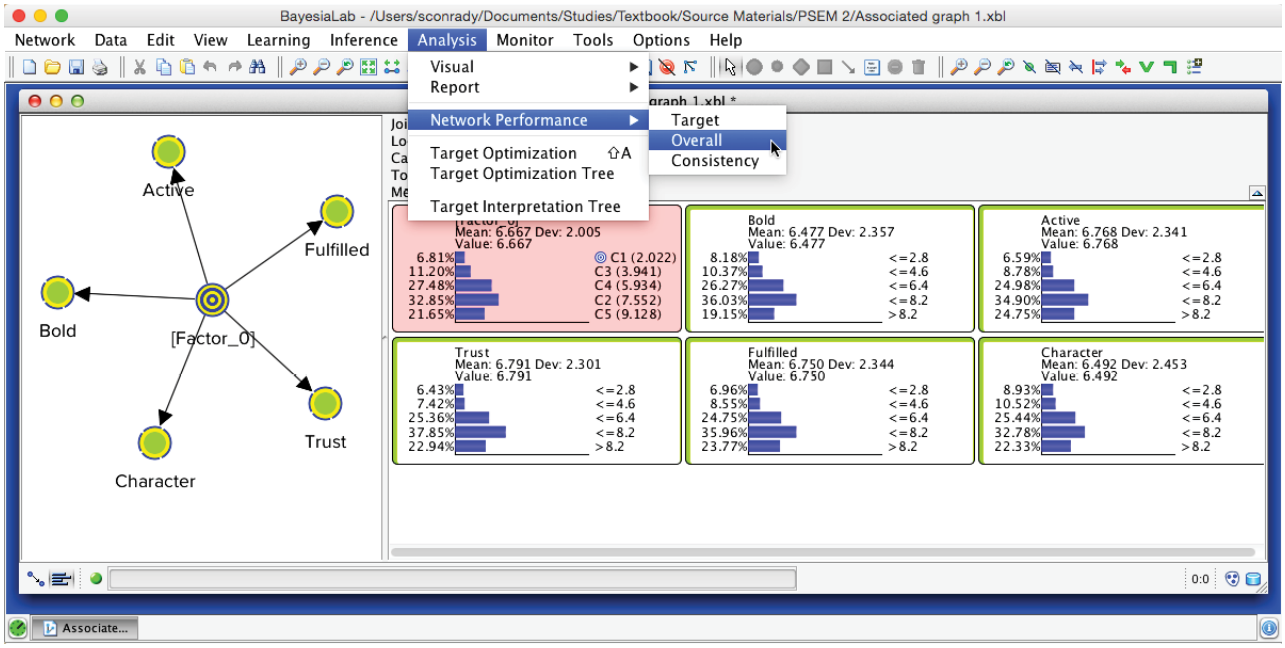


Using the Record Selector in the extended toolbar, we can now scroll through each record in the associated dataset. The Monitors of the Manifest nodes show the actual survey observations, while the Monitor of [Factor_0] shows the posterior probability distribution of the states given these observations. The state highlighted in light blue marks the modal value, i.e., the "winning" state, which is the imputed state now recorded in the dataset. Clicking the Stop icon ⊗ closes this function.

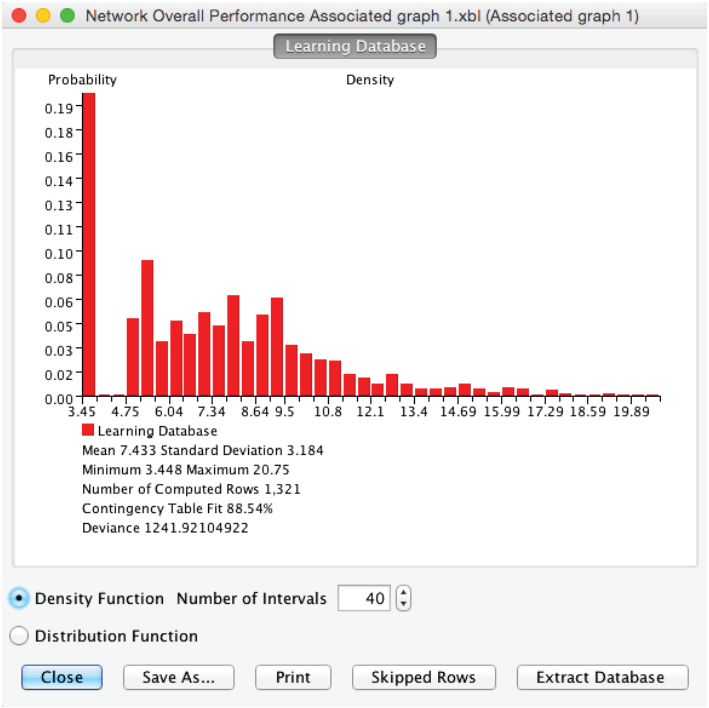


Network Performance Analysis

While the performance indices shown in the Data Clustering Report have already included some measures of fit, we can further study this point by starting a more formal performance analysis: `Menu > Analysis > Network Performance > Overall`.



The resulting report provides measures of how well this network represents the underlying dataset.



Contingency Table Fit

Of particular interest is BayesiaLab's Contingency Table Fit (CTF), which measures the quality of the Joint Probability Distribution (JPD) representation. It is defined as:

$$CTF(B) = \frac{\overline{ll}(B_u) - \overline{ll}(B)}{\overline{ll}(B_f) - \overline{ll}(B)}$$

where:

$\overline{ll}(B)$ is the mean of the log-likelihood of the data given the network currently under study, and $\overline{ll}(B_u)$ is the mean of the log-likelihood of the data given the fully unconnected network, i.e., the "worst-case scenario," and

$\overline{ll}(B_f)$ is the mean of the log-likelihood of the data given the fully connected network, i.e., the "best-case scenario." The fully connected network is the complete graph in which all nodes have direct links to all other nodes. Therefore, it is the exact representation of the chain rule without any conditional independence assumptions in the representation of the joint probability distribution.

Accordingly, we can interpret the following key values of the CTF:

- CTF is equal to 0 if the network represents a joint probability distribution no different from the one produced by the fully unconnected network, in which all the variables are marginally independent.
- CTF is equal to 100 if the network represents the joint probability distribution of the data without any approximation, i.e., it has the same log-likelihood as the fully connected network.

The main benefit of employing CTF as a quality measure is that it has normalized values ranging between 0% and 100%.

> This measure can become negative if the parameters of the model are not estimated from the currently associated dataset.

CTF in Practice

It must be emphasized that CTF measures only the quality of the network in terms of its data fit. As such, it represents the second term in the definition of the MDL Score:

$$MDL(B, D) = \alpha \times DL(B) + \boxed{DL(D \mid B)}$$

Even though this says the higher the CTF, the better the representation of the JPD, we are not aiming for CTF=100%. This would conflict with the objective of finding a compact representation of the JPD.

The Naive structure of the network used for Data Clustering implies that the entire JPD representation relies on the Factor node. Removing this node would produce a fully unconnected network with a CTF=0%. Therefore, BayesiaLab excludes–but does not remove–the Factor node when computing the CTF. This allows measuring the quality of the JPD representation with the induced clusters only.

It is not easy to recommend a threshold value below which the Factor should be "reworked," as the CTF depends directly on the size of the JPD and the number of states of the Factor. For instance, given a Factor with 4 states and 2 binary Manifest variables, a CTF any lower than 100% would be a poor representation of the JPD, as the JPD only consists of 4 cells. On the other hand, given 10 manifest variables, with 5 states each, and a Factor also consisting of 5 states, a CTF of 50% would be a very compact representation of the JPD. This means that 5 states would represent a JPD of 510 cells with a quality of 50%.

Returning to the context of our PSEM workflow, we have the following 3 conditions:

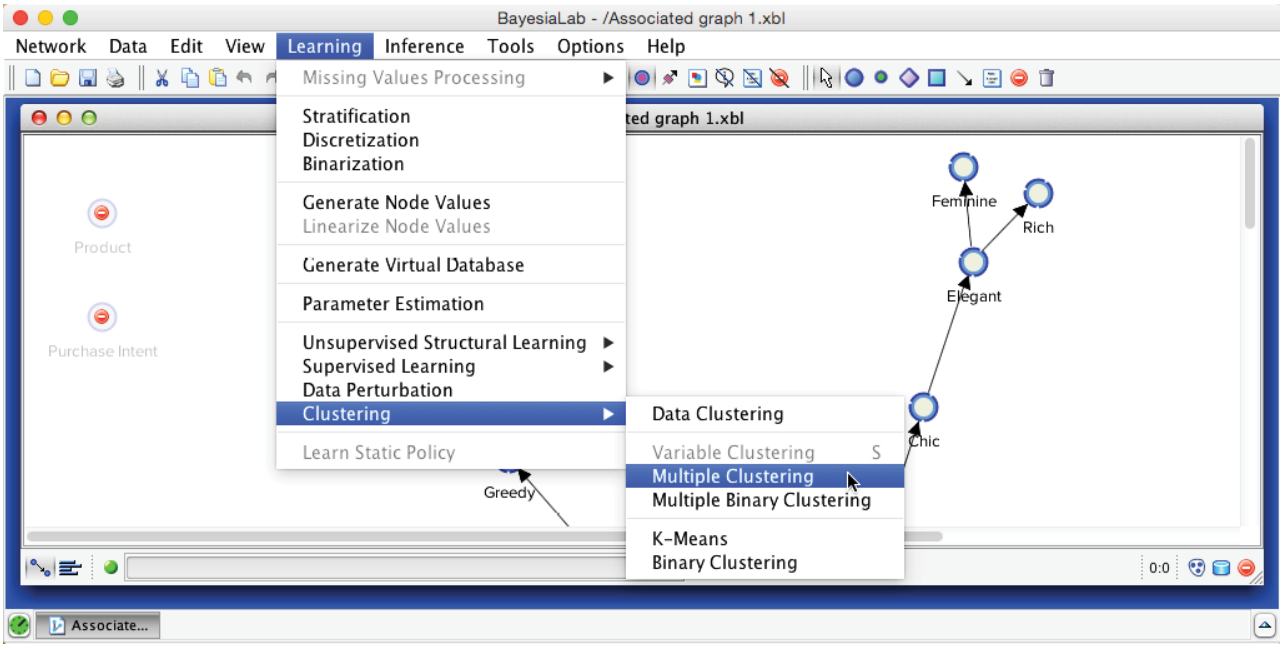
1. A maximum number of 5 variables per cluster of variables;
2. Manifest variables with 5 states;
3. Factors with a maximum of 5 states.

In this situation, we recommend using 70% as an alert threshold. However, this threshold level would have to be reduced if conditions #1 and #2 increased in their values or if condition #3 decreased.

Multiple Clustering

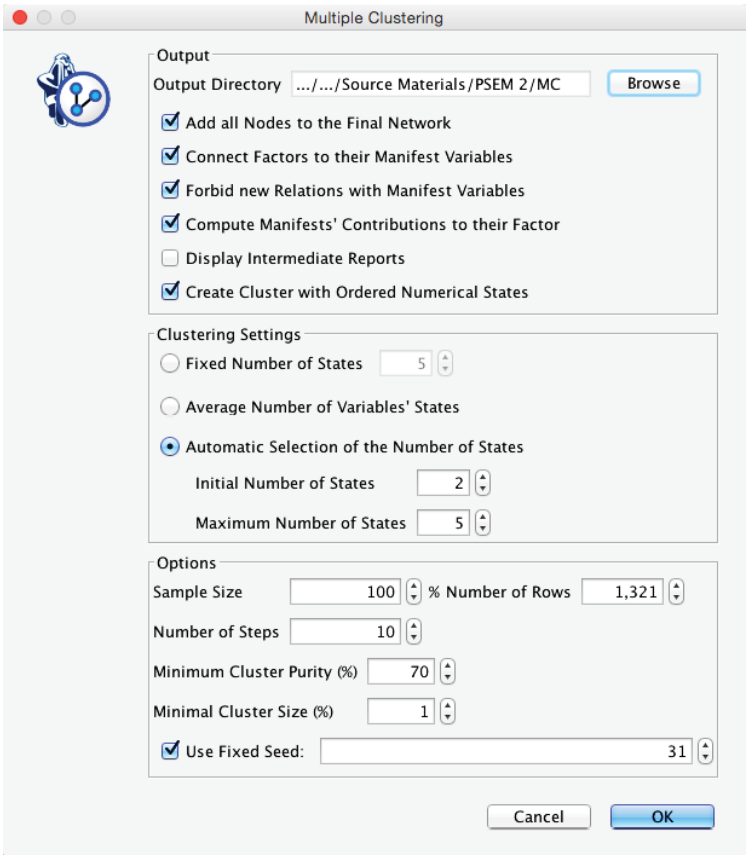
The previous section on Data Clustering dealt exclusively with the induction of [Factor_0]. In our perfume study, however, we have 15 clusters of Manifest variables, for which 15 Factors need to be induced. This means all steps applicable to Data Clustering must be repeated 15 times. BayesiaLab simplifies this task by offering the Multiple Clustering algorithm, which automates all necessary steps for all Factors.

We now return to the original network. On this basis, we can immediately start Multiple Clustering: `Menu > Learning > Clustering > Multiple Clustering`.

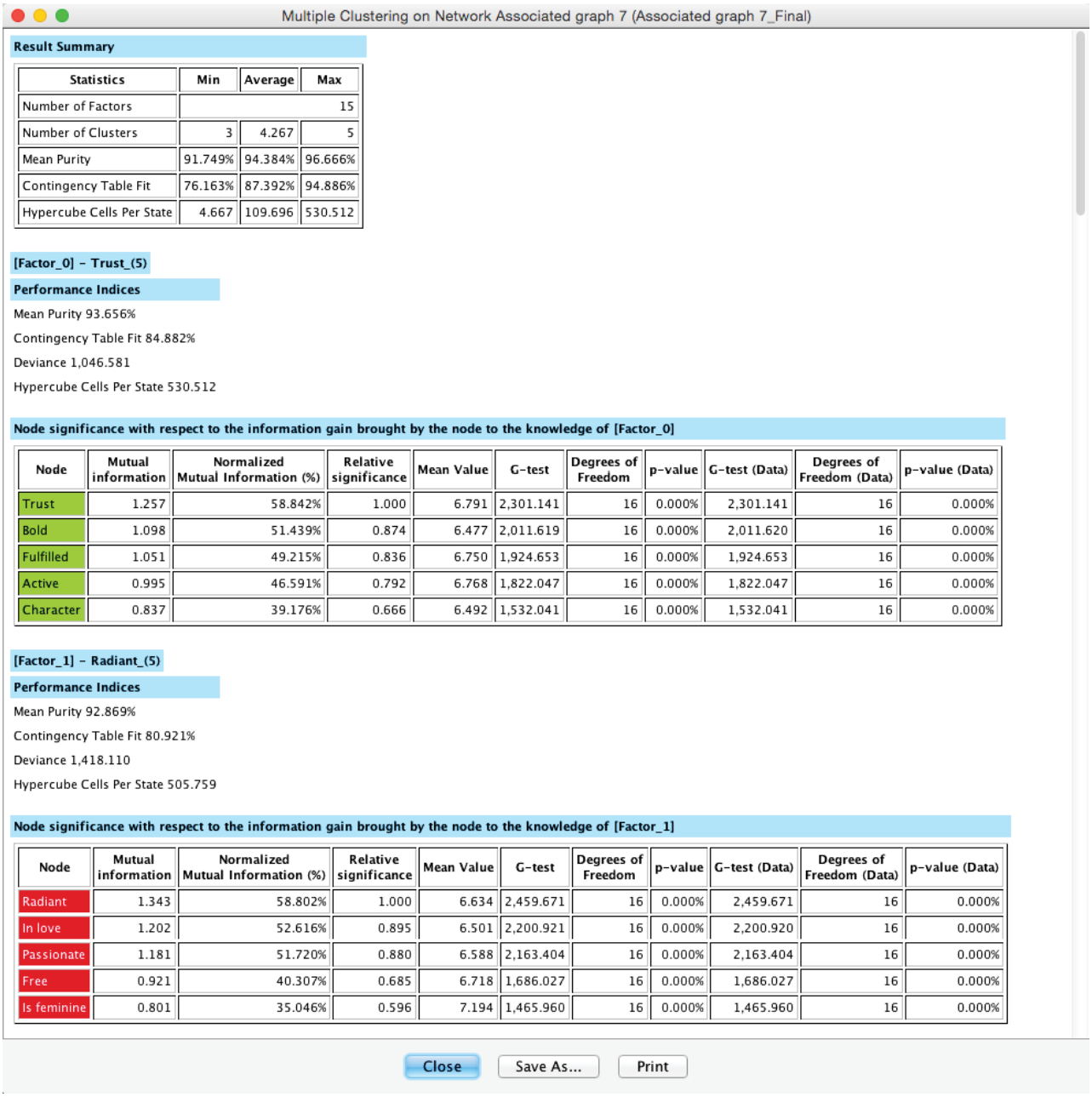


Compared to the dialogue box for Data Clustering, the options for Multiple Clustering are much expanded. Firstly, we need to specify an Output Directory for the to-be-learned networks. This will produce a separate network for each Factor, which we can subsequently examine. Furthermore, we want the new Factors to be connected to their Manifest variables, but we do not wish the Manifest variables to be connected among themselves. We have already learned the relationships between the manifest variables during Step 1. These relationships will ultimately be encoded via the connections between their associated Factors in this Step 3. We consider these new Factor nodes to belong to the second layer of our hierarchical Bayesian network. This also means that, at this point, all structural learning involving the nodes of the first layer, i.e., the Manifest variables, is completed.

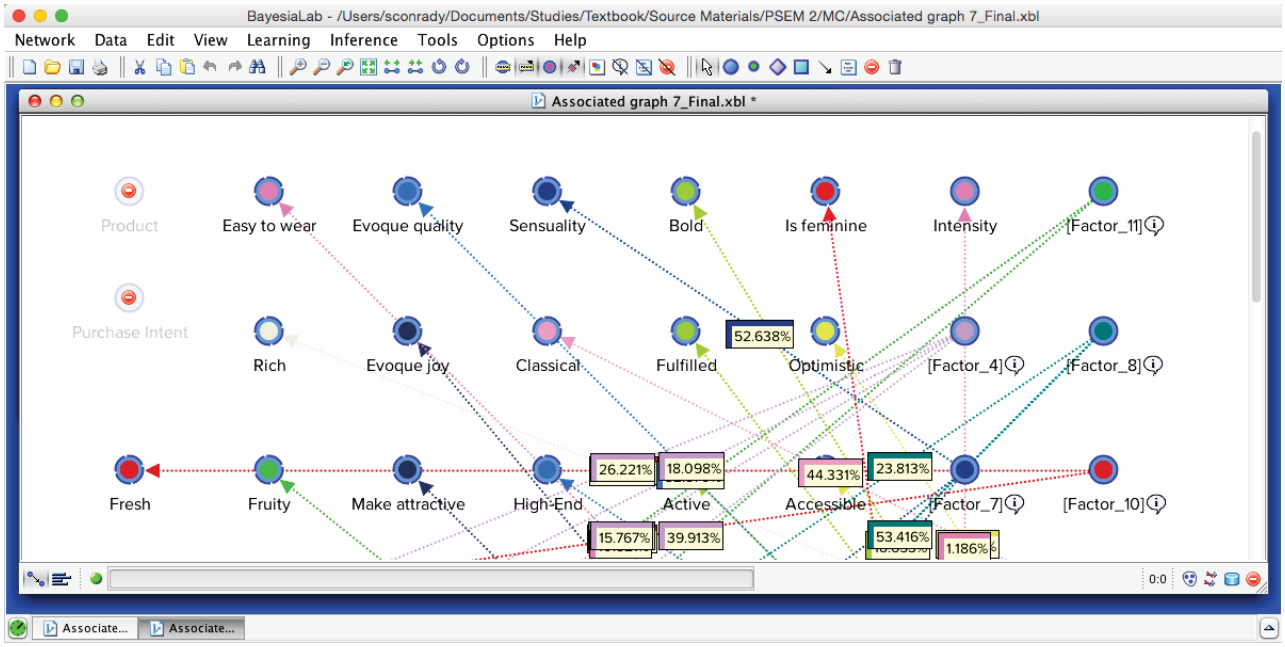
We set the above requirements via `Connect Factors to their Manifest Variables` and `Forbid New Relations` with Manifest Variables. Another helpful setting is `Compute Manifests' Contributions to their Factor`, which helps to identify the dominant nodes within each Factor.



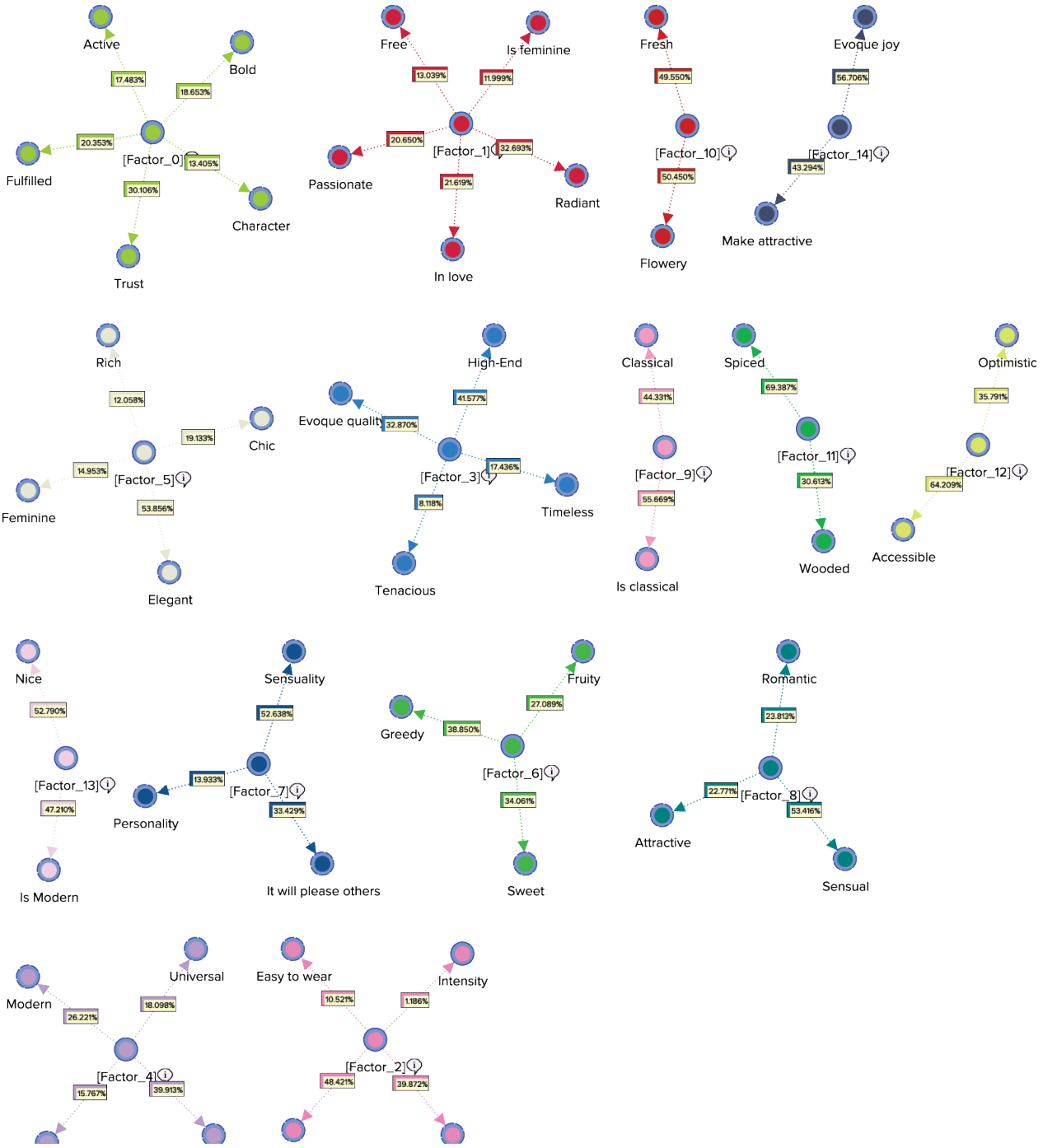
The Multiple Clustering process concludes with a report showing details regarding the generated clustering. Among the many available metrics, we can check the minimum value of the Contingency Table Fit, which is reported as 76.16%. Given the recommendations we provided earlier, this suggests that we did not lose too much information by inducing the latent variables.



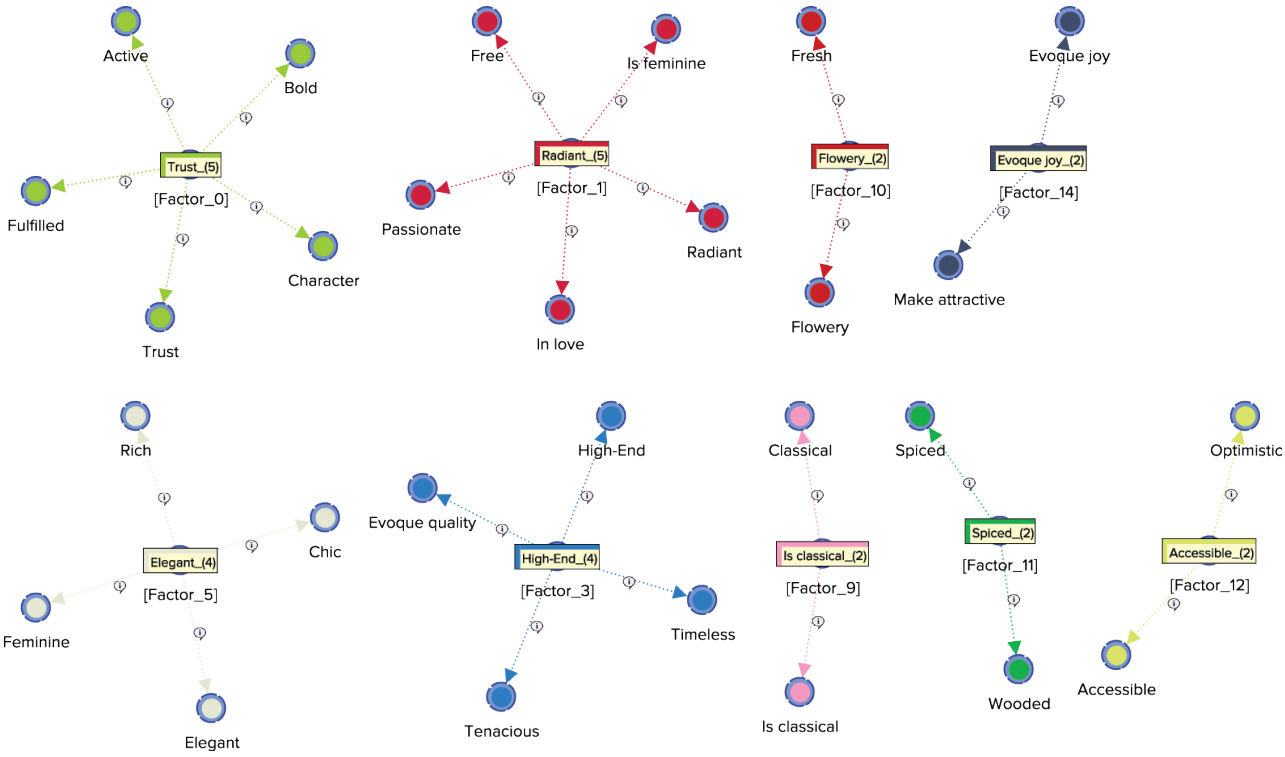
We can save the report or proceed to the new network in the Graph Panel, which has all nodes arranged in a grid-like arrangement: Manifest variables are on the left; the new Factors are stacked up on the right.



Upon applying Automatic Layout (P), we can identify 15 Factors surrounded by their Manifest nodes, arranged almost like a field of flowers.



The Arc Comments, shown by default, display the Contribution of each Manifest variable towards its Factor. Once we turn off the Arc Comments and turn on the Node Comments, we see that the Node Comments contain the name of the "strongest" associated Manifest variable, along with the number of associated Manifest variables in parentheses. The following graph includes a subset of the nodes with their respective Node Comments.



Also, by going into our previously specified output directory, we can see that 15 new sub-networks–in BayesiaLab's XBL format for networks–were generated. Any of these files would allow us to study the properties of the sub-network, as we did for the single Factor, [Factor_0] that was generated by Data Clustering.

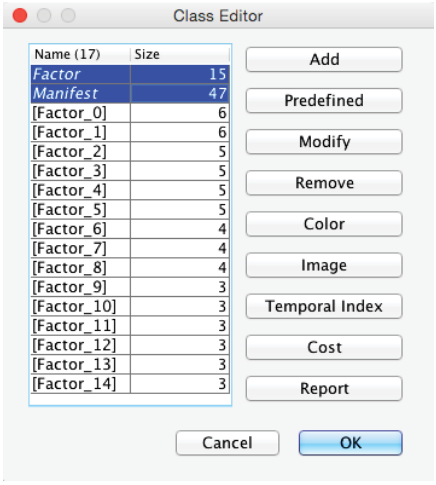
Additionally, one more file was created in this directory, which is highlighted in the screenshot below. The file marked the suffix "_Final" is the network that consists of both the original Manifest variables and the newly created Factors. As such, it is labeled as the "final" network in BayesiaLab parlance. It is also the network that is currently active.

Name	Date Modified	Size	Kind
Associated graph 1_[Factor_0].xbl	Aug 10, 2015, 6:12 PM	11 KB	BayesiaLab.app Document
Associated graph 1_[Factor_1].xbl	Aug 10, 2015, 6:12 PM	11 KB	BayesiaLab.app Document
Associated graph 1_[Factor_2].xbl	Aug 10, 2015, 6:12 PM	8 KB	BayesiaLab.app Document
Associated graph 1_[Factor_3].xbl	Aug 10, 2015, 6:12 PM	10 KB	BayesiaLab.app Document
Associated graph 1_[Factor_4].xbl	Aug 10, 2015, 6:12 PM	10 KB	BayesiaLab.app Document
Associated graph 1_[Factor_5].xbl	Aug 10, 2015, 6:12 PM	9 KB	BayesiaLab.app Document
Associated graph 1_[Factor_6].xbl	Aug 10, 2015, 6:12 PM	8 KB	BayesiaLab.app Document
Associated graph 1_[Factor_7].xbl	Aug 10, 2015, 6:12 PM	8 KB	BayesiaLab.app Document
Associated graph 1_[Factor_8].xbl	Aug 10, 2015, 6:12 PM	8 KB	BayesiaLab.app Document
Associated graph 1_[Factor_9].xbl	Aug 10, 2015, 6:12 PM	6 KB	BayesiaLab.app Document
Associated graph 1_[Factor_10].xbl	Aug 10, 2015, 6:12 PM	6 KB	BayesiaLab.app Document
Associated graph 1_[Factor_11].xbl	Aug 10, 2015, 6:12 PM	6 KB	BayesiaLab.app Document
Associated graph 1_[Factor_12].xbl	Aug 10, 2015, 6:12 PM	6 KB	BayesiaLab.app Document
Associated graph 1_[Factor_13].xbl	Aug 10, 2015, 6:12 PM	5 KB	BayesiaLab.app Document
Associated graph 1_[Factor_14].xbl	Aug 10, 2015, 6:13 PM	6 KB	BayesiaLab.app Document
Associated graph 1_Final.xbl	Aug 10, 2015, 6:13 PM	78 KB	BayesiaLab.app Document

In this context, BayesiaLab also created two new Classes:

- Manifest, which contains all the manifest variables;
- Factor, which contains all the latent variables.

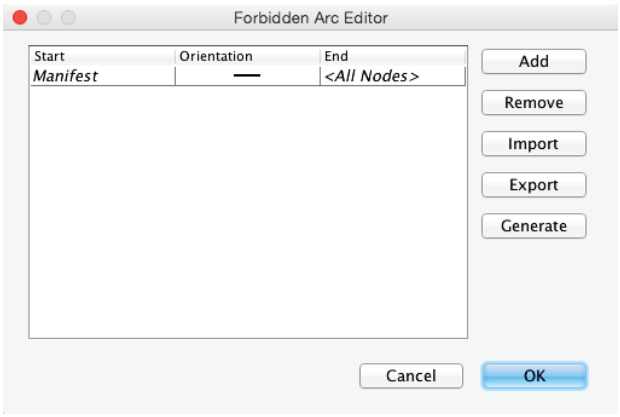
Opening the Class Editor confirms their presence (highlighted below).



Step 4: Completing the Probabilistic Structural Equation Model

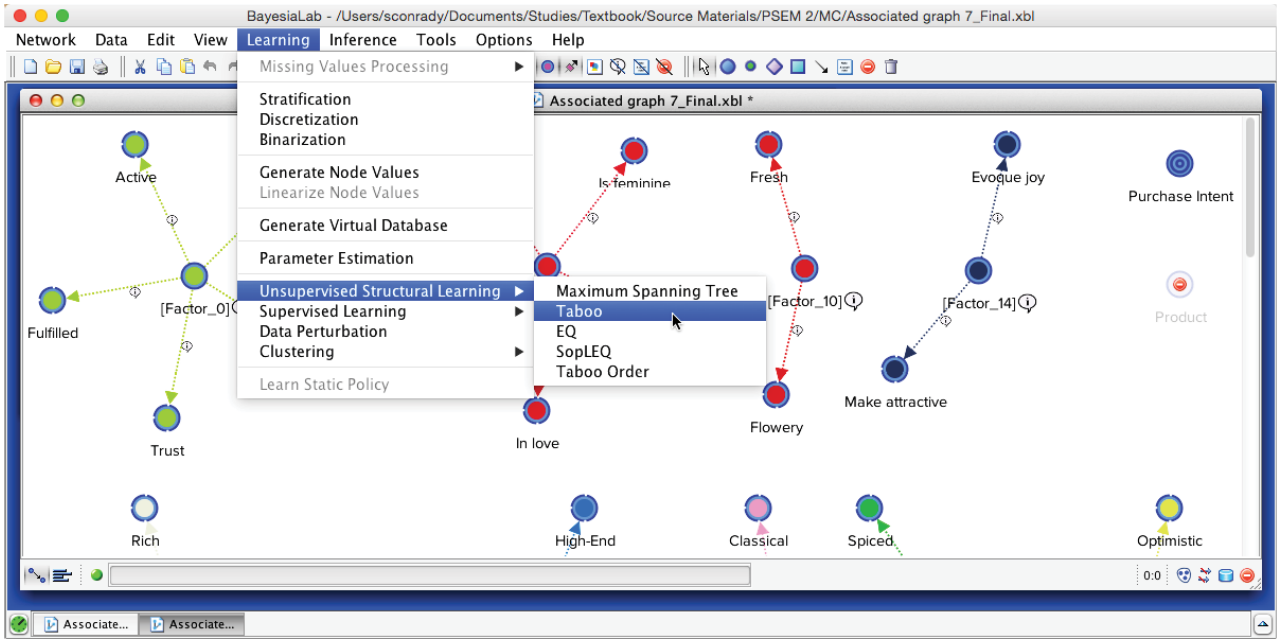
Based on the "final" network, we can proceed to the next step in our network-building process. We now introduce *Purchase Intent*, which has been excluded up to this point. Clicking this node while holding `X` renders it "un-excluded." This makes *Purchase Intent* available for learning. Additionally, we designate *Purchase Intent* as Target Node by double-clicking the node while holding `T`.

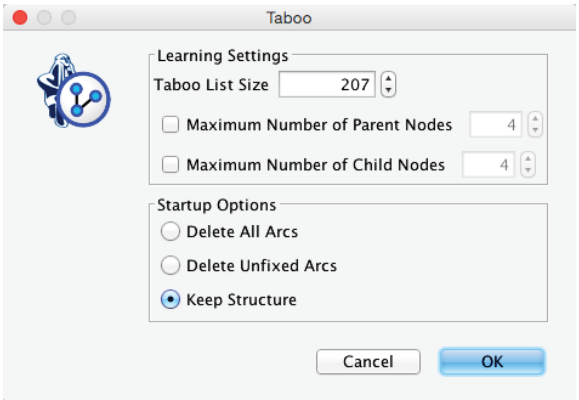
Looking for an SEM-type network structure stipulates that Manifest variables be connected exclusively to the Factors and that all the connections with *Purchase Intent* must go through the factors. We have already imposed this constraint by setting the option Forbid New Relations with Manifest Variables in the `Multiple Clustering` dialog box. This created so-called Forbidden Arcs, which prevent learning algorithms from creating new arcs between the specified nodes. BayesiaLab indicates the presence of Forbidden Arcs with an icon in the lower right-hand corner of the Graph Panel window. Clicking on the icon brings up the Forbidden Arc Editor, which allows us to review the currently set constraints. We see that the nodes belonging to the Class Manifest must not have any links to any other nodes, i.e., both directions are "forbidden."



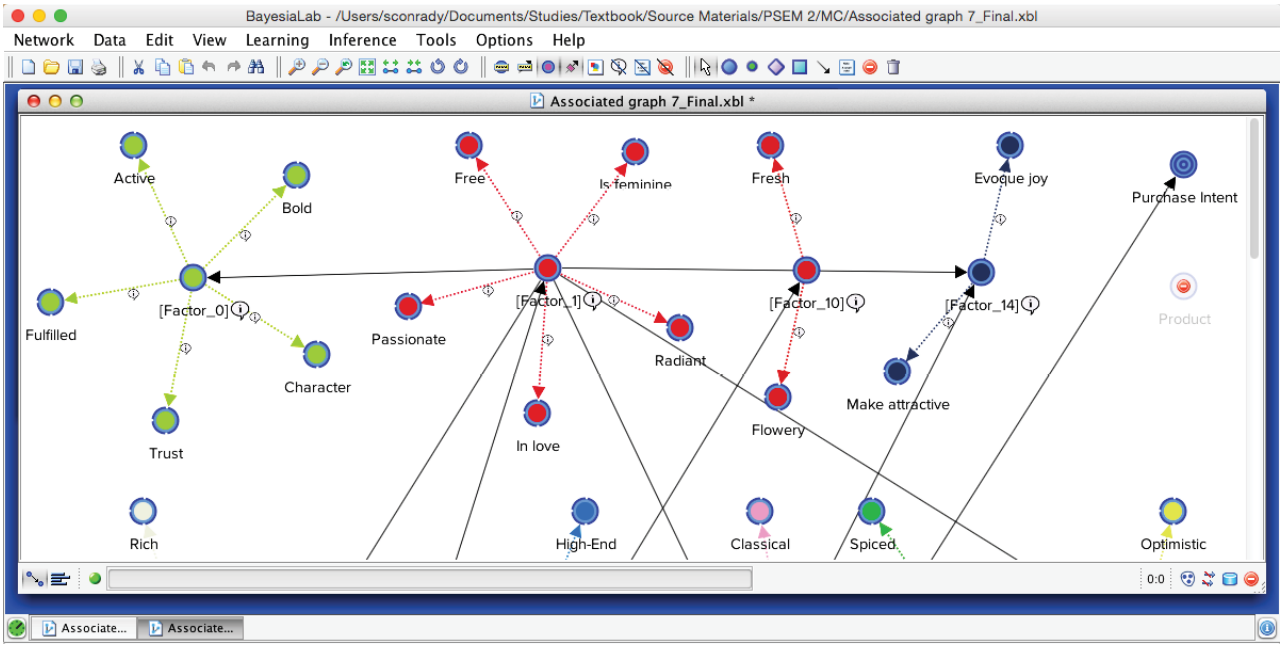
Upon confirming these constraints, we start Unsupervised Learning to generate a network that includes the Factors and the Target Node. In this particular situation, we need to utilize Taboo Learning. It is the only algorithm in BayesiaLab that can learn a new structure on top of an existing network structure and simultaneously guarantee to keep Fixed Arcs unchanged (EQ can also be used for structural learning on top of an existing network, but as it searches in the space of Essential Graphs, there is no guarantee that the Fixed Arcs remain unchanged). This is important as the arcs linking the Factors and their Manifest variables are such Fixed Arcs. To distinguish them visually, Fixed Arcs appear as dotted lines in the network instead of the solid lines of "regular" arcs.

We start Taboo Learning from the main menu by selecting `Menu > Learning > Unsupervised Structural Learning > Taboo` and check the option Keep Network Structure in the `Taboo Learning` dialog box.

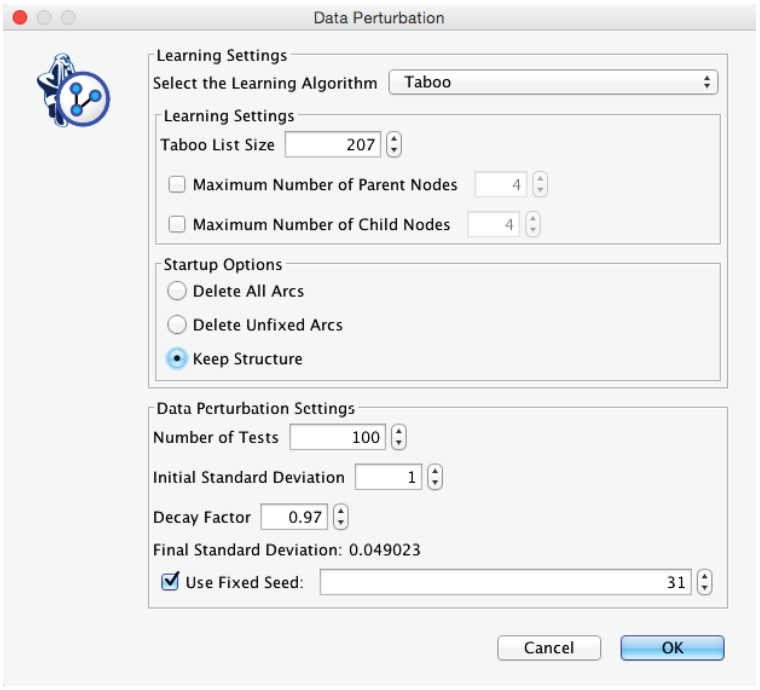




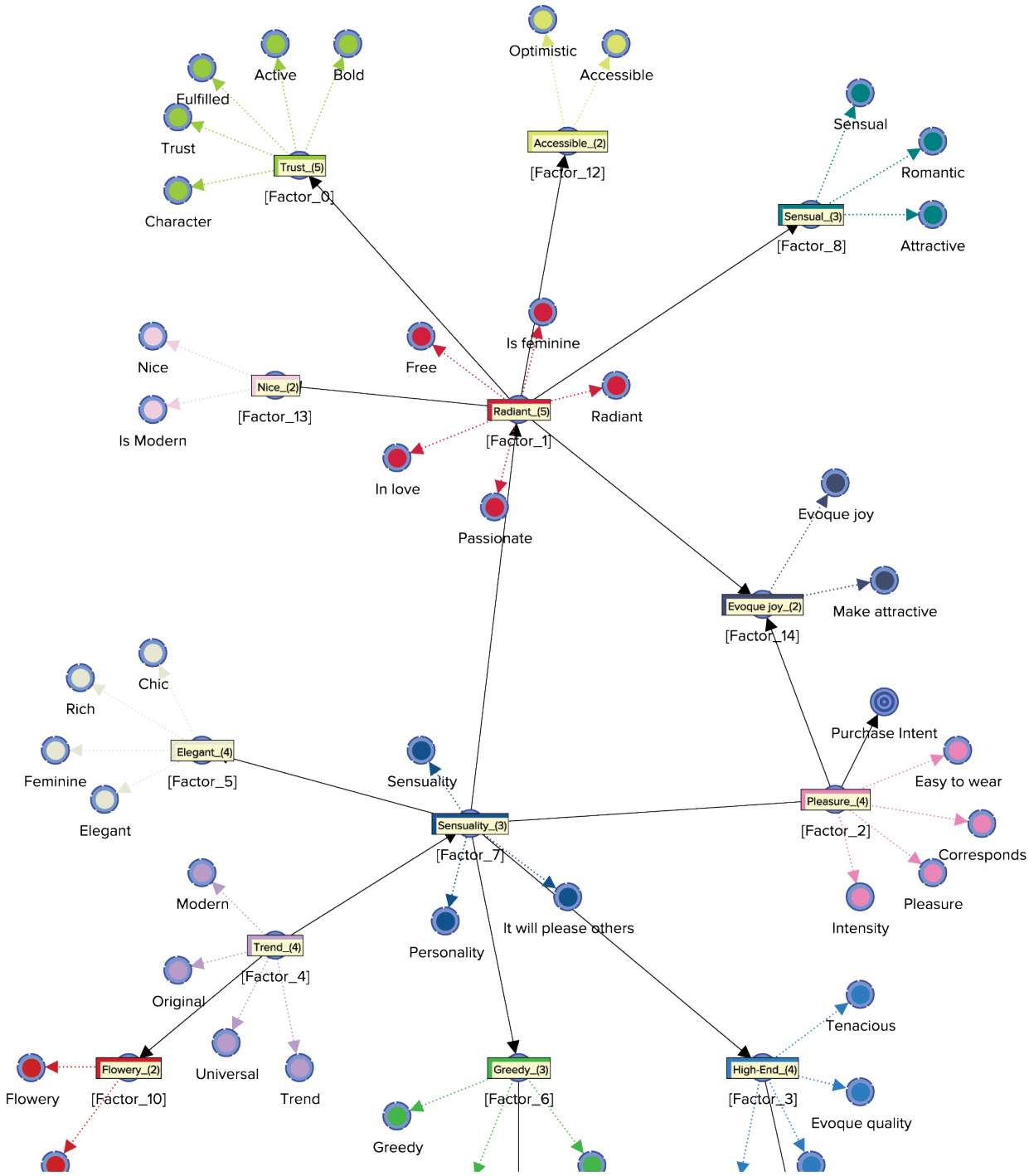
Upon completing the learning process, we obtain the network shown below.



As in Step 1, we also try to improve the quality of this network by using the Data Perturbation algorithm.



As it turns out, this algorithm allowed us to escape from a local optimum and returned a final network with a lower MDL Score. Using Automatic Layout `P` and turning on Node Comments, we can quickly transform this network into a more interpretable format.



Now we see how the manifest variables are "laddering up" to the Factors and how the Factors are related to each other. Most importantly, we can observe where the *Purchase Intent* node was attached to the network during the learning process. The structure conveys that *Purchase Intent* is only connected to [Factor*2], which is labeled with the Node Comment "Pleasure*(4)."

Chapter 9: Missing Values Processing

Introduction

Missing values are encountered in virtually all real-world data collection processes. Missing values can result from non-responses in surveys, poor record-keeping, server outages, attrition in longitudinal surveys, faulty sensors of a measuring device, etc. Despite the intuitive nature of this problem and the fact that almost all quantitative studies are affected by it, applied researchers have given it remarkably little attention in practice. Burton and Altman (2004) state this predicament very forcefully in the context of cancer research: "We are concerned that very few authors have considered the impact of missing covariate data; it seems that missing data is generally either not recognized as an issue or considered a nuisance that it is best hidden."

Given the abundance of "big data" in the field of analytics, missing values processing may not be a particularly fashionable topic. After all, who cares about a few missing data points if there are many more terabytes of observations waiting to be processed? One could be tempted to analyze complete data only and remove all incomplete observations. Regardless of how many more complete observations might be available, this naive approach would almost certainly lead to misleading interpretations or create a false sense of confidence in one's findings.

Koller and Friedman (2009) provide an example of a hypothetical medical trial that evaluates the efficacy of a drug. In this trial, patients can drop out, in which case their results are not recorded. If patients withdraw at random, there is no problem ignoring the corresponding observations. On the other hand, if patients prematurely quit the trial because the drug does not seem to help them, discarding these observations introduces a strong bias in the efficacy evaluation. As this example illustrates, it is crucial to understand the mechanism that produces the missingness, i.e., the conditions under which some values are not observed.

Beyond the naive ad hoc approaches, missing values processing can be a demanding task, both methodologically and computationally. Traditionally, the process of specifying an imputation model has been a scientific modeling effort on its own, and few non-statisticians dared to venture into this specialized field (van Buuren, 2007). With Bayesian networks and BayesiaLab, handling missing values properly now becomes feasible for researchers who might otherwise not attempt to deal with missing values beyond the ad hoc approaches. Responding to Burton and Altman's serious concern, we believe that the presented methods can help missing values processing become an integral part of more research projects in the future. We have already mentioned missing values processing several times in earlier chapters, as it is one of the steps in the Data Import Wizard. However, we have delayed a formal discussion of the topic until now because the recommended missing values processing methods are tightly integrated with BayesiaLab's learning algorithms. Indeed, all of BayesiaLab's core functions for learning and inference are prerequisites for successfully applying missing values processing. With all the building blocks in place, we can now explore this subject in detail.

Types of Missingness

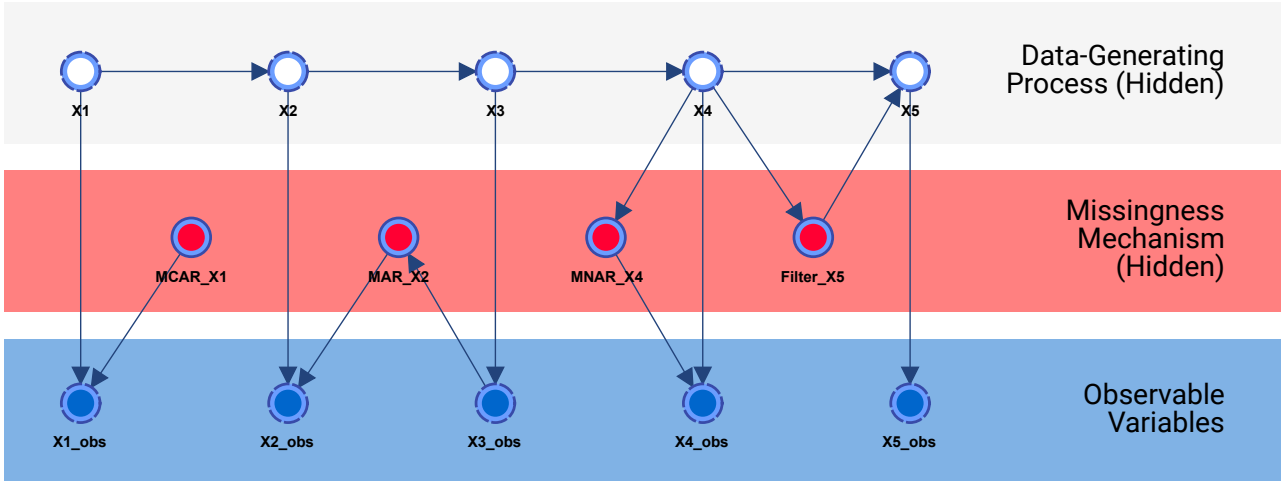
Four principal types of missing values are typically encountered in research:

1. Missing Completely at Random (MCAR)
2. Missing at Random (MAR)
3. Missing Not at Random (MNAR) or Not Missing at Random (NMAR). Both of these equivalent expressions, MNAR and NMAR, appear equally frequently in the literature. We use MNAR throughout this chapter.
4. Filtered Values

We can explain each of these conditions using the causal Bayesian network. It illustrates:

- the data-generating process (DGP);
- the mechanism that causes the missingness;
- the observable variables that contain the missing values.

Reference Network Model



Generating a Test Dataset

We use this reference network to simulate all missingness conditions and generate a test dataset from it for subsequent evaluation:

- Learn about Generating the Test Dataset

With such a test dataset, the problems associated with missingness become very obvious. Given that we have specifically encoded all types of missingness mechanisms in the reference network, the resulting test dataset is a kind of worst-case scenario, which is ideal for testing purposes.

However, before we can apply any missing values processing methods, we need to bring the test dataset into BayesiaLab. While the Data Import Wizard is explained in detail in the BayesiaLab User Guide (see Open Data Source), we quickly summarize the steps in the following sub-topics:

- Importing the Test Dataset into BayesiaLab

Missing Values Processing in BayesiaLab

Once imported into BayesiaLab, we attempt to recover the reference network's original distributions from the test dataset.

In a typical data analysis workflow in BayesiaLab, a researcher encounters Missing Values Processing in Step 3 of the Data Import Wizard, i.e., when importing a dataset. So, we evaluate each available Missing Values Processing method in the context of a prototypical workflow.

- Filter (Listwise/Casewise Deletion)
- Replace By (Mean/Modal Imputation)
- Infer — Static Imputation
- Infer — Dynamic Imputation
- Infer — Structural EM
- Infer — Entropy-Based Imputations
- Approximate Dynamic Imputation

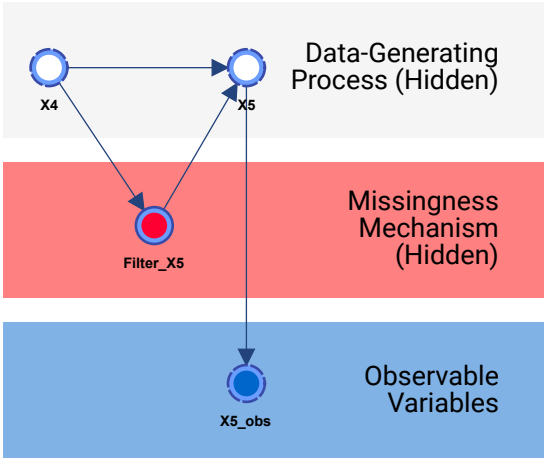
Each of the above methods yields an imputed dataset. Now, we can examine how well the imputed datasets match the distributions from the reference model. We explore the advantages and disadvantages of each method on this basis. Ultimately, this assessment of approaches is meant as a guide for choosing a Missing Values Processing method as a function of what we know about the data-generating process and the missingness mechanism in particular.

> Before we even present results, we need to warn you that some of the to-be-evaluated methods, such as Filter (Listwise/Casewise Deletion) and Replace By (Mean/Modal Imputation), are not recommended for default use. We still include them for two reasons: First, they are almost universally used in statistical analysis, and second, under certain circumstances, they can be safe to use. Regardless of their suitability for research, they can help us understand the challenges of missing values processing.

Filtered Values

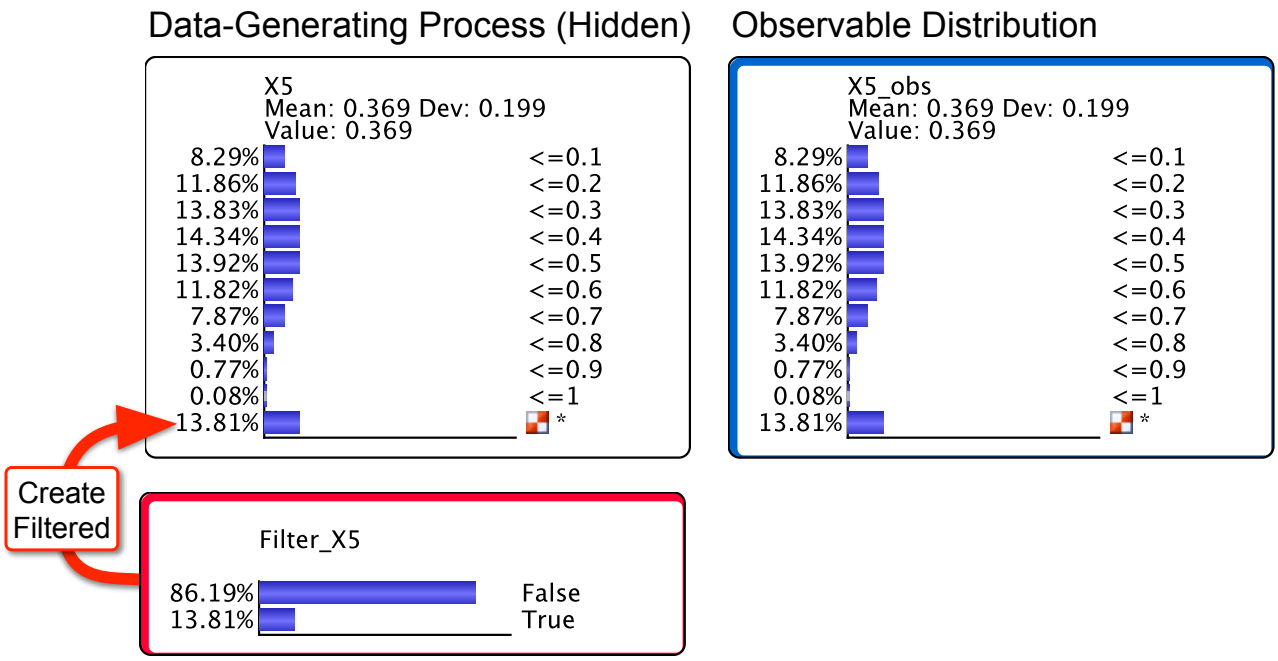
- There is a fourth type of missingness, which is less often mentioned in the literature. In BayesiaLab, we refer to missing data of this kind as Filtered Values (see also Filtered Values in Chapter 5).

- In fact, Filtered Values are technically not missing at all. Instead, Filtered Values are values that do not exist in the first place. Clearly, something nonexistent cannot become missing due to a missingness mechanism.
- For instance, in a hotel guest survey, a question about one's satisfaction with the hotel swimming pool cannot be answered if the hotel property does not have a swimming pool. This question is not applicable. The absence of a swimming pool rating for this hotel would not be a missing value. On the other hand, for a hotel with a swimming pool, the absence of an observation would be a missing value.
- Conceptually, Filtered Values are quite similar to Missing at Random (MAR) values, as Filtered Values usually depend on other variables in the dataset, too, which may or may not be fully observed. However, Filtered Values should never be processed as missing values. In our example, it is certainly not reasonable to impute a value for the swimming pool rating if there is no swimming pool. Instead, a Filtered Value should be considered a special type of observation.
- In BayesiaLab, an additional state, marked with a funnel icon, is added to this type of variable in order to denote Filtered Values (BayesiaLab's learning algorithms implement a kind of local selection for excluding the observations with Filtered Values while estimating the probabilistic relationships).
- The following illustration shows an example of a network including Filtered Values.



- Once again, we must describe the parameters of the subnetwork, including the Filtered Values mechanism and the observable variable:
- *Filter_X5* is a boolean variable with one parent, which specifies that it depends on the hidden variable *X4*. Here, *X5* becomes a Filtered Value if *X4* is greater than 0.7.
- *X5* is a continuous variable with values between 0 and 1. It has two parents, *X4* and the Filtered Values mechanism: `IF Filter_X5 THEN X5=Filtered Value ELSE X5=f(X4)`
- *X5_obs* is a pure clone of *X5*, i.e., *X5* is fully observed: `X5_obs=X5`

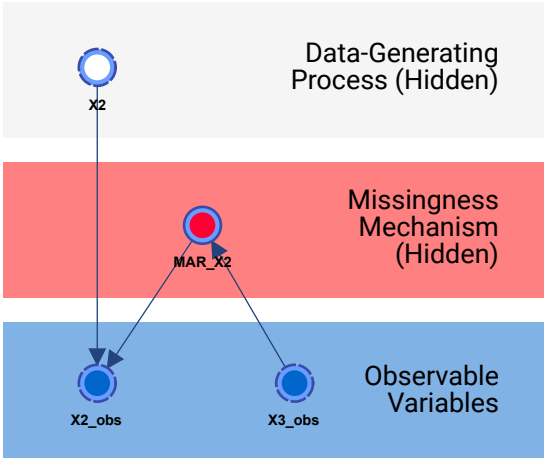
- For the sake of completeness, we present the Monitors of *X5* (left) and *X5_obs* (right):



Missing at Random (MAR)

Secondly, data can be Missing at Random (MAR). Here, the missingness of data depends on observed variables. A brief narrative shall provide some intuition for the MAR condition: in a national survey of small business owners about the business climate, there is a question about the local cost of energy. Chances are that the owner of a business that uses little electricity, e.g., a yoga studio, may not know of the current price of 1 kWh of electric energy and could not answer that question, thus producing a missing value in the questionnaire. On the other hand, the owner of an energy-intensive business, e.g., an electroplating shop, would presumably be keenly aware of the electricity price and able to respond accordingly. In this story, the probability of non-response is presumably inversely proportional to the energy consumption of the business.

In the subnetwork shown below, *X3_obs* is the observed variable that causes the missingness, e.g., the energy consumption in our story. *X2_obs* would be the stated price of energy if known. *X2* would represent the actual price of energy in our narrative. Indeed, from the researcher's point of view, the actual cost of energy in each local market and for each electricity customer is hidden.



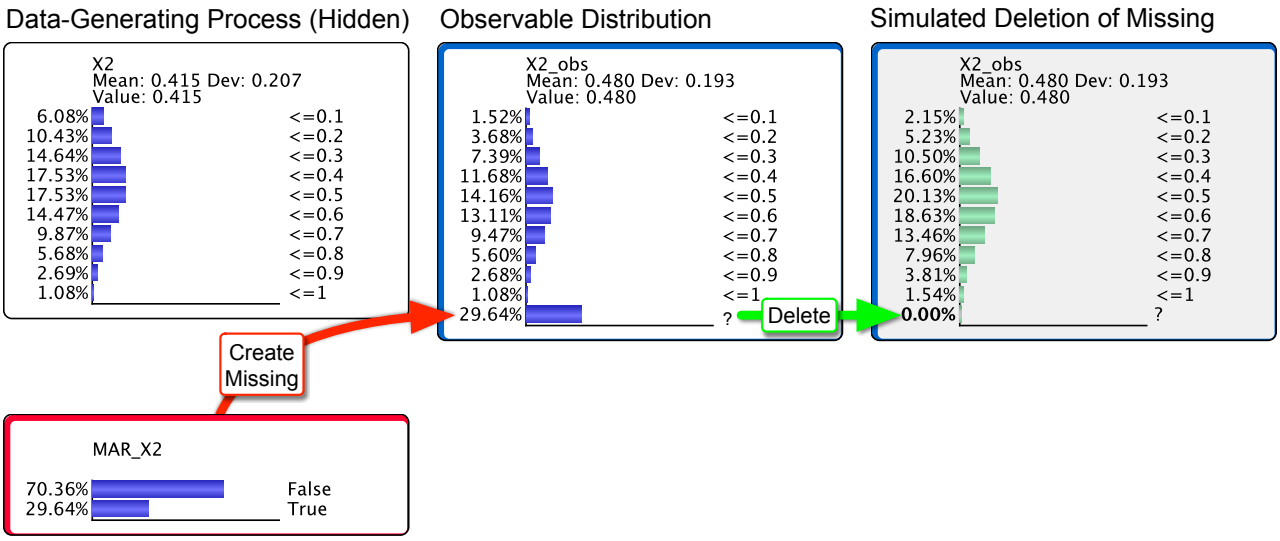
To simulate this network, we need to define its parameters, i.e., the quantitative part of the network structure:

- *X2* is a continuous variable with values between 0 and 1. Here, too, we have arbitrarily defined a Normal distribution for modeling the DGP.
- *MAR_X2* is a boolean variable with one parent, which specifies that the missingness probability depends directly on the fully observed variable *X3_obs*. The exact values are not important here, as we only need to know that the probabilities of missingness are inversely proportional to the values of *X3_obs*:

$$P(MAR_{X2} = true \mid X3_{obs}) \propto \frac{1}{X3_{obs}}$$

- *X2_obs* has two parents, i.e., the data-generating variable *X2* and the missingness mechanism *MAR_X2*. The conditional probability distribution of *X2_obs* can be described by the following deterministic rule: `IF MAR_X2 THEN X2_obs=?  ELSE X2_obs=X2`

Given the fully specified network, we can now simulate the impact of the missingness mechanism on the observable variable *X2_obs*.



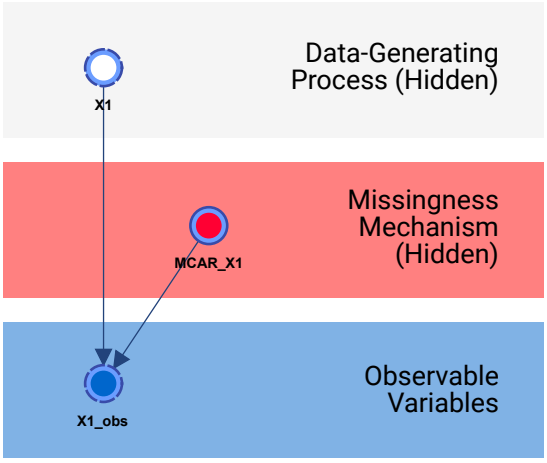
As the above screenshot shows, the mean and standard deviation in the Monitor of *X2_obs* indicates that the distribution of the observed values of *X2* differs significantly from the original distribution, leading to an overestimation of *X2* in this example. We can simulate the deletion of incomplete observations by setting negative evidence on "?" in the Monitor of *X2_obs* (green arrow labeled "Delete"). The simulated distribution of *X2_obs* (right) clearly differs from the one of *X2* (left).

Missing Completely at Random (MCAR)

Missing Completely at Random (MCAR) means that the missingness mechanism is entirely independent of all other variables. In our causal Bayesian network, we encode this independent mechanism with a boolean variable named *MCAR_X1*.

Furthermore, we assume that there is a variable *X1* that represents the original data-generating process. This variable, however, is hidden, so we cannot observe it directly. Instead, we can only observe *X1* via the variable *X1_obs*, which is a "clone" of *X1* but with one additional state, "?", which indicates that the value of *X1* is not observed.

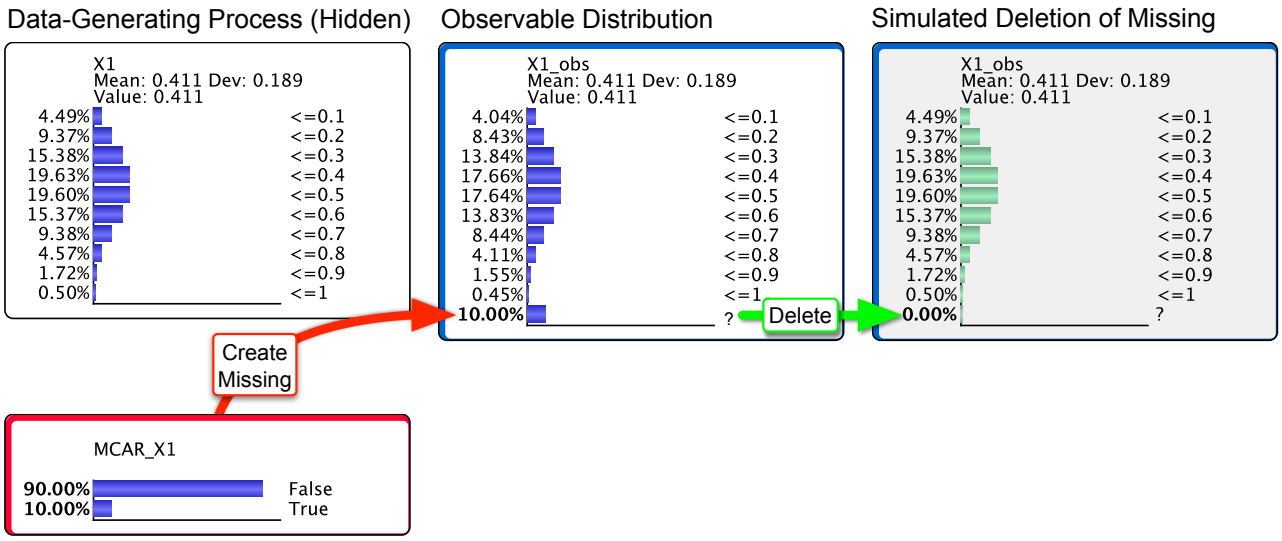
The Bayesian network shown below is a subnetwork of the complete network presented above. The behavior of the three variables we just described is encoded in this subnetwork.



In addition to this qualitative structure, we need to describe the quantitative part, i.e., the parameters of this subnetwork, including the missingness mechanism and the observable variable:

- *X1* is a continuous variable with values between 0 and 1. We have arbitrarily defined a Normal distribution for modeling the DGP.
- *MCAR_X1* is a boolean variable without any parent nodes. This means that *MCAR_X1* is independent of all variables, whether hidden or not. Its probability of being true is 10%.
- *X1_obs* has two parents: the data-generating variable *X1* and the missingness mechanism *MCAR_X1*. The following deterministic rule defines the conditional probability distribution of *X1_obs*: `IF MCAR_X1 THEN X1_obs=?  ELSE X1_obs=X1`

Now that our causal Bayesian network is fully specified, we can evaluate the impact of the missingness mechanism on the observable variable *X1_obs*. Given that we have created a complete model of this small domain, we automatically have perfect knowledge of the distribution of *X1*. Thus, we can directly compare *X1* and *X1_obs* via the Monitors.



We see that *X1* (left) and *X1_obs* (center) have the same mean and the same standard deviation. This suggests that the remaining observations in *X1_obs* (center) are not different from the non-missing cases in *X1* (left). The only difference is that *X1_obs* (center) has one additional state ("?") for missing values, representing 10% of the observations. Thus, deleting the missing observations of an MCAR variable should not bias the estimation of its distribution. In BayesiaLab, we can simulate this assumption by setting negative evidence on "?" (green arrow labeled "Delete"). As we can see, the distribution of *X1_obs* (right) is now exactly the same as the one of *X1* (left).

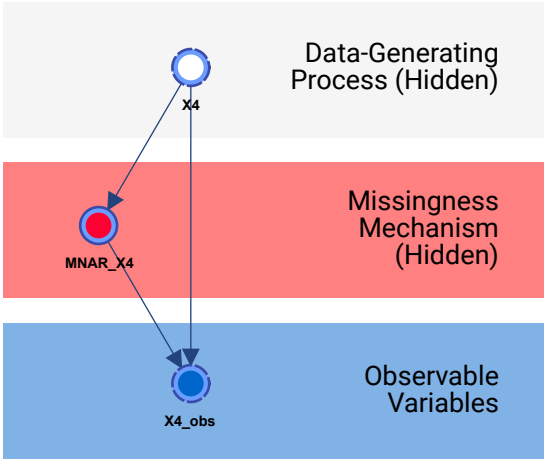
Under real-world conditions, however, we typically do not know whether the missing values in our dataset were generated completely at random (MCAR). This would be a strong assumption to make, and it is generally not testable. As a result, we can rarely rely on this fairly benign condition of missingness and, thus, should never be too confident in deleting missing observations.

Missing Not at Random (MNAR)

> Missing Not at Random (MNAR) or Not Missing at Random (NMAR) are equivalent expressions; MNAR and NMAR appear equally frequently in the literature. We use MNAR throughout this chapter.

Missing Not at Random (MNAR) refers to situations in which the missingness of a variable depends on hidden causes (unobserved variables), such as the data-generating variable itself. This condition is depicted in the subnetwork below.

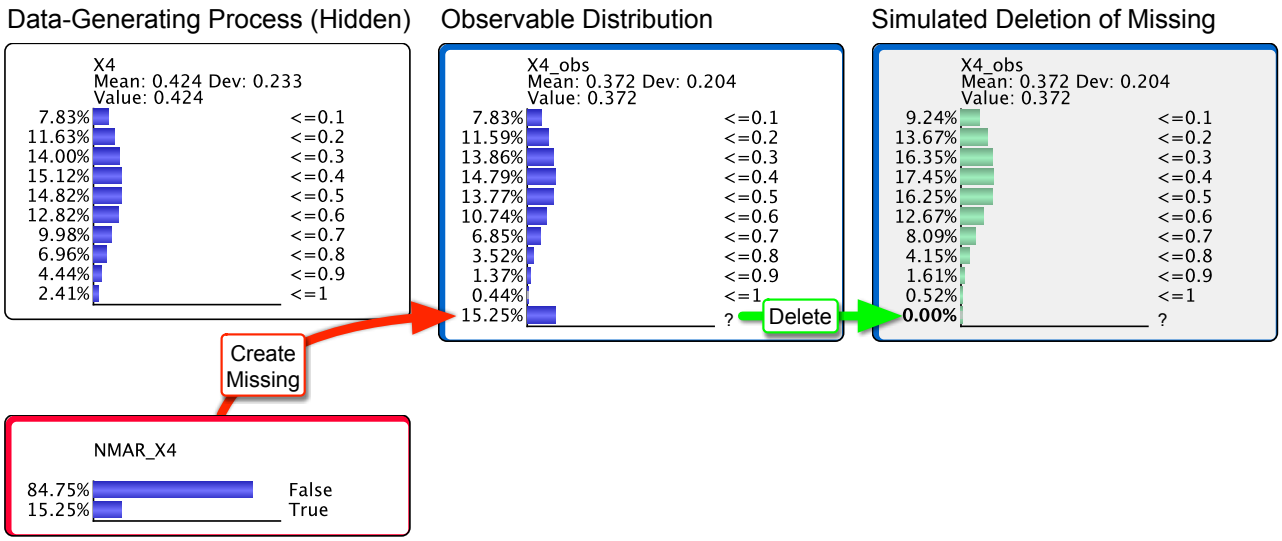
An example of the MNAR condition would be a hypothetical telephone survey about alcohol consumption. Heavy drinkers might decline to provide an answer out of fear of embarrassment. On the other hand, survey participants who drink very little or nothing at all might readily report their actual drinking habits. As a result, the missingness is a function of the very variable in which we are interested.



In order to proceed to simulation, we need to specify the parameters of the missingness mechanism and the observable variable:

- *X4* is a continuous variable with values between 0 and 1, and a Normal distribution models the DGP.
- *MNAR_X4* is a boolean variable with one parent, which specifies that the missingness probability depends directly on the hidden variable X4. However, the exact values are unimportant. We need to state that the probabilities of missingness are proportional to the values of *X4*: $P(MNAR_{X4} = true \mid X4) \propto X4$
- *X4_obs* has two parents, i.e., the data-generating variable *X4* and the missingness mechanism *MNAR_X4*.
- The following deterministic rule defines the conditional probability distribution: `IF MNAR_X4 THEN X4_obs=?  ELSE X4_obs=X4`

The impact of the mechanism of the missing value becomes apparent as we compare the Monitors of the network side by side.



As the above screenshot shows, the mean and standard deviation in the Monitor of *X4_obs* (center column) indicate that the distribution of the observed values of *X4* differs significantly from the original distribution (left column), leading to an underestimation of *X4* in this example. We can simulate the deletion of incomplete observations by setting negative evidence on "?" (green arrow labeled "Delete"). The simulated distribution of *X4_obs* (right column) indeed differs from the one of *X4* (left column).

Chapter 10: Causal Effect Identification and Estimation

Introduction

> Δημόκριτοςέλεγεβούλεσθαιμάλλονμίανευρείναιτιολογίανή τηνΠερσώνβασιλείανεαυτού γενέσθαι." ("Democritus used to say that 'he prefers to discover a causality rather than become a king of Persia'.") — Democritus, according to a late testimony of Dionysius, Bishop of Alexandria, by Eusebius of Caesarea in Præparatio Evangelica (Ευαγγελικὴ προπαρασκευή)

Bayesian Belief networks and modern causality analysis are intimately tied to the seminal works of Judea Pearl. It is presumably fair to say that one of the "unique selling points" of Bayesian networks is their capability to perform causal inference. However, we do want to go beyond merely demonstrating the mechanics of causal inference. Rather, we want to establish under what conditions causal inference can be performed. More specifically, we want to see the assumptions required to perform causal inference with non-experimental data.

To approach this topic, we need to break with the pattern established in the earlier chapters of this book. Instead of starting with a case study, we start off at a higher level of abstraction. First, we discuss in theoretical terms what is required for performing causal identification, estimation, and inference. Once these fundamentals are established, we can proceed to discuss the methods, along with their limitations, including Directed Acyclic Graphs and Bayesian networks. These techniques can help us distinguish causation from association when working with non-experimental data.

This chapter was prepared in collaboration with Felix Elwert on the basis of his course, Causal Inference with Graphical Models.

Motivation: Causality for Policy Assessment and Impact Analysis

In this chapter, we discuss causality mostly on the basis of a "toy problem," i.e., a simplified and exaggerated version of a real-world challenge. As such, the issues we raise about causality may appear somewhat contrived. Additionally, the constant practical use of causal inference in our daily lives may make our discussion seem somewhat artificial.

To highlight the importance of causal inference on a large scale, we want to consider how and under what conditions big decisions are typically made. Major government or business initiatives generally call for extensive studies to anticipate the consequences of actions not yet taken. Such studies are often referred to as "policy analysis" or "impact assessment":

> "Impact assessment, simply defined, is the process of identifying the future consequences of a current or proposed action." (IAIA, 2009)

> "Policy assessment seeks to inform decision-makers by predicting and evaluating the potential impacts of policy options." (Adelle and Weiland, 2012)

What can be the source of such predictive powers? Policy analysis must discover a causal mechanism that links a proposed action/policy to a potential consequence/impact. Unfortunately, experiments are typically out of the question in this context. Rather, impact assessments–from non-experimental observations alone–must determine the existence and the size of a causal effect.

Given the sheer number of impact analyses performed and their tremendous weight in decision-making, one would like to believe that there has been a long-established scientific foundation with regard to (non-experimental) causal effect identification, estimation, and inference. Quite naturally, as decision-makers quote statistics in support of policies, the field of statistics comes to mind as the discipline that studies such causal questions.

However, casual observers may be surprised to hear that causality has been anathema to statisticians for the longest time.

> "Considerations of causality should be treated as they always have been treated in statistics, preferably not at all..." (Speed, 1990).

The repercussions of this chasm between statistics and causality can still be felt today. Judea Pearl highlights this unfortunate state of affairs in the preface of his book Causality:

> "... I see no greater impediment to scientific progress than the prevailing practice of focusing all our mathematical resources on probabilistic and statistical inferences while leaving causal considerations to the mercy of intuition and good judgment." (Pearl, 1999)

Rubin (1974) and Holland (1986), who introduced the counterfactual (potential outcomes) approach to causal inference, can be credited with overcoming statisticians' traditional reluctance to engage causality. However, it will take many years for this fairly recent academic consensus to fully reach the world of practitioners, which is one of our key drivers for promoting Bayesian networks.

Key Concepts, Methods, and Practical Implementation

Please read the following subchapters in sequence to receive a complete explanation.

- **Sources of Causal Information**
- **Example: Simpson's Paradox**
- **Causal Identification and Estimation**
- **Example: Augmented Simpson's Paradox**

Causal Identification and Estimation

We will now explore formal methodologies that can help us derive causal effects from observational data. These methodologies will ultimately allow us to answer the question raised by Simpson's Paradox. The process of determining the size of a causal effect from observational data can be divided into two steps, namely identification, and estimation.

Identification

Identification analysis is about determining whether or not a causal effect can be established from the observed data. This requires a formal causal model or at least partial knowledge of how the data was generated. In this chapter, all causal assumptions for identification are expressed explicitly in the form of a Directed Acyclic Graph (DAG) (Pearl 1995, 2009). They represent our complete causal understanding of the DGP for the system we are studying.

Where do we get such causal assumptions? We would like to say that advanced algorithms can generate causal assumptions from data. That is not the case, unfortunately. Causal assumptions do still require human expert knowledge or, more generally, theory. In practice, this means that we need to build (or draw) a causal graph of our domain. Then, we can examine this graph against formal criteria, which determine whether the effect is identifiable or not.

It is important to realize that the absence of causal assumptions cannot be compensated for by clever statistical techniques or by providing more data. So, recognizing that a causal effect is not identifiable brings the effect analysis to an abrupt halt.

Estimation

But if the causal effect is identifiable, we can proceed to estimate the effect size. The same criteria that determine identifiability do also tell us how to perform the effect estimation. With that, we can utilize the available observational data and estimate the causal effect. Depending on the complexity of the domain, the effect estimation can bring a new set of challenges. However, in the context of Simpson's Paradox, the effect estimation will be very straightforward.

Identification and Estimation Methods

- **Causal DAGs**
- **Graphical Identification Criteria**
- **Effect Estimation**

Causal DAGs

We need to understand some important properties before encoding our causal knowledge in a DAG. We learned in Chapter 2 that Bayesian networks use DAGs for the qualitative description of the Joint Probability Distribution.

In the causal context, however, the arcs in a DAG explicitly state causality instead of only representing direct probabilistic dependencies in a Bayesian network. We now designate a DAG with a causal semantic as a Causal DAG (CDAG) to highlight this distinction.

Structures Within a DAG

A DAG has three basic configurations in which nodes can be connected. Graphs of any size and complexity can be broken down into these basic graph structures. While these basic structures show direct dependencies/causes explicitly, there are more statements contained in them, albeit implicitly. In fact, we can read all marginal and conditional associations that exist between the nodes.

Why are we even interested in associations? Isn't all this about understanding causal effects? It is essential to understand all associations in a system because, in non-experimental data, all we can do is observe associations, some of which represent non-causal relationships. Our objective is to identify causal effects from associations.

- .

  Indirect Connection

  This DAG represents an indirect connection of A on B via C.

  – A Directed Arc represents a potential causal effect. The arc direction indicates the assumed causal direction, i.e., $A \to C$ means A causes C.
  – A Missing Arc encodes the definitive absence of a direct causal effect, i.e., no arc between A and B means no direct causal relationship exists between A and B and vice versa. As such, a missing arc represents an assumption.

  Implication for Causality

  A has a potential causal effect on B mediated by C.

  Implication for Association

  Marginally (or unconditionally), A and B are dependent. This means that without knowing the exact value of C, learning about A informs us about B and vice versa, i.e., the path between the nodes is unblocked, and information can flow in both directions.

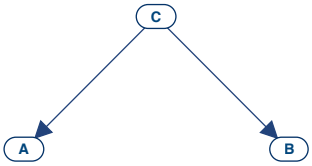
  Conditionally on C, i.e., by setting Hard Evidence on (or observing) C, A, and B become independent. In other words, by "hard"-conditioning on C, we block the path from A to B and from B to A. Thus, A and B are conditionally independent, given C:

  $A \not\perp B,\ A \perp B \mid C$

  > Hard Evidence means that there is no uncertainty regarding the value of the observation or evidence. If uncertainty remains regarding the value of C, the path will not be entirely blocked, and an association will remain between A and B.

- .

  Common Parent

  The second configuration has C as the common parent of A and B.

Implication for Causality

C is the common cause of both A and B.

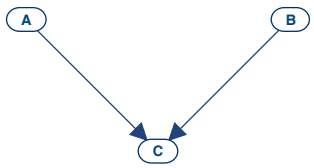
Implication for Association

In terms of association, this structure is absolutely equivalent to the Indirect Connection. Thus, A and B are marginally dependent but conditionally independent given C (by setting Hard Evidence on C):

$A \not\perp B,\ A \perp B \mid C$

- .

Common Child (Collider)



The final structure has a common child C, with A and B being its parents. This structure is called a "V-Structure." In this configuration, the common child C is also known as a "collider."

Implication for Causality

A and B are the direct causes of C.

Implication for Association

Marginally (or unconditionally), A and B are independent, i.e., there is no information flow between A and B. Conditionally on C — with any kind of evidence — A and B become dependent. If we condition on the collider C, information can flow between A and B, i.e., conditioning on C opens the information flow between A and B:

$A \perp B,\ A \not\perp B \mid C$

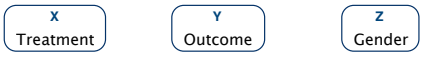
> Even introducing a minor change in the distribution of C, e.g., from no observation ("color unknown") to a very vague observation ("it could be anything, but it is probably not purple"), opens the information flow.

For purposes of formal reasoning, this type of connection is of special significance. Conditioning on C facilitates inter-causal reasoning, often referred to as the ability to "explain away" the other cause, given that the common effect is observed (see Inter-Causal Reasoning in Chapter 4).

Creating a CDAG Representing Simpson's Paradox

Step 1:

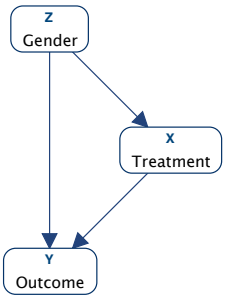
To begin the encoding of our causal knowledge in the form of a CDAG, we draw three nodes, which represent $X(Treatment), Y(Outcome)$, and $Z(Gender)$. For now, we are only using the qualitative part of the network, i.e., we are not considering probabilities.



The absence of further nodes means that we assume that there are no additional variables in the Data-Generating Process (DGP), either observable or unobservable. Unfortunately, this is a very strong assumption that cannot be tested. We need to have a justification purely on theoretical grounds to make such an assumption.

Step 2:

In the next step, we must encode our causal assumptions regarding this domain. Given our background knowledge of this domain, we state that Z causes X and Y and that X causes Y.



This means that we believe that gender is a cause of taking the treatment and has a causal effect on the outcome, too. We also assume that the treatment has a potential causal effect on the outcome.

Step 3:

Having accepted these causal assumptions, we now wish to identify the causal effect of X on Y. The question is whether this is possible on the basis of this causal graph and the available observational data for these three variables. Before we can answer this question, we need to think about what this CDAG specifically implies. Recall the types of structures that can exist in a DAG (see Structures Within a DAG). As it turns out, we can find all three of the basic structures in this example:

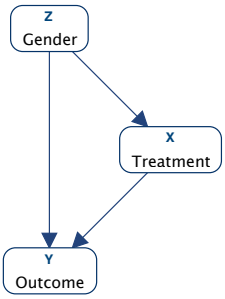
- Indirect Connection: Z causes Y via X
- Common Parent: Z causes X and Y
- Common Child: Z and X cause Y

Effect Estimation

Estimation

Returning to the original version of the CDAG, without the hidden variable, we are now ready to proceed with the estimation. However, this CDAG is only a qualitative representation of our theory about the DGP. We now need to consider this graph as a model representing the joint probability distribution of our three variables $P(X, Y, Z)$.

We do not yet need to determine what this probability function is; we simply need to consider this graph as a non-parametric probability function linking X, Y, and Z. This will help us understand what it means to adjust for Z to estimate the causal effect.



Estimation Methods

General Methods

- **Graph Surgery**
- **Adjustment Formula**

Methods in BayesiaLab

- **Causal Effect Estimation in BayesiaLab with Graph Surgery**
- **Causal Effect Estimation in BayesiaLab with Likelihood Matching**

Adjustment Formula

How can we translate the abstract concept of Graph Surgery into something that can compute actual numerical values? In fact, we can work directly with graphs — in the form of Bayesian networks — and use BayesiaLab to perform Graph Surgery and simulate interventions.

However, before we illustrate that in the next section of this chapter, we want to formally conclude the line of reasoning that connects the pre-intervention distribution P to the post-intervention distribution P_m and introduce the Adjustment Formula. We paraphrase Pearl, Glymour, and Jewell (2016), p. 56f. to develop this formula.

In our example, we can easily estimate the pre-intervention distribution P from the available data, but we need the post-intervention distribution P_m to calculate the causal effect. The key lies in recognizing that P_m shares two essential properties with P.

First, the marginal distribution $P(Z = z)$ remains invariant under the intervention because the process of determining Z is unaffected by removing the arc $Z \to X$. In our example, this means that the share of men and women must remain the same before and after the intervention:

$$P_m(Z = z) = P(Z = z)$$

Secondly, the conditional probability distribution $P(Y = y \mid Z = z, X = x)$ remains invariant under the intervention because the process that determines how Y responds to X and Z stays the same, regardless of whether X changes naturally or through external intervention. We can state this formally as follows:

$$P_m(Y = y \mid Z = z, X = x) = P(Y = y \mid Z = z, X = x)$$

Furthermore, X and Z are marginally independent in the mutilated graph. This means that the conditional probability distribution of Z given X in the mutilated graph is the same as the marginal probability distribution of Z in the pre-intervention graph:

$$P_m\left(Z = z \mid X = x\right) = P_m\left(Z = z\right) = P\left(Z = z\right)$$

Since the Adjustment Criterion is satisfied in the mutilated graph, we have the following:

$$P(Y = y \mid do(X = x)) = P_m(Y = y \mid X = x)$$

By conditioning on Z and summing over all values z, we obtain:

$$P(Y = y \mid do(X = x)) = \sum_z P_m(Y = y \mid X = x, Z = z) P_m(Z = z \mid X = x)$$

Furthermore, X and Z are independent in the mutilated graph:

$$P(Y = y \mid do(X = x)) = \sum_z P_m(Y = y \mid X = x, Z = z) P_m(Z = z)$$

Using the two invariance equations above, we obtain what is known as the Adjustment Formula. It expresses the post-intervention distribution exclusively in terms of the pre-intervention distribution:

$$P(Y = y \mid do(X = x)) = \sum_z P(Y = y \mid X = x, Z = z) P(Z = z)$$

The Adjustment Formula computes the association between X and Y for each value z (or strata of $z \in Z$) and then produces a weighted average. On this basis, we can now estimate the Average Causal Effect (ACE):

$$ACE = P\left(Y = 1 \mid do(X = 1)\right) - P\left(Y = 1 \mid do(X = 0)\right)$$

We know that by performing a randomized experiment, we obtain an unbiased estimate of the causal effect of the treatment. More specifically, through randomization, we randomly split the patient population into two sub-populations and forced the first group to receive treatment, and withheld the treatment from the second group. Through the random assignment of the treatment, we ensure that there is no association between Z and X. Also, all other properties remain unaffected by the randomization of treatment, including the distribution of Z, the relationship between Z and Y, and the relationship between X and Y.

As a result, graph surgery can be seen as a "randomization after the fact." However, we need to realize that performing graph surgery can only achieve quasi-randomization with regard to observed and known confounders, in our case Z. A randomized experiment, however, can make treatment independent of all other confounders, observed, unobserved, and unknown. Thus, randomized experiments remain the gold standard for establishing causal effects.
All our efforts in estimating causal effects through adjustment or graph surgery are merely an attempt to mimic the properties of a randomized experiment. Unfortunately, we can never measure how close we are to achieving this goal. We can only be disciplined with our assumptions and make a causal claim based on that.

Simpson's Paradox Resolved

Returning to Simpson's Paradox, the equation

$$P(Y = y \mid do(X = x)) = \sum_z P(Y = y \mid X = x, Z = z)P(Z = z)$$

gives us the answer to our question of whether we need to look at the aggregate data table or the gender-specific data table for determining the true causal effect of treatment on the outcome: "Conditioning on Z and summing over all values z" means that we need to utilize the gender-specific table. More specifically, we need to compute the association between X and Y for each value of Z, i.e., each stratum of $z \in Z$, and then calculate the weighted average. This estimation method is also known as stratification.

Aggregate Table

	Patient Recovered	
Treatment	**Yes**	**No**
Yes	50%	50%
No	40%	60%

Gender-Specific Table

		Patient Recovered	
Gender	**Treatment**	**Yes**	**No**
Male	Yes	60%	40%
	No	70%	30%
Female	Yes	20%	80%
	No	30%	70%

$$\begin{aligned}
ACE &= P(Y = 1 \mid do(X = 1)) - P(Y = 1 \mid do(X = 0)) \\
&= (P(Y = 1 \mid X = 1, Z = 0)P(Z = 0) + P(Y = 1 \mid X = 1, Z = 1)P(Z = 1)) \\
&\quad - (P(Y = 1 \mid X = 0, Z = 0)P(Z = 0) + P(Y = 1 \mid X = 0, Z = 1)P(Z = 1)) \\
&= (0.2 \times 0.5 + 0.6 \times 0.5) - (0.3 \times 0.5 + 0.7 \times 0.5) \\
&= -0.1
\end{aligned}$$

The ACE turns out to be negative, i.e., it has the opposite sign of what we would have inferred by merely looking naively at the association between treatment and outcome. This illustrates that a

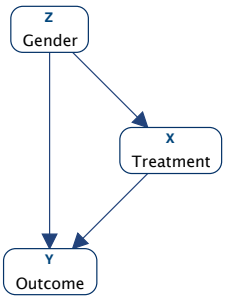
bias in the estimation of an effect can be more than just a nuisance for the analyst. Bias can reverse the sign of the effect! In our example, the treatment under study would kill people instead of healing them.

The good news is that we have a theory stipulating that gender is a confounder, and this variable is observed. If it were not recorded in our dataset (hidden variable), we would not be able to compute the causal effect of treatment. We can also imagine situations where we do not know that confounders exist and, therefore, do not measure them. This can lead to substantially wrong estimations of causal effects and lead to policies with catastrophic consequences.

Causal Effect Estimation in BayesiaLab with Graph Surgery

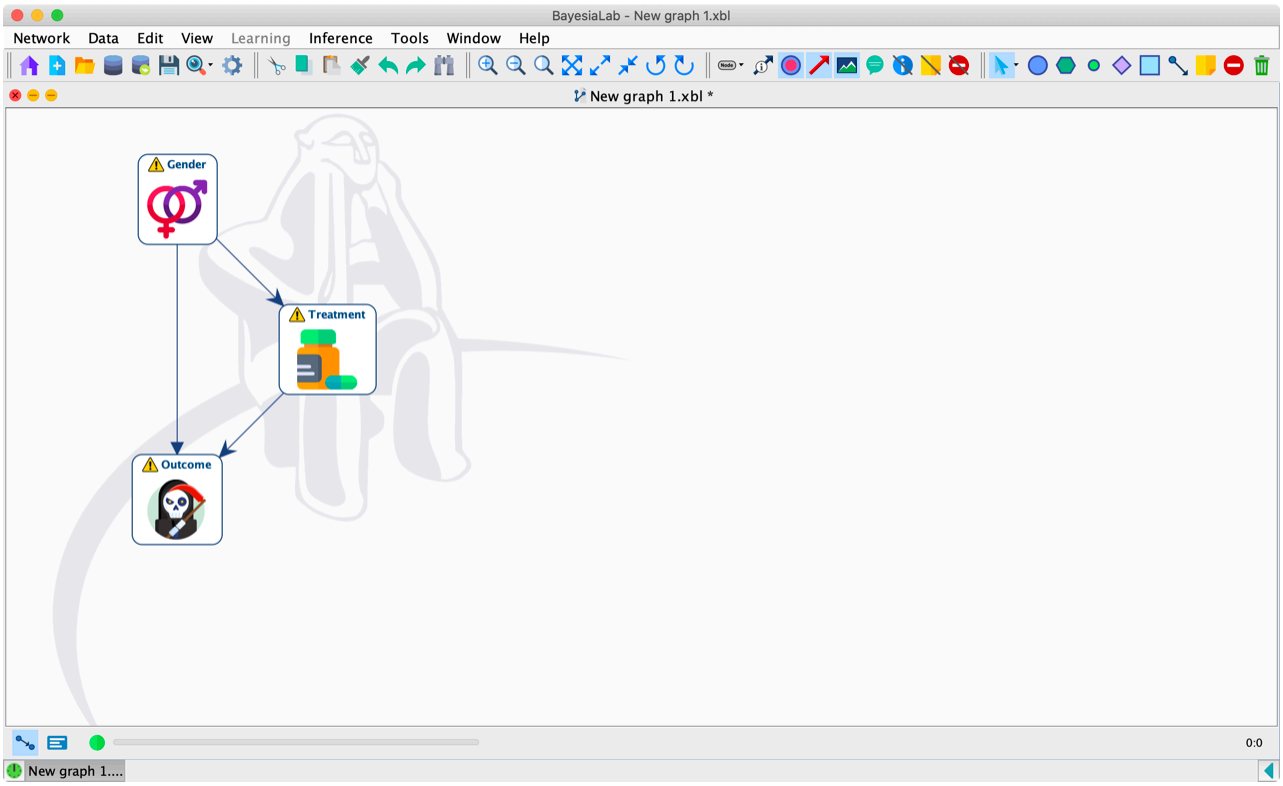
We now understand that Graph Surgery and Adjustment are equivalent. However, with Bayesian networks, we can go beyond the metaphor and–quite literally–perform graph surgery. In this section, we create a Bayesian network to represent Simpson's Paradox example and then perform graph surgery to estimate the causal effect.

We have already defined a causal graph earlier when we encoded our causal assumptions regarding this domain. We can reuse this causal understanding for building a causal Bayesian network in BayesiaLab.



As we illustrated in the context of the knowledge modeling exercise in Chapter 4, we manually create the nodes and draw the arcs on BayesiaLab's Graph Panel. We choose to use long names for the nodes instead of X, Y, and Z. Letters were very convenient for formulas, but long names increase the readability of Bayesian networks. To further help with interpretation, we also associate images with each node and display them as Badges. Then, we select `Menu > View > Layout > Genetic Grid Layout > Top-Down Repartition` to obtain a layout that takes into account the direction of the arcs and define layers accordingly.

> The Genetic Grid Layout algorithms are particularly useful for causal networks. We can, therefore, define one of these two algorithms as the one associated with the shortcut `P` via `Menu > Preferences > Window > Preferences > Automatic Layout > Layout Algorithm Associated with Shortcut`.



The yellow warning symbols ⚠ remind us that the probability tables associated with the nodes have yet to be defined. At this point, we could set the parameters based on our knowledge of all the probabilities in this domain. Instead, we utilize the available data and use BayesiaLab's Parameter Estimation to establish the quantitative part of this network via Maximum Likelihood Estimation. We have been using Parameter Estimation extensively in this book, either implicitly or explicitly, for instance, in the context of structural learning and missing values estimation (see Parameter Estimation in Chapter 5).

Parameter Estimation

Previously, we acquired the data needed for Parameter Estimation via the Data Import Wizard. Now we will use the Associate Data Wizard for the same purpose. Whereas the Data Import Wizard generates new nodes from columns in a database, the Associate Data Wizard links columns of data with existing nodes. This way, we can "fill" our qualitative network with data and then perform Parameter Estimation to generate the quantitative part of the network. We now show the corresponding steps in detail.

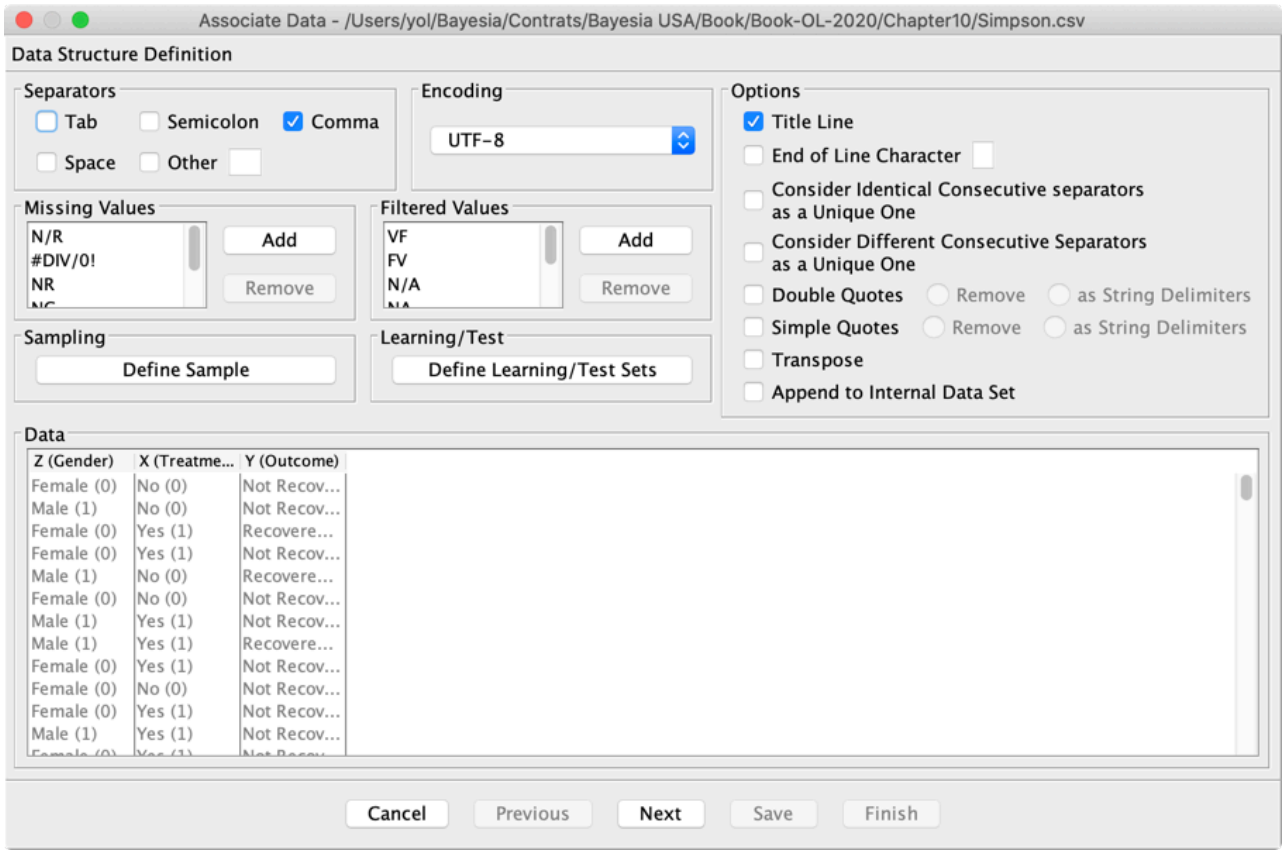
Step 1:

We start the Associate Data Wizard from the main menu: `Menu > Data > Associate Data Source > Text File`.

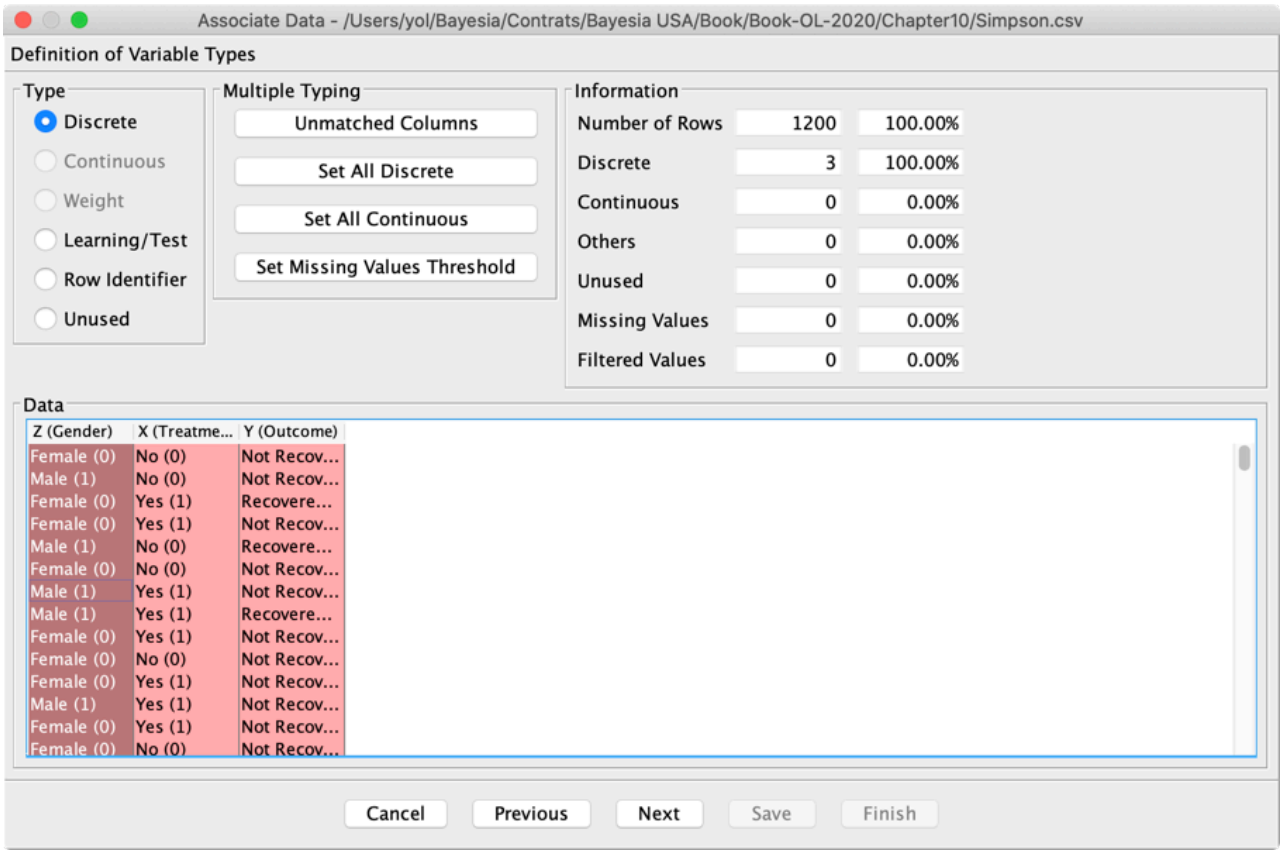
This prompts us to select the text file containing our observational data:

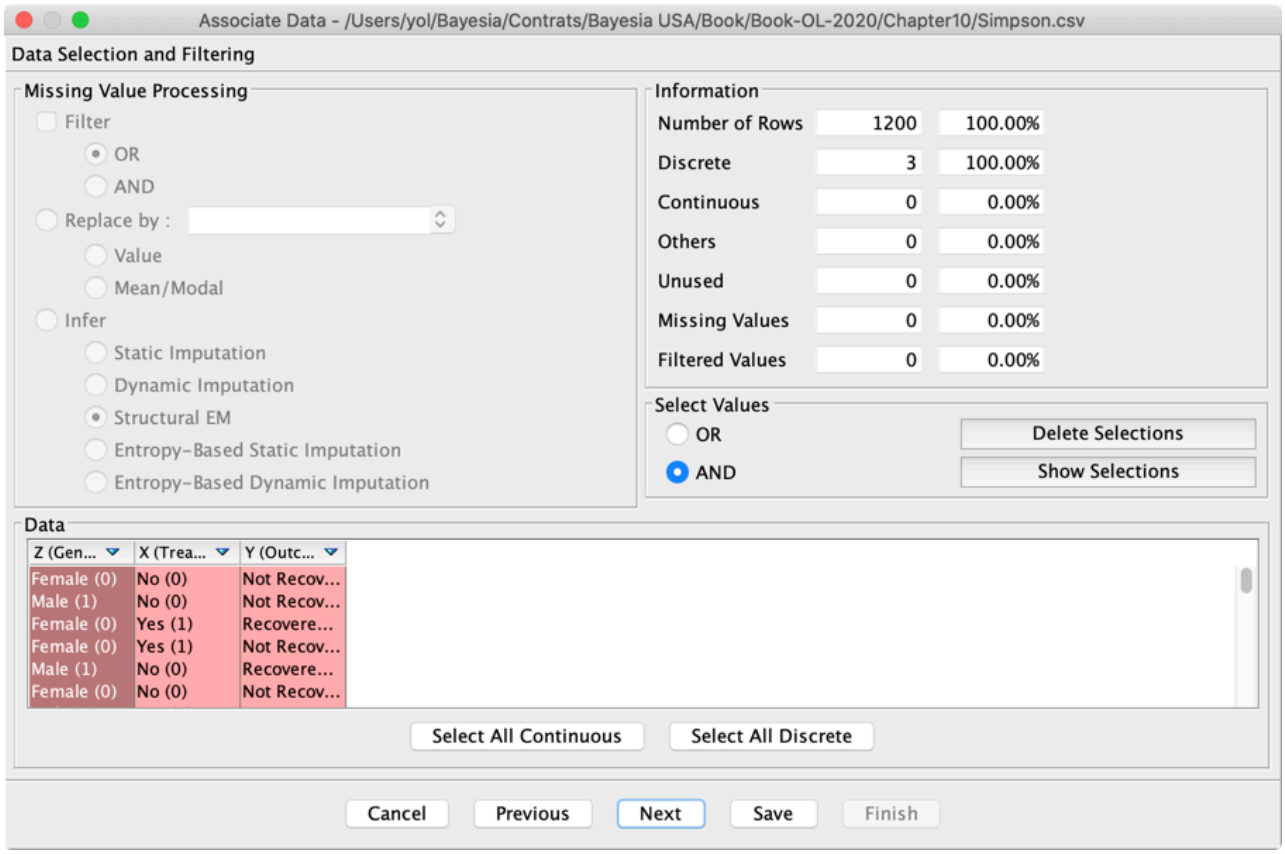
Download: `Simpson.csv`

Upon selecting the file, BayesiaLab brings up the first screen of the Associate Data Wizard.



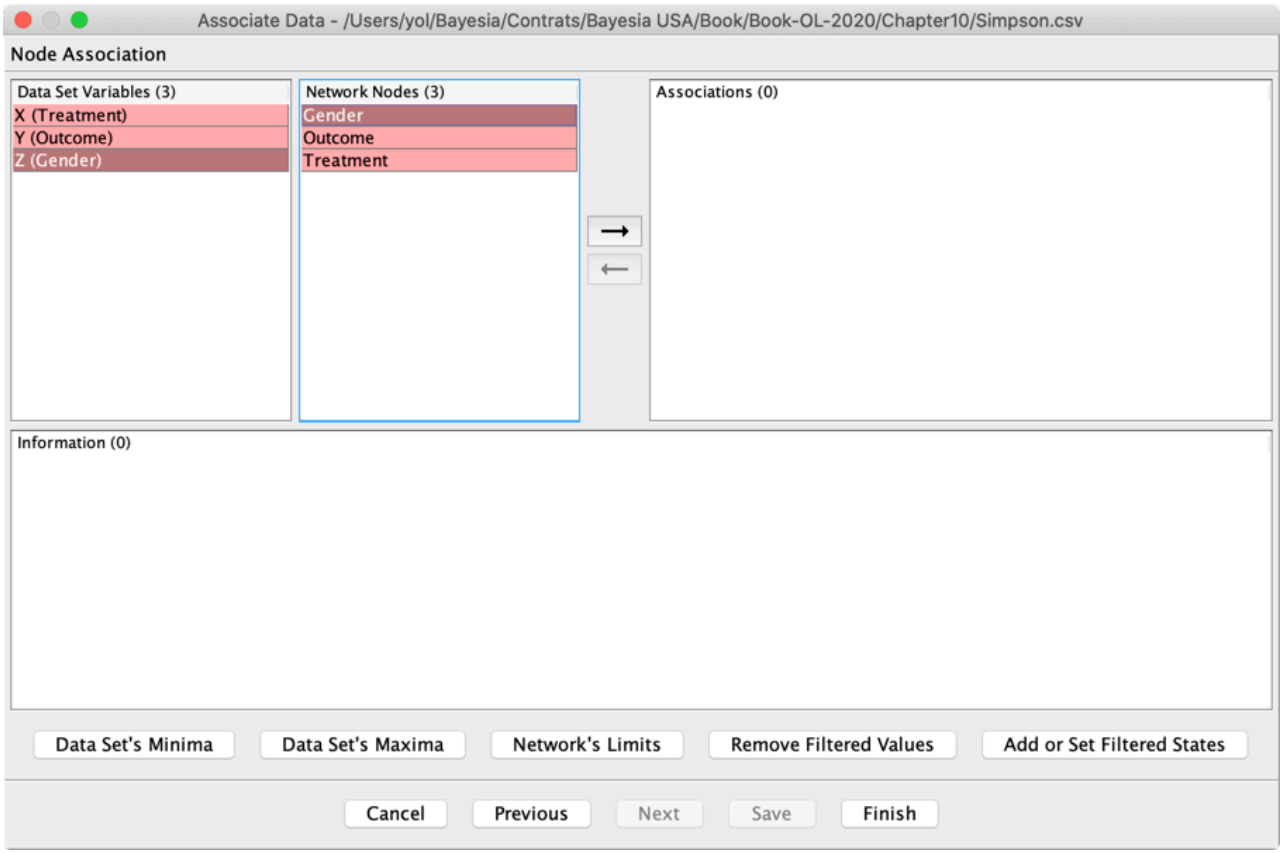
Given that the Associate Data Wizard mirrors the Data Import Wizard in most of its options, we omit to describe them here. We merely show the screens for reference as we click `Next` to progress through the wizard.


Associate Data - /Users/yol/Bayesia/Contrats/Bayesia USA/Book/Book-OL-2020/Chapter10/Simpson.csv
Definition of Variable Types
Type
Discrete
Continuous
Weight
Learning/Test
Row Identifier
Unused
Multiple Typing
Unmatched Columns
Set All Discrete
Set All Continuous
Set Missing Values Threshold
Information
Number of Rows 1200 100.00%
Discrete 3 100.00%
Continuous 0 0.00%
Others 0 0.00%
Unused 0 0.00%
Missing Values 0 0.00%
Filtered Values 0 0.00%
Data
Z (Gender) X (Treatme... Y (Outcome)
Female (0) No (0) Not Recov...
Male (1) No (0) Not Recov...
Female (0) Yes (1) Recovere...
Female (0) Yes (1) Not Recov...
Male (1) No (0) Recovere...
Female (0) No (0) Not Recov...
Male (1) Yes (1) Not Recov...
Male (1) Yes (1) Recovere...
Female (0) Yes (1) Not Recov...
Female (0) No (0) Not Recov...
Female (0) Yes (1) Not Recov...
Male (1) Yes (1) Not Recov...
Female (0) Yes (1) Not Recov...
Female (0) No (0) Not Recov...
Cancel
Previous
Next
Save
Finish

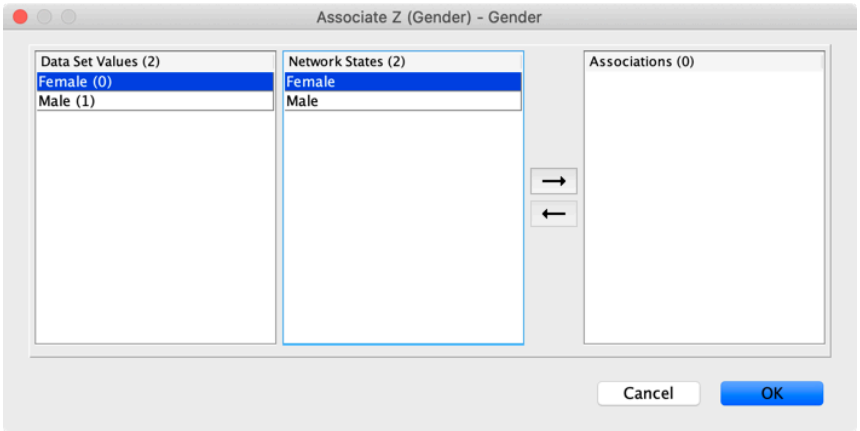


Step 2:

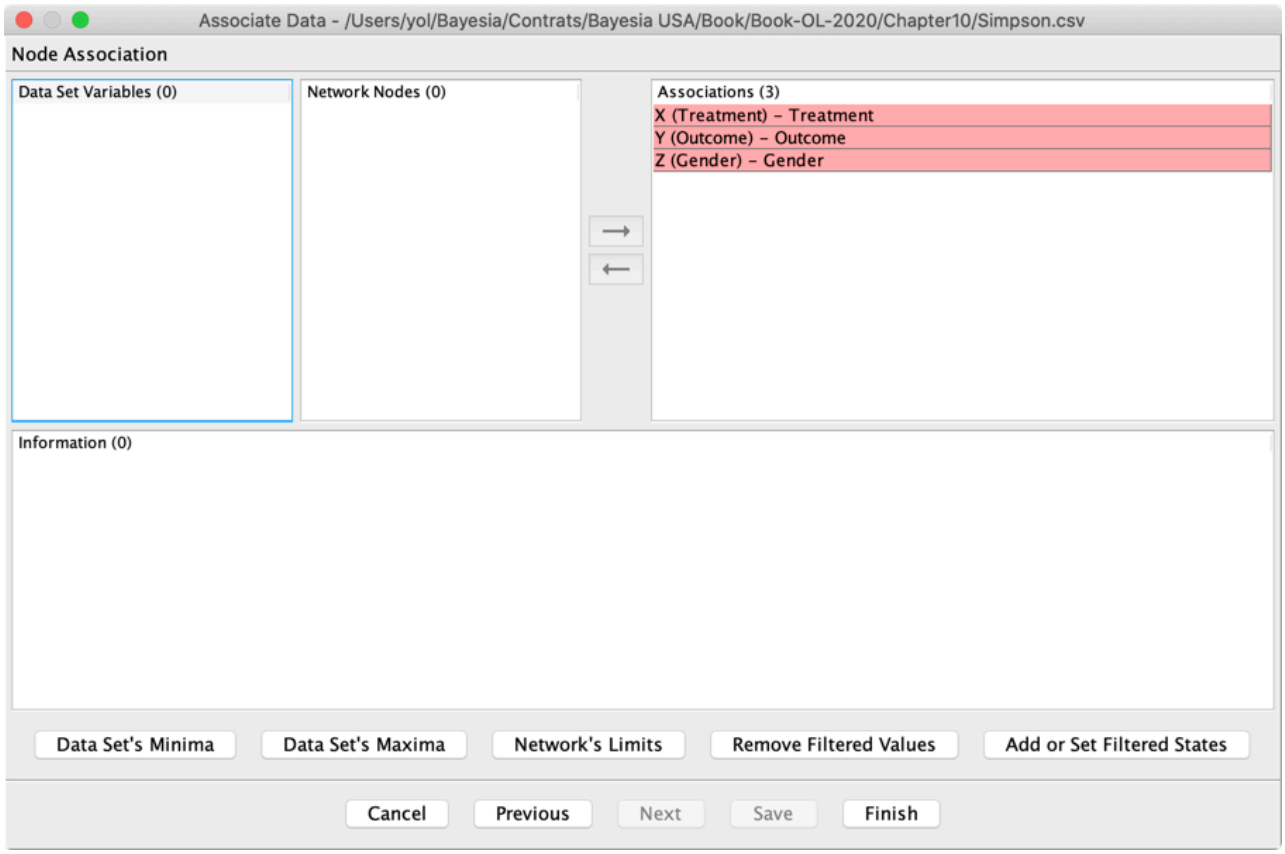
The last step shows how the variables in the dataset will be associated with the nodes of the network. If the column names in the dataset perfectly match the existing node names, BayesiaLab automatically creates an association. However, this is not the case in our example. Therefore, we have to manually define the association by iteratively selecting each `Dataset Variable` and `Network Node`, and then clicking on the right arrow.



Upon clicking on the right arrow, BayesiaLab brings up a screen for defining the association between the values used in the dataset and the states of the node. Again, the state names of our nodes do not correspond exactly to the values used in the dataset. So we have to manually define the association by iteratively selecting each `Dataset Value` and `Network State` and then clicking on the right arrow.

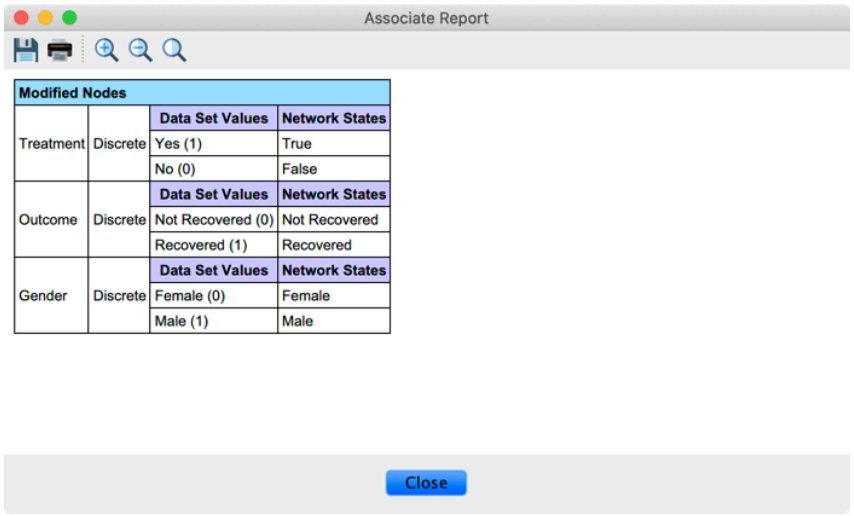


Once this is done for all three variables, the Associate Data Wizard displays how the columns in the dataset are associated with the nodes of the network.



Step 3:

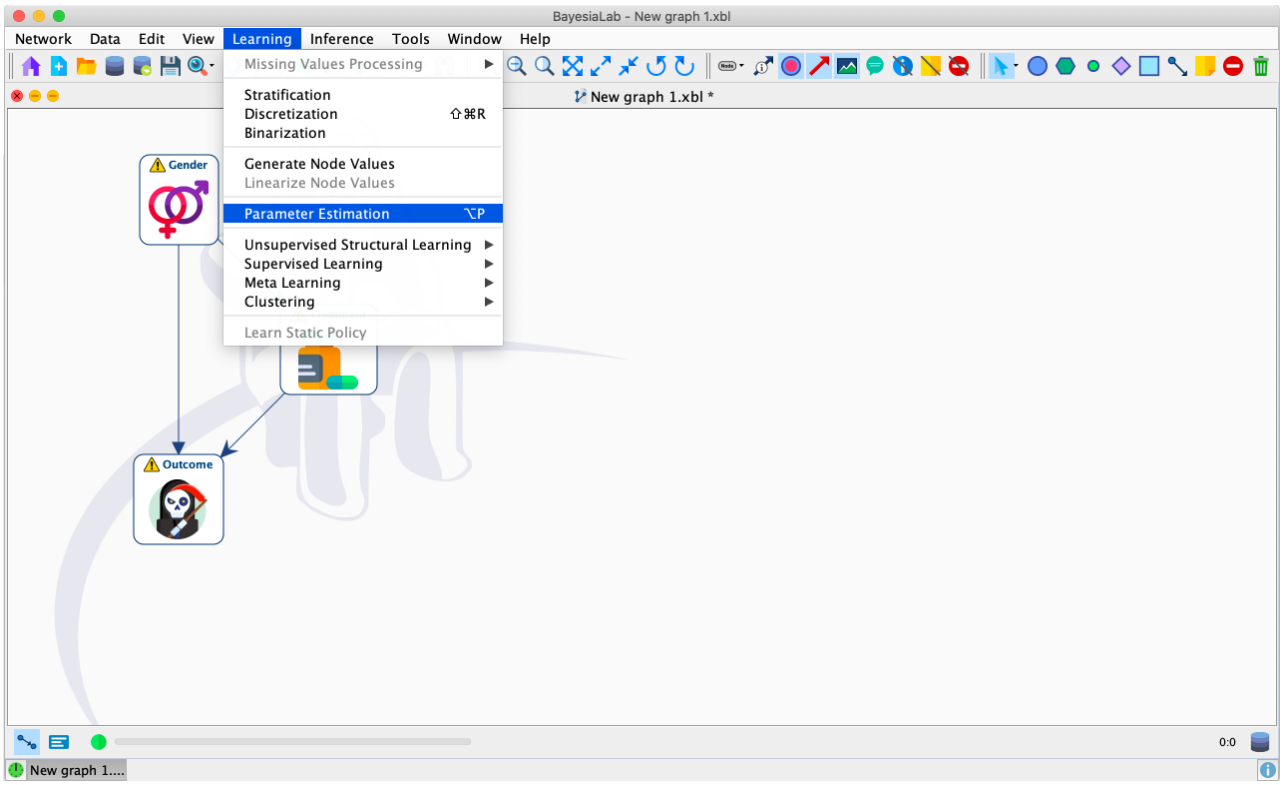
Upon clicking `Finish`, we are prompted whether we want to view the Associate Report.



Modified Nodes			
Treatment	Discrete	Data Set Values	Network States
		Yes (1)	True
		No (0)	False
Outcome	Discrete	Data Set Values	Network States
		Not Recovered (0)	Not Recovered
		Recovered (1)	Recovered
Gender	Discrete	Data Set Values	Network States
		Female (0)	Female
		Male (1)	Male

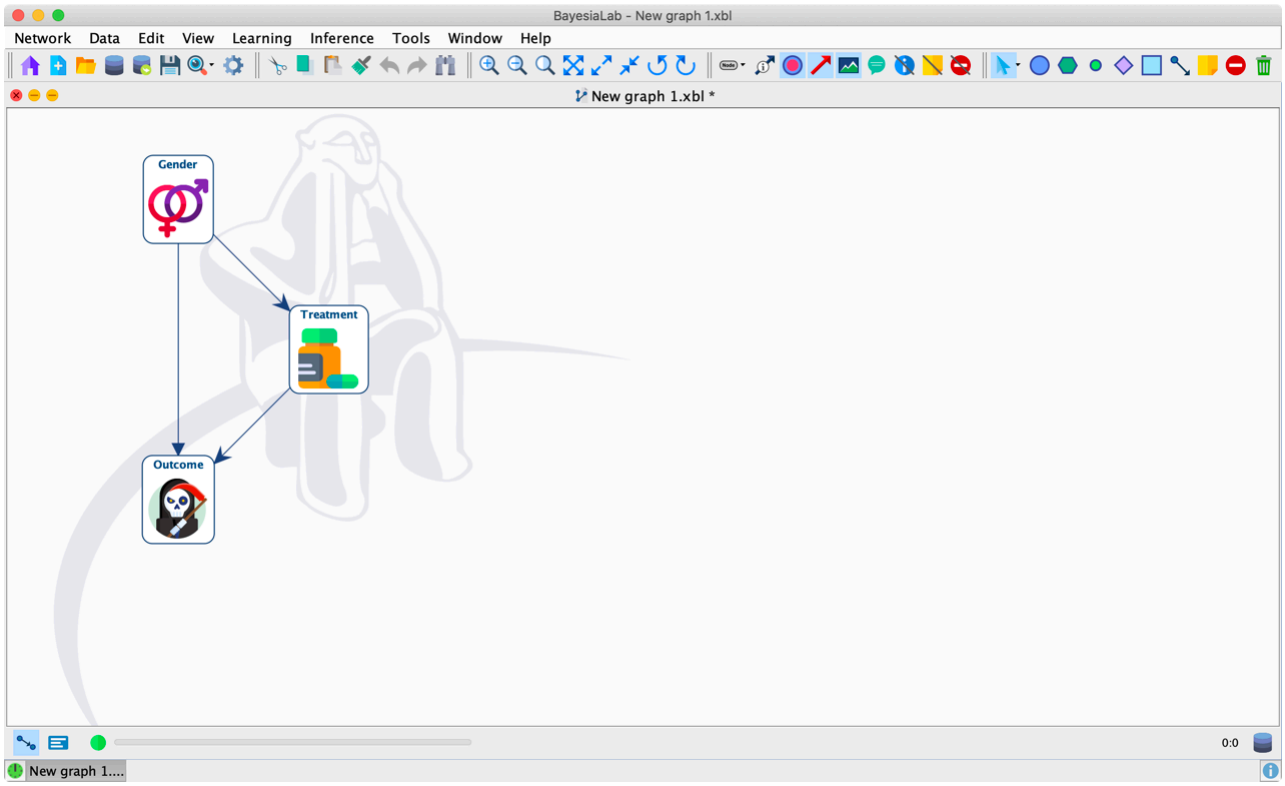
Step 4:

The Database icon in the lower right-hand corner of the main window indicates that our network now has a database associated with its structure. We now use this data to estimate the parameters of the network: `Learning > Parameter Estimation`.



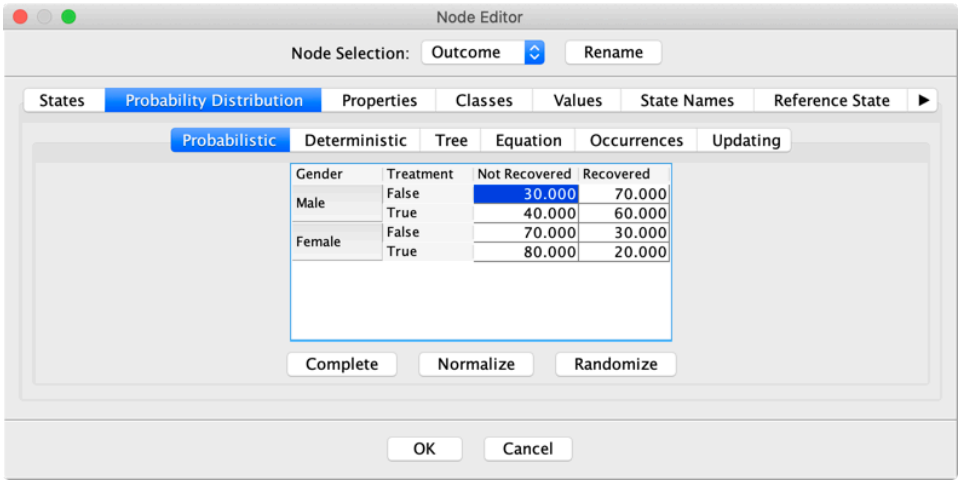
Step 5:

Once the parameters are estimated, there are no longer any warning symbols tagged onto the nodes.



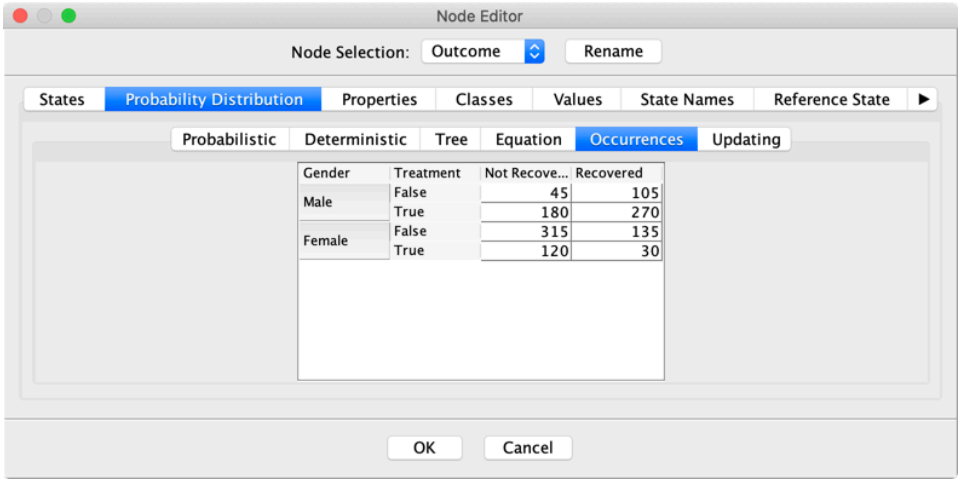
Step 6:

We now have a fully specified Bayesian network. By opening the Node Editor of *Outcome*, for instance, we see that the CPT is indeed filled with probabilities.



Step 7:

Upon clicking on the `Occurrences` tab, we can see the counts that were used by the Maximum Likelihood Estimation to derive these probabilities.



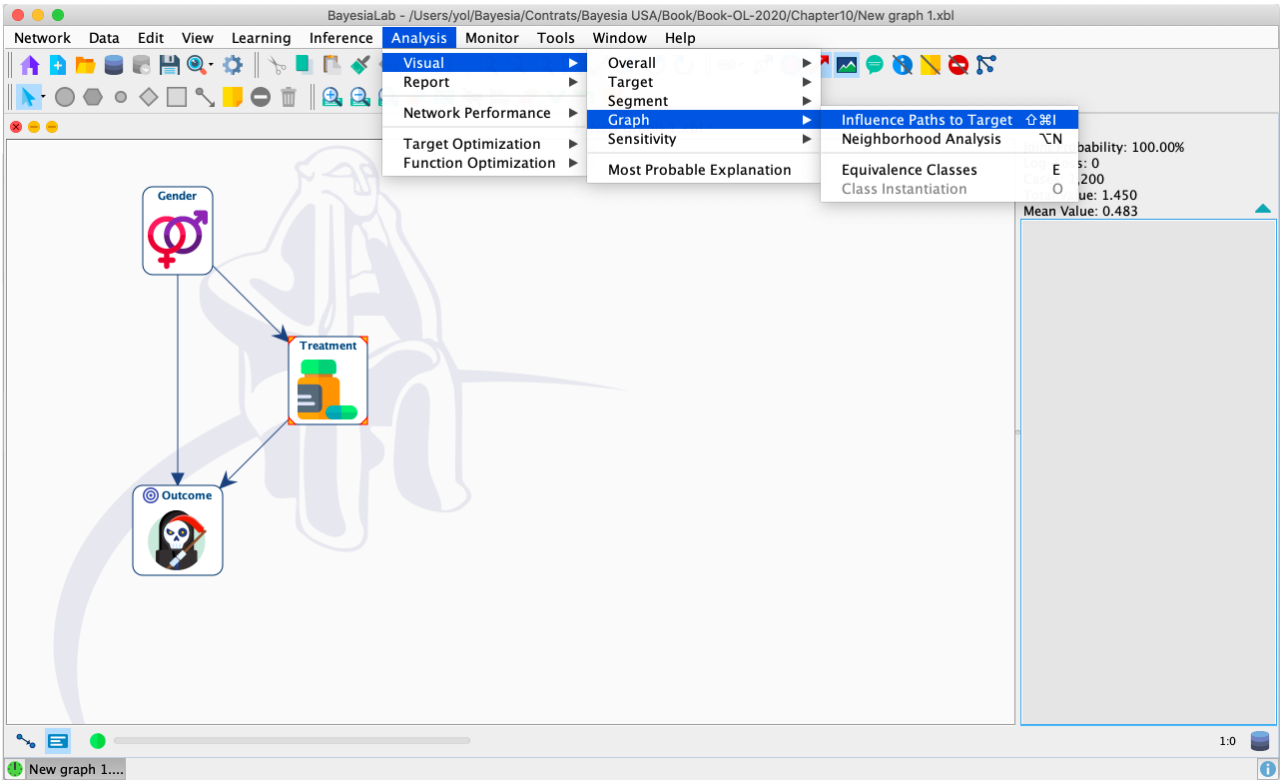
Path Analysis

Recall that distinguishing between causal and non-causal paths is crucial for the application of the Adjustment Criterion. BayesiaLab can help us review the paths that are present in the graph. Given that we already understand the paths, showing the formal path analysis with BayesiaLab is merely for reference.

Once we define *Outcome* as Target Node and switch into the Validation Mode `F5`, we can examine all possible paths to the Target Node in this network.

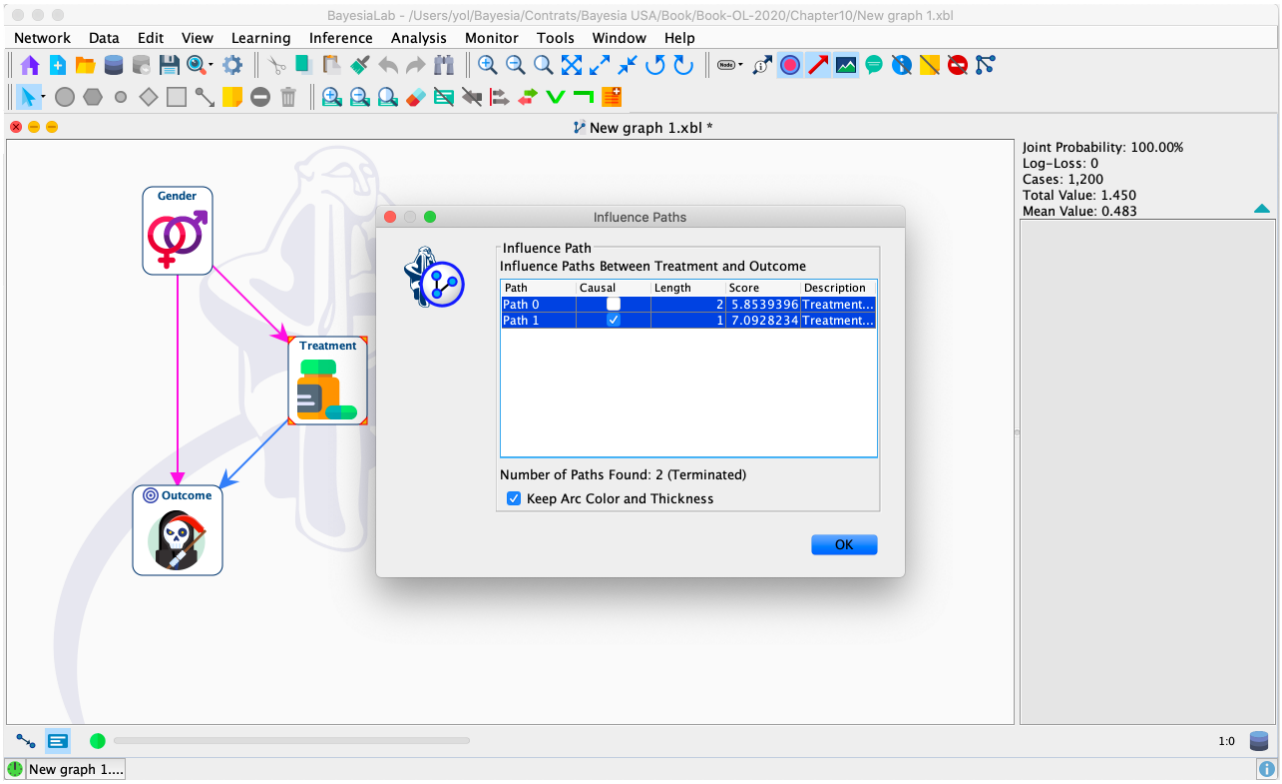
Step 1:

We select *Treatment* and then select `Menu > Analysis > Visual > Graph > Influence Paths to Target`.



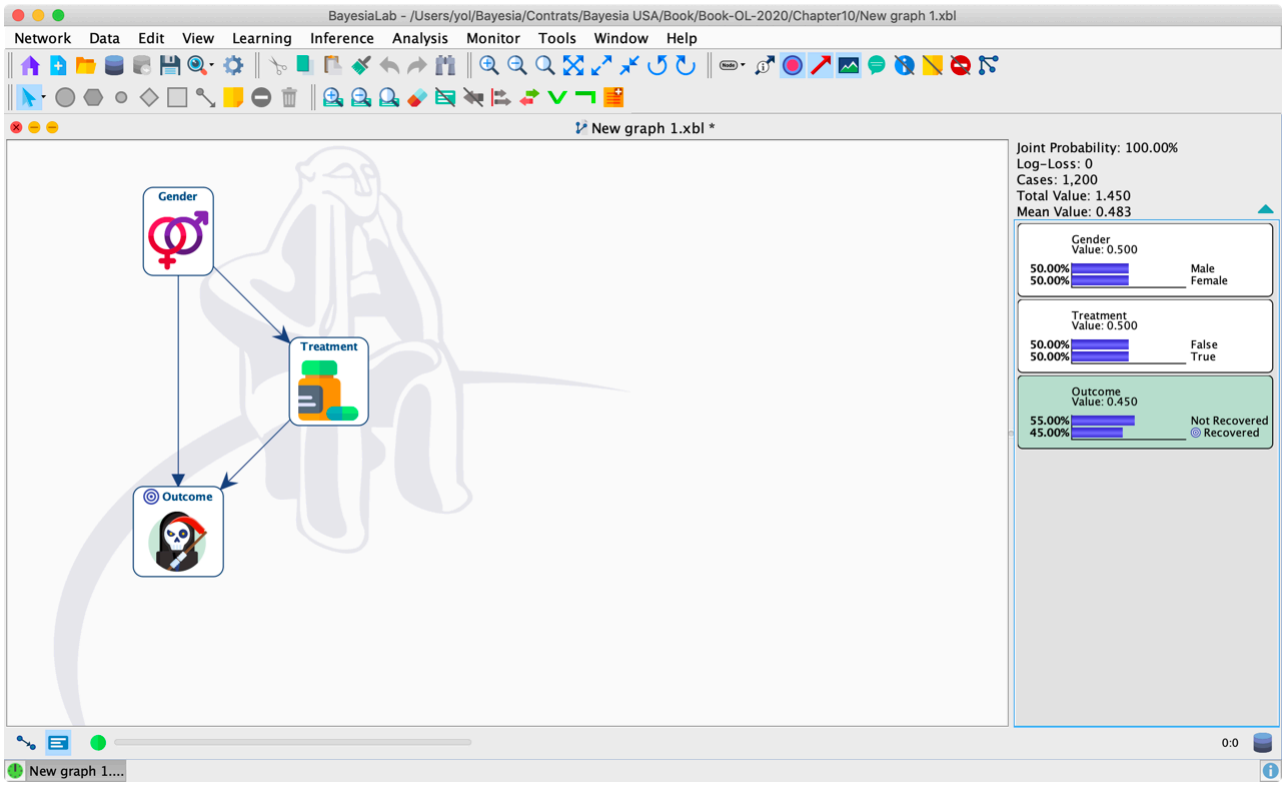
Step 2:

Then, BayesiaLab displays a pop-up window with the Influence Paths report. Selecting any of the listed paths shows the corresponding arcs in the Graph Panel. Causal paths are shown in blue; non-causal paths are pink.

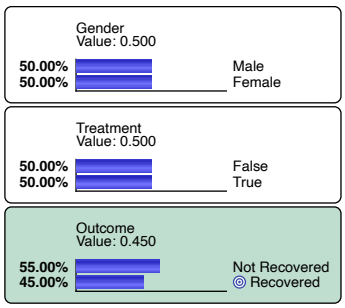


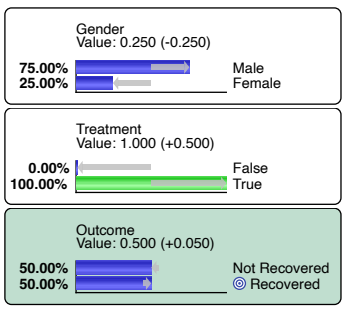
Observational Inference

Before proceeding to the effect estimation, we bring up the Monitors of all three nodes and compare the probabilities reported by the network with the Aggregate and Gender-Specific Tables, which gave rise to the paradox.



For instance, the screenshot below shows the prior distributions (left) and the posterior distributions (right) given the observation $Treatment = True$.



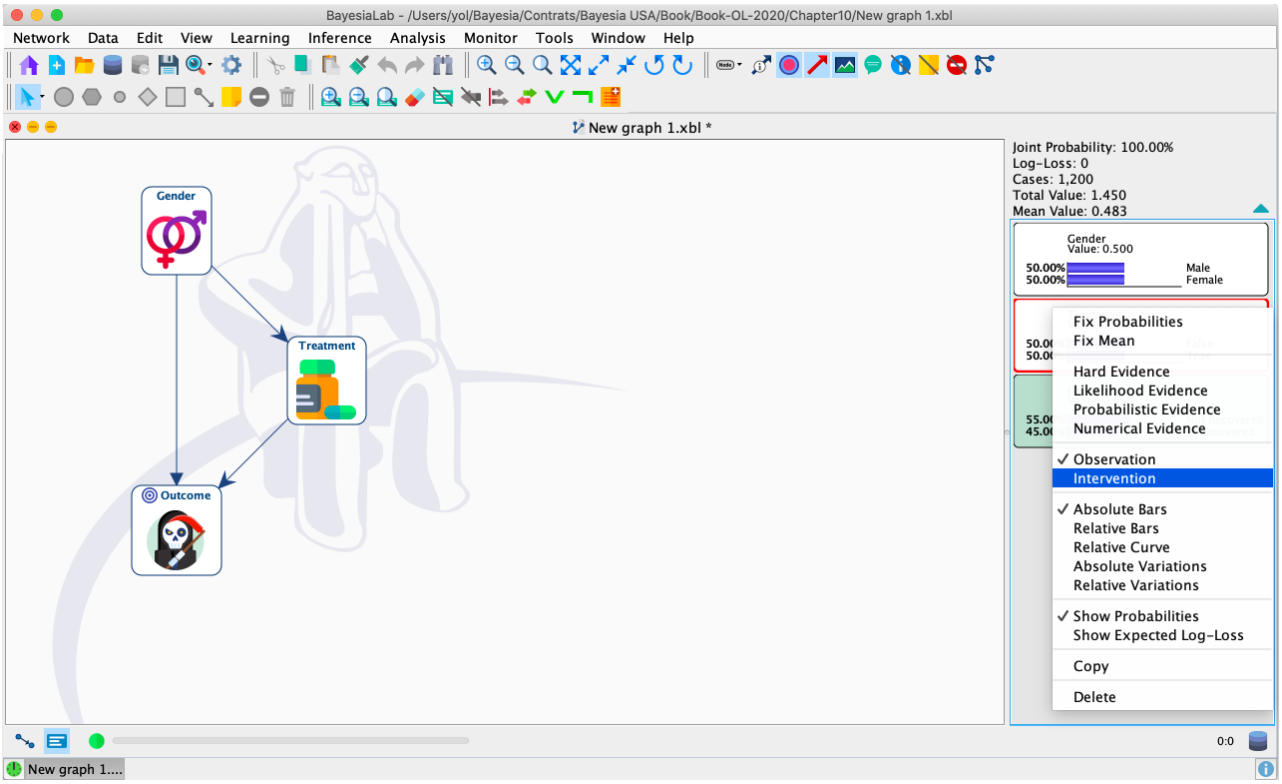


As expected, the target variable *Outcome* changes upon setting this evidence. However, *Gender* changes as well, even though we know that the treatment cannot possibly change the gender of a patient. What we observe here is a manifestation of the non-causal path: *Treatment* ← *Gender* → *Outcome*. These probabilities are obviously perfectly correct from the observational point of view: in the observed population of 1,200 individuals, three times as many men as women took the treatment.

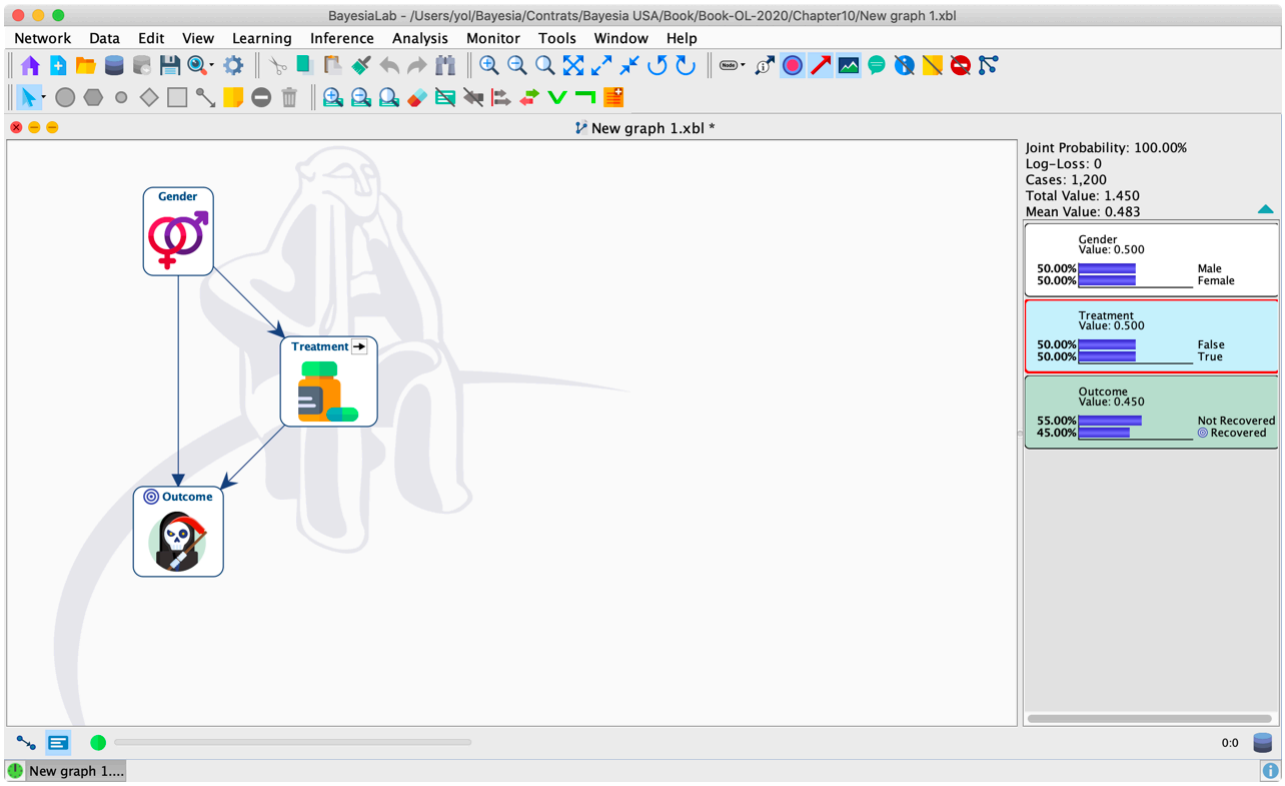
Causal Inference

For causal inference, however, we need a network that computes all probabilities under an intervention scenario. As we learned, Graph Surgery transforms the original causal network representing the *pre-intervention distribution* into a new, mutilated network that yields the *post-intervention* distribution.

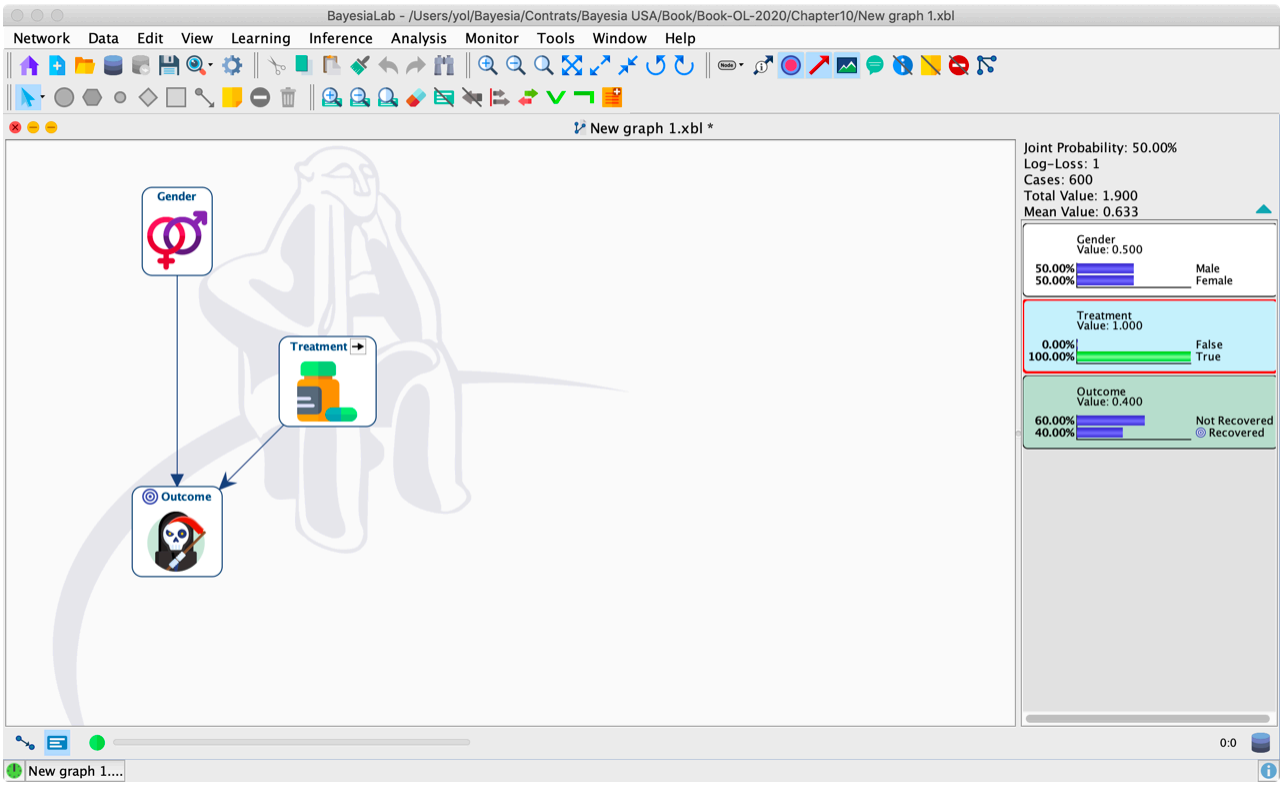
In BayesiaLab, Graph Surgery is automated. After right-clicking the Monitor of the node *Treatment*, we select `Intervention` from the Contextual Menu.



The activation of the Intervention Mode for this node is highlighted by the blue background of the *Treatment* Monitor and the arrow symbols (→) in the *Treatment* badge.



By double-clicking a state of *Treatment*, we now set an Intervention and no longer an Observation.

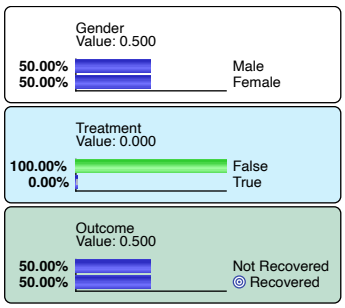


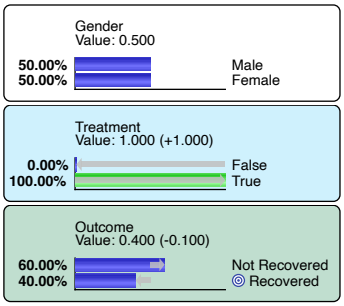
By intervening on *Treatment*, BayesiaLab applies Graph Surgery and removes the inbound arc into *Treatment*.

Recall the formula that computes the *Average Causal Effect (ACE)*:

$$ACE = P(Y = 1 \mid do(X = 1)) - P(Y = 1 \mid do(X = 0))$$

We can take it directly as a set of instructions and compare the probability of $Outcome = Recovered$ under $do(Treatment = False)$ and $do(Treatment = True)$. Note that the distribution of *Gender* remains the same pre- and post-intervention.





Thus, we obtain an *Average Causal Effect* of -0.1, which agrees with what we previously computed with the Adjustment Formula.

Causal Effect Estimation in BayesiaLab with Likelihood Matching

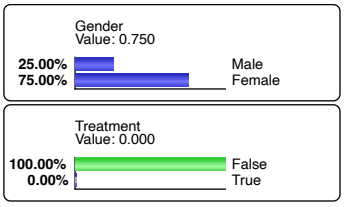
We now introduce Causal Effect Estimation by means of Likelihood Matching. Given the simplicity of Simpson's Paradox example, the need for yet another estimation method may not be immediately apparent. The advantages of Likelihood Matching will only become clear as we study a more complex domain, such as the marketing mix example of the next chapter. However, the current example makes it easy to explain Likelihood Matching.

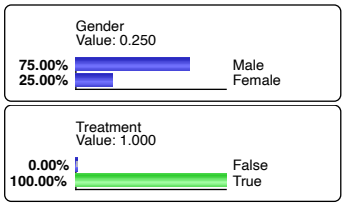
Intuition for Matching

In statistics, matching refers to the technique that makes confounder distributions of the treated and untreated sub-populations as similar as possible to each other. As such, applying matching to variables qualifies as adjustment, and we can use it with the objective of keeping causal paths open and blocking non-causal paths. In the Simpson's Paradox example, matching is fairly simple as we only need to match a single binary variable, i.e., *Gender*. That will meet our requirement for adjustment and block the only non-causal path in our model.

As our terminology of "blocking paths by matching" may not be understood outside the world of graphical models and Bayesian networks, we can offer a more intuitive interpretation of matching, which our example can illustrate very well.

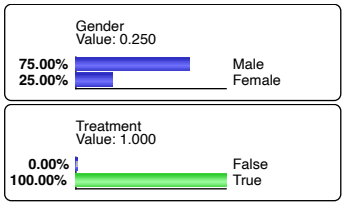
Because of the self-selection phenomenon of our population, *Gender* distribution is a function of *Treatment*. In other words, of those who are treated, 75% turn out to be male. Among untreated patients, only 25% are male.



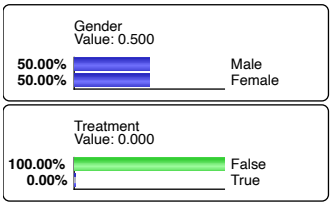


Given that we know that *Gender* has a causal effect on *Outcome* (men are more likely to recover than women, with or without treatment), and given this difference in gender composition, comparing the outcomes between the treated and non-treated patients is clearly not an apples-to-apples comparison.

We can propose a common-sense solution to this predicament. How about searching for a subset of patients within treated and non-treated groups, which have an identical gender mix, as illustrated below?



In statistical matching, this process typically involves the selection of units in such a way that comparable groups are created:



In practice, this can be more challenging as the observed units typically have more than just a single binary attribute. So, the idea of matching has to be extended to higher dimensions, and the observed units need to be matched on a range of attributes, including both continuous and discrete variables.

However, matching observations exactly with regard to all covariates is rarely feasible. For instance, patients are characterized by dozens or even hundreds of attributes and comorbidities. Finding two matching patients would be difficult enough, but finding populations with many matching pairs of patients would presumably be impossible.

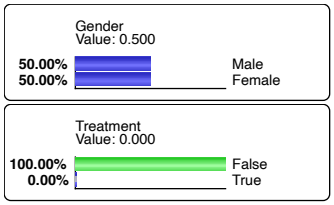
So, how does randomization do it? Actually, randomization does not guarantee identical populations but rather ensures that the distributions of confounders are balanced between the populations under study. So, to pursue balanced confounders instead of pursuing perfect matches, numerous similarity measures have been proposed for matching.

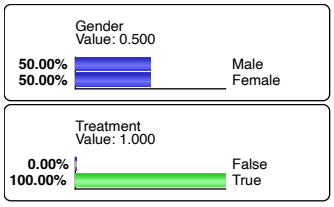
The concept of **Propensity Score Matching** has become a particularly popular method (Rubin and Rosenbaum, 1983). Instead of matching individuals on their high-dimensional attributes, we would match observations by their probability of treatment, i.e., $P(X = 1 \mid Z)$, which is known as the Propensity Score. Rubin and Rosenbaum have shown that matching on the propensity score achieves a balance of the covariate distributions.

However, matching on the Propensity Score requires the score itself to be estimated first. Conventional models only represent the outcome variable as a function of the treatment variable and the confounders, i.e., $P(Y \mid X, Z)$. If we need to understand the relationship between the treatment and the confounders, i.e., $P(X \mid Z)$, we have to estimate this separately. This usually means fitting a function, such as a regression, that models the propensity score, $PS = P(X \mid Z)$. For binary treatment variables, logistic regression is a common choice for the functional form.

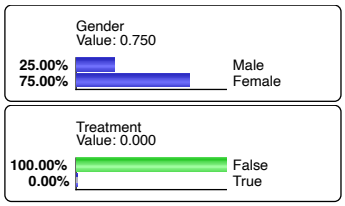
Likelihood Matching

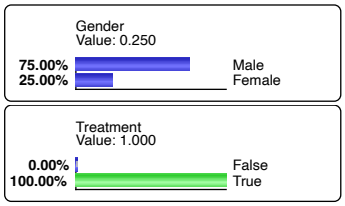
With BayesiaLab's Likelihood Matching, we do not directly match the underlying observations. Rather we match the distributions of the relevant nodes on the basis of the joint probability distribution represented by the Bayesian network. In our example, we need to ensure that the gender compositions of untreated and treated groups are the same, i.e., a 50/50 gender mix. This *theoretically ideal* condition is shown in the Monitors below.



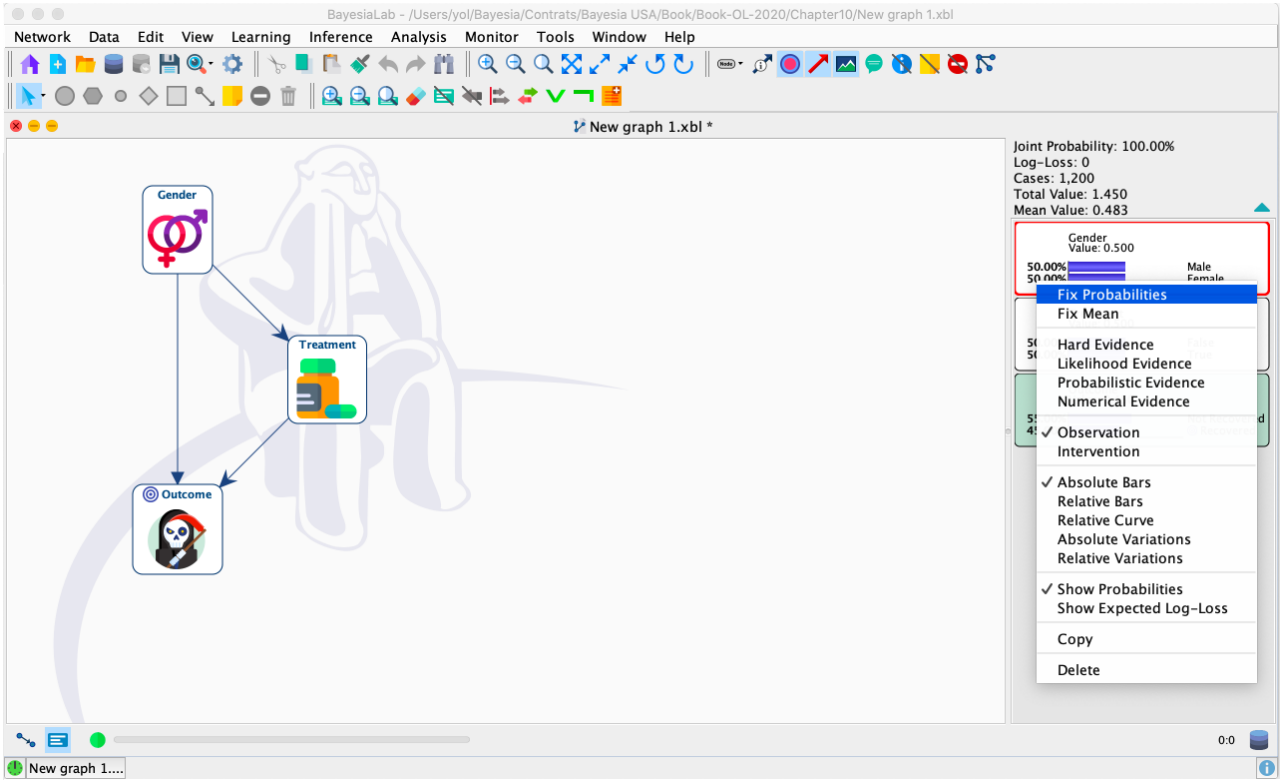


However, the actual distributions reveal the inequality of gender distributions for the untreated and the treated.

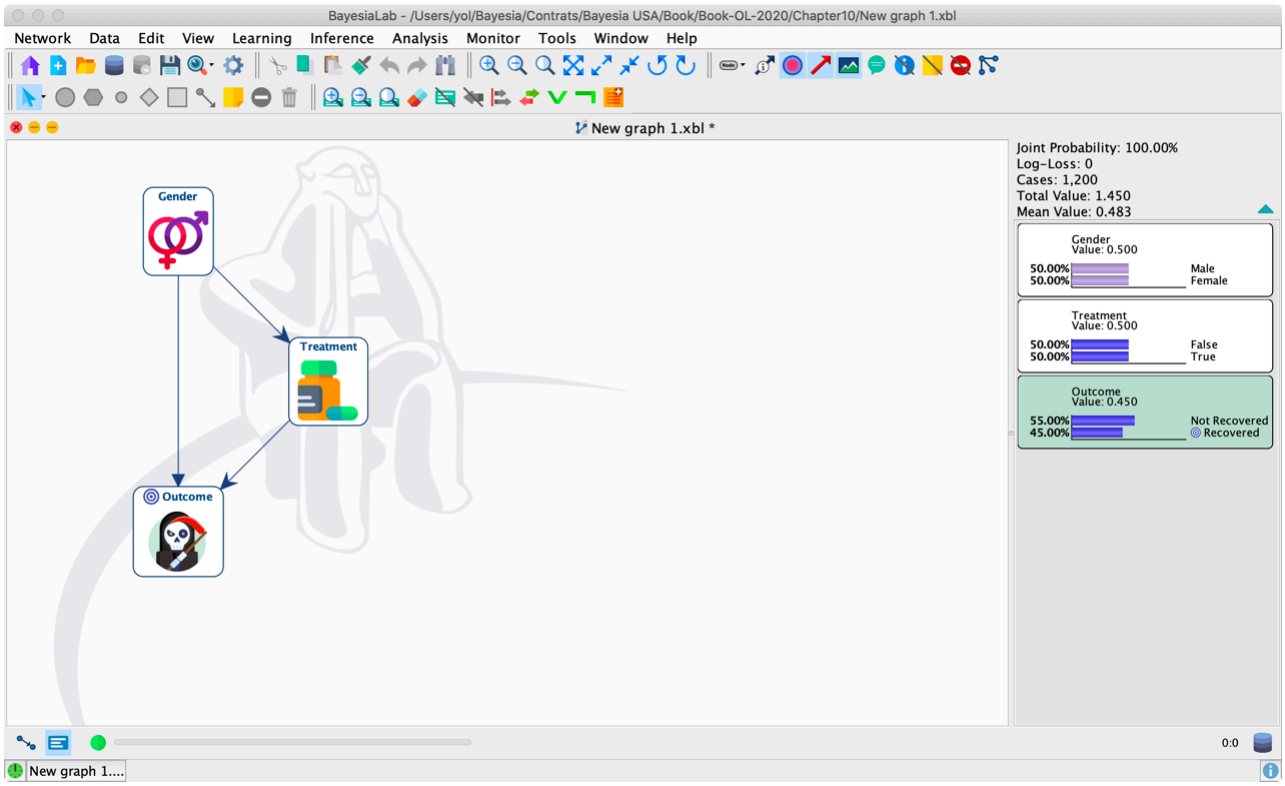




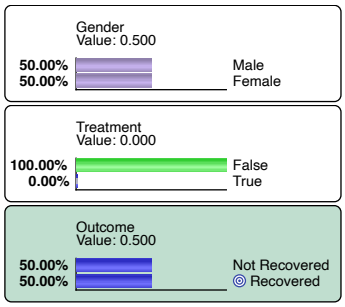
How can we overcome this? By simply using Probabilistic Evidence to set a 50/50 gender mix, i.e., the marginal distribution of *Gender*, upon setting evidence on *Treatment*. We can also right-click on the Monitor for *Gender* and select `Fix Probabilities` from the Context Menu. This will automatically use Probabilistic Evidence to set the current marginal distribution after each conditioning on *Treatment*, or any other node.

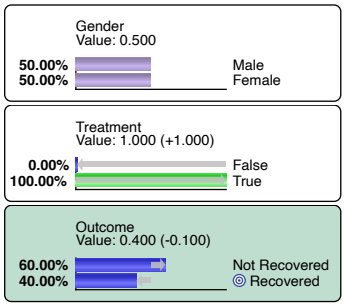


With Fix Probabilities applied, the distribution of *Gender* in its Monitor is highlighted in purple.



Other than colors, nothing appears to have changed. However, once we set the values *Treatment* = *False* and *Treatment* = *True*, we see that the distribution for *Gender* does not change. We can also observe that the corresponding posterior probabilities for *Outcome* are the same as those obtained with Graph Surgery.





With that, we can once again calculate the Average Causal Effect:

$$ACE = P\left(Y = 1 \mid do(X = 1)\right) - P\left(Y = 1 \mid do(X = 0)\right) = -0.1$$

For now, Likelihood Matching applied to Simpson's Paradox may not seem like a breakthrough method. Conceptually and practically, it appears to be another form of adjustment. The fundamental advantages of Likelihood Matching will become clear in the context of the next chapter, Causality and Optimization.

Graph Surgery

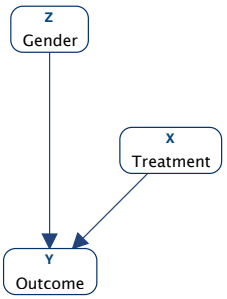
Let us summarize what we have so far: First, we have observational data from our domain. Second, we developed a theory about the DGP, i.e., the causal relationships in the domain. Both together will serve as the basis for estimating the causal effect. Before we do that, we should contemplate a very literal interpretation of these causal relationships.

If this causal graph is a correct representation of how the domain works, then every relationship between a pair of variables holds independently. Thus, the causal graph represents autonomous relationships between parent nodes and child nodes. It is as if each node were "listening for instructions" from its parents and only from its parents: the child node's values are solely determined by the value of its parents, not of any other nodes in the system. Also, these relationships remain invariant regardless of any values that other nodes take on.

Let us now consider an outside intervention on X. Thus, rather than "listening" to its parent Z, X is now entirely determined by an external force and set to specific values, e.g., $X = 1$ or $X = 0$. This external intervention breaks the natural relationship between X and Z. Thus, Z no longer influences X. However, $Z \to Y$ and $X \to Y$ remain unaffected, and the original "natural" values of Z are not affected either.

What is the significance of all this? The idea is that intervening on X is like trying out, or simulating, what would happen if treatment were to be applied universally to the entire population — or withheld universally. Isn't this the causal effect we are interested in? In other words, computing the causal effect is like simulating outside interventions on the treatment variable X.

How does this help us? By simulating an intervention, we "mutilate" the graph. This new graph looks like we had severed the arc going into the treatment variable X. This operation is what Judea Pearl has rather colorfully named "graph surgery" or "graph mutilation."



Note that the mutilated graph achieves what was stipulated by the Adjustment Criterion: the non-causal path → $X \leftarrow Z \rightarrow Y$ does not exist any longer, as required by the Adjustment Criterion, and, given the autonomy of the other arcs, the causal path → $X \rightarrow Y$ remains unblocked.

Applying Graph Surgery allows us to transform a causal graph that represents a joint probability distribution P of observational data, i.e., pre-intervention distribution, into a new mutilated graph that represents the joint probability distribution P_m of the same variables under a simulated intervention, i.e., post-intervention distribution.

This new graph can tell us what happens to Y when we intervene and set X to a specific value, i.e., $P(Y = y \mid do(X = x))$. Note the do-operator! With this mutilated graph, we can compute the quantity of interest, the Average Causal Effect (ACE):

$$ACE = P\left(Y = 1 \mid do(X = 1)\right) - P\left(Y = 1 \mid do(X = 0)\right)$$

where

$$P(Y = y \mid do(X = x)) = P_m(Y = y \mid X = x)$$

Graphical Identification Criteria

It is self-evident that causal arcs have implications in terms of causation. However, as we pointed out earlier in this chapter (see Structures Within a DAG), there are also implications regarding the association of variables. This will perhaps become clearer as we introduce the concepts of "causal path" and "non-causal path."

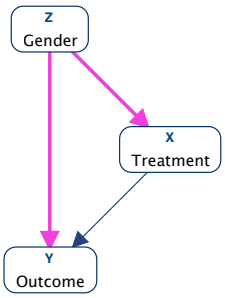
Causal and Non-Causal Paths

In a DAG, a path is a sequence of non-intersecting, adjacent arcs, regardless of their direction.

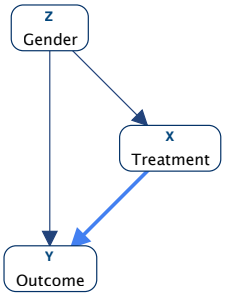
- A causal path can be any path from cause to effect, in which all arcs are directed away from the cause and pointed toward the effect.
- A non-causal path can be any path between cause and effect in which at least one of the arcs is oriented from effect to cause.

Our example contains both types of paths:

Non-Causal Path: $X \leftarrow Z \rightarrow Y$ (→)



Causal Path: $X \to Y$ (➡)



This distinction between causal and non-causal paths is critically important for identification.

Adjustment Criterion and Identification

The Adjustment Criterion (Shpitser et al., 2010) is perhaps the most intuitive among several graphical identification criteria. The Adjustment Criterion states that a causal effect is identified if we can adjust for a set of variables such that:

- All non-causal paths (➡) between treatment and outcome are "blocked" (non-causal relationships prevented).
- All causal paths (➡) from treatment to outcome remain "open" (causal relationships preserved).

What does "adjust for" mean in practice? "Adjusting for a variable" can stand for any of the following operations, which all introduce information on a variable:

- Controlling
- Conditioning
- Stratifying
- Matching

At this point, the adjustment technique is irrelevant. Rather, we only need to determine which variables, if any, need to be adjusted in order to block the non-causal paths while keeping the causal paths open. Revisiting both paths in our CDAG, we can now examine which ones are open or blocked:

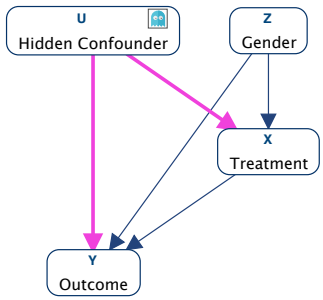
- First, we look at the non-causal path (➡) in our CDAG: $X \leftarrow Z \rightarrow Y$. This implies that there is an indirect association between X and Y via Z that has to be blocked by adjusting for Z.
- Next is the causal path (➡) in our CDAG: $X \rightarrow Y$. It consists of a single arc from X to Y, which is open by default and cannot be blocked.

In this example, the Adjustment Criterion can be met by blocking the non-causal path $X \leftarrow Z \rightarrow Y$ by means of adjusting for Z. In other words, adjusting for Z allows identifying the causal effect from X to Y. From now on, we will often refer to such variables Z as Confounders.

Readers may be familiar with the expression "controlling for confounders." What is important to bear in mind is that not all covariates in a system are Confounders! Recall Judea Pearl's warning about ignorability and the risk of treating every covariate as a Confounder (see Ignorability).

Unobserved Confounders

Thus far, we have assumed that our example has no unobserved (also called hidden or latent) variables. However, if we had reason to believe that there is another variable, U, which appears to be relevant on theoretical grounds but was not recorded in the dataset, identification could no longer be possible. Why? Let us assume U is a hidden common cause of X and Y. By adding this unobserved variable U, a new non-causal path appears between X and Y via U.



Given that U is hidden, there is no way to adjust for it, and, therefore, we have an open, non-causal path that cannot be blocked. Hence, the causal effect is no longer identifiable, and thus, it can no longer be estimated without bias.
This highlights how easily identification can be "ruined." Once again, we can only justify the absence of unobserved variables on theoretical grounds.

Example: Augmented Simpson's Paradox

Now that we have seen how to estimate the Average Causal Effect by manually interacting with the BayesiaLab's Monitors, with both Graph Surgery and Likelihood Matching, we will use the

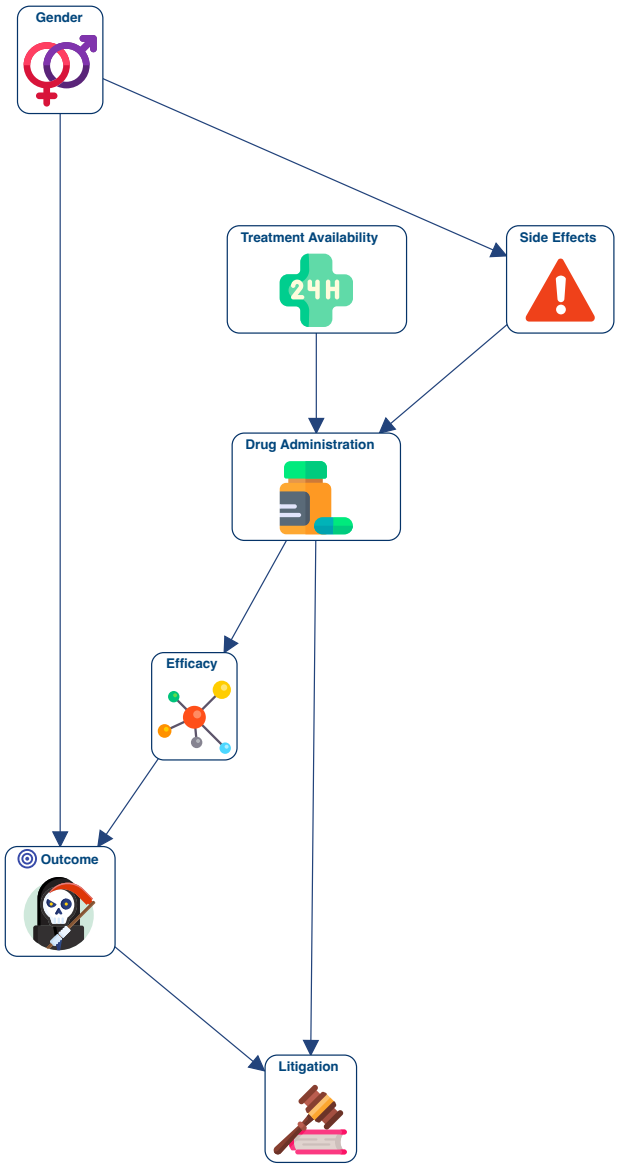
BayesiaLab's Direct and Total Effect functions to compute causal effects automatically for a set of variables. But first, we present a slightly more complex version of Simpson's Paradox to illustrate these features (see Example: Simpson's Paradox).

Augmented Simpson's Paradox

Our updated story contains four additional dimensions:

- Treatment Availability: the treatment is not always available;
- Side Effects: the treatment may produce severe side effects;
- *Efficacy*: some patients do not respond to the drug;
- Litigation: the families of patients who died may sue the pharmaceutical company that had provided the treatment.

The manually designed CDAG shown below describes this new domain qualitatively:



Next, we describe the quantitative part of the domain. First, we state that Treatment Availability is 75%. We also assume that the treatment may have Side Effects, which are much more frequent for females. The following conditional probability table quantifies this direct causal dependency on Gender:

	Side Effects	
Gender	FALSE	TRUE
Male	75%	25%
Female	25%	75%

Patients decide whether or not to take the treatment based on two criteria, Treatment Availability and Side Effects. The dependencies are described in the following table. It states that if the treatment

is unavailable, patients cannot have the treatment, which is deterministic (and obvious). However, if the treatment is available, those patients who do not have any risk of experiencing side effects will always choose the treatment, while those at risk will be unlikely to submit to the treatment:

		Drug Administration	
Treatment Availability	**Side Effects**	FALSE	TRUE
FALSE	FALSE	100%	0%
	TRUE	100%	0%
TRUE	FALSE	0%	100%
	TRUE	95%	5%

Furthermore, the *Efficacy* of the treatment depends on *Drug Administration* plus some hidden factors that render the treatment ineffective in 20% of patients:

	Efficacy	
Drug Administration	FALSE	TRUE
FALSE	100%	0%
TRUE	20%	80%

The Target Node, *Outcome*, is defined by Gender and *Efficacy*. In this context, "not recovered" means that the patient died — hence the grim illustration attached to the icon.

		Outcome	
Gender	**Efficacy**	Not Recovered	Recovered
Male	FALSE	30%	70%
	TRUE	40%	60%
Female	FALSE	70%	30%
	TRUE	80%	20%

Finally, half of the families of those patients who took the treatment and died are pursuing litigation. More specifically, these families are suing the pharmaceutical company that provided the treatment.

		Litigation	
Drug Administration	**Outcome**	FALSE	TRUE
FALSE	Not Recovered	100%	0%
	Recovered	100%	0%
TRUE	Not Recovered	50%	50%
	Recovered	100%	0%

Path Analysis

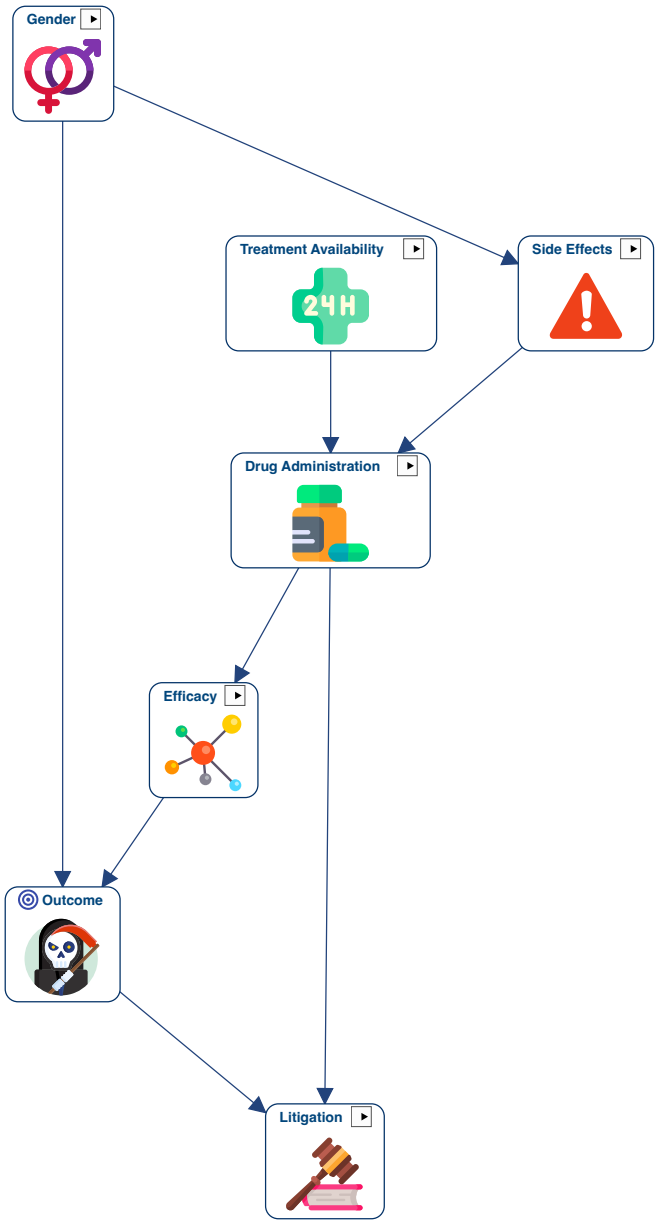
We now list the paths between each variable and the target variable *Outcome* by using `Menu > Analysis > Visual > Graph > Influence Paths to Target`. The causal paths are highlighted in blue (→), and the non-causal paths (i.e., paths with at least one "backward" arrow ←) are shown in pink (→):

1	Gender → Outcome
2	Gender → Side Effects → Drug Administration → Efficacy → Outcome
3	Treatment Availability → Drug Administration → Efficacy → Outcome
4	Side Effects ← Gender → Outcome
5	Side Effects → Drug Administration → Efficacy → Outcome
6	Drug Administration ← Side Effects ← Gender → Outcome
7	Drug Administration → Efficacy → Outcome
8	Efficacy ← Drug Administration ← Side Effects ← Gender → Outcome
9	Efficacy → Outcome
10	Litigation ← Drug Administration ← Side Effects ← Gender → Outcome
11	Litigation ← Drug Administration → Efficacy → Outcome
12	Litigation ← Outcome

Recall the Adjustment Criterion, which stipulates that we must keep all of a variable's causal paths to the target variable open and simultaneously block all its non-causal paths for estimating its causal effect.

Graph Surgery

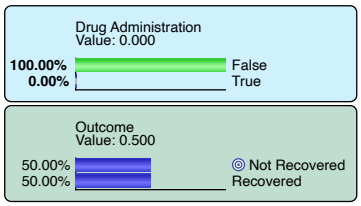
To illustrate BayesiaLab's Total and Direct Effects functions with Graph Surgery, we set all nodes but *Outcome* to Intervention Mode.

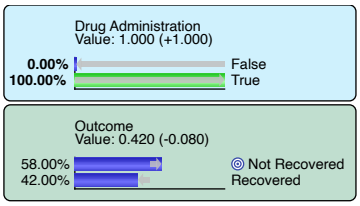


Notice the arrow symbols (→) in the badges of the nodes that are set to Intervention Mode.

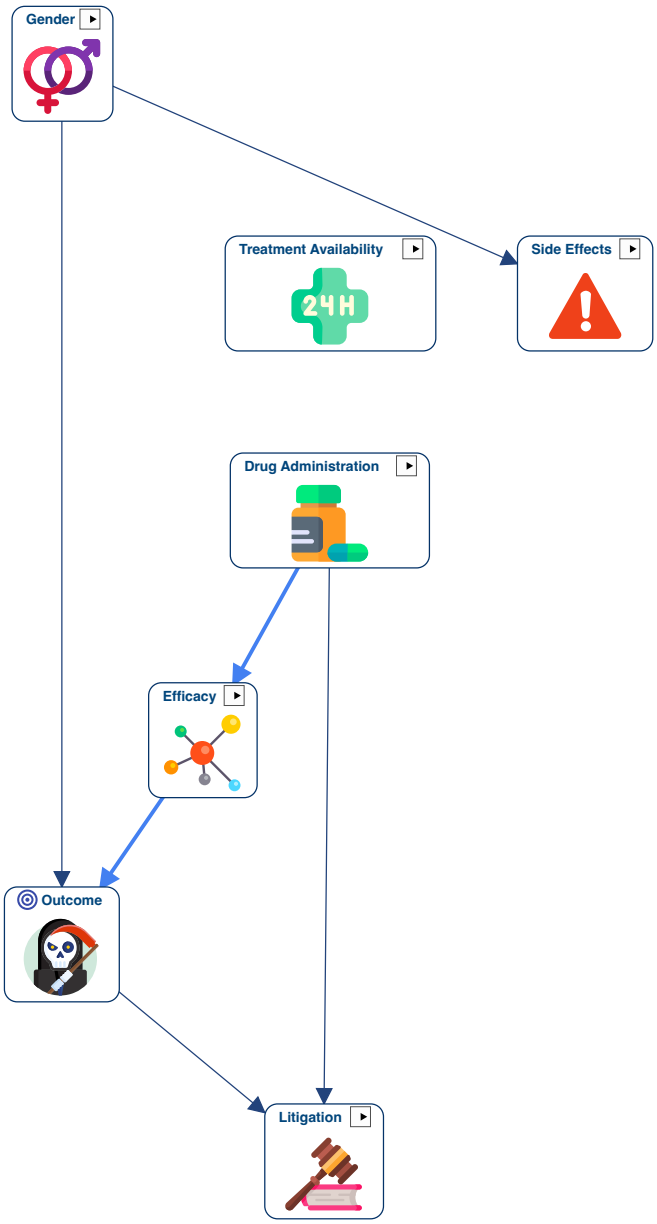
Average Causal Effect

Before using BayesiaLab's automated tools for computing causal effects, we manually estimated the causal effect of our main variable of interest, *Drug Administration*, by using the Monitors.





Setting a piece of Evidence in Intervention Mode simulates an intervention on *Drug Administration* and mutilates the graph, as shown below, which meets the Adjustment Criterion by blocking the non-causal path (cf. Path Analysis, #6).

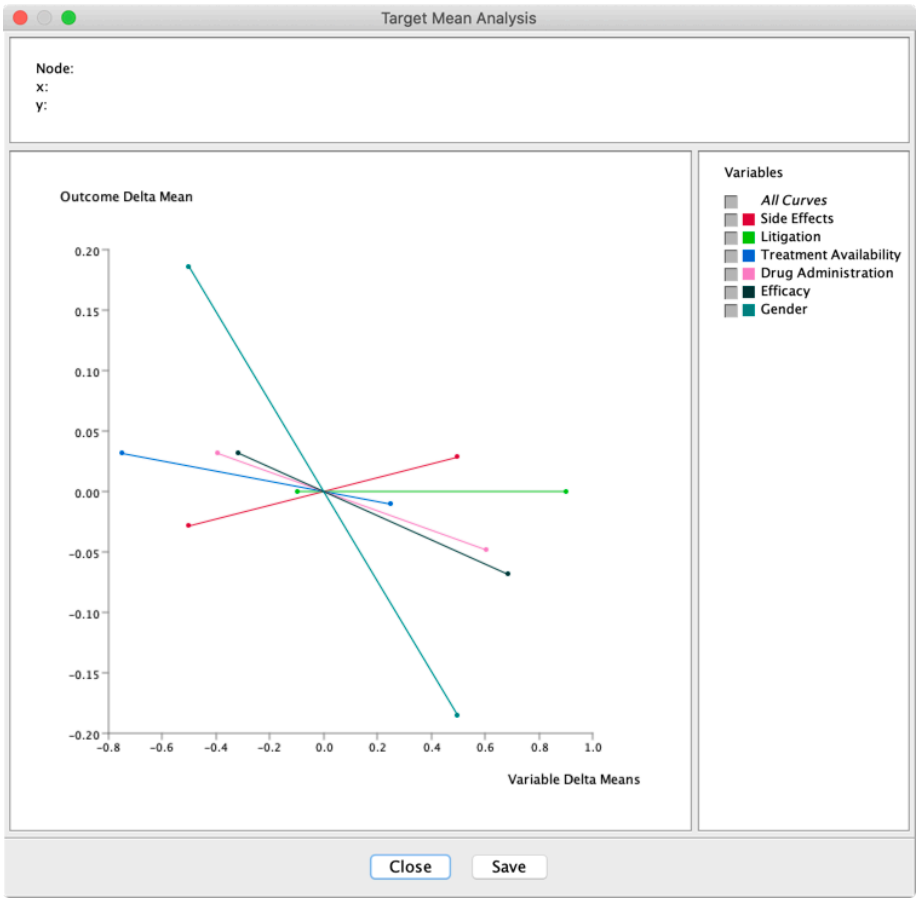


The Average Causal Effect of *Drug Administration* on *Outcome*, mediated by *Efficacy*, is -0.08.

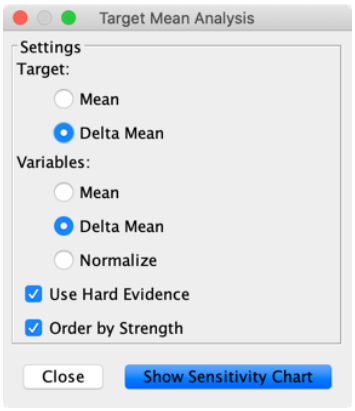
Total Effects

We have seen in Chapter 8 that BayesiaLab estimates Total Effects as the derivatives of Total Effect Curves. These curves are based on the Posterior Mean Values of the Target Node given Mean Values from the interval of the variable under study. While the variables are in Intervention Mode, the Posterior Mean Values are computed based on the mutilated graph.

We can plot these curves with `Main Menu > Analysis > Visual > Target > Target's Posterior > Curves > Total Effects`.



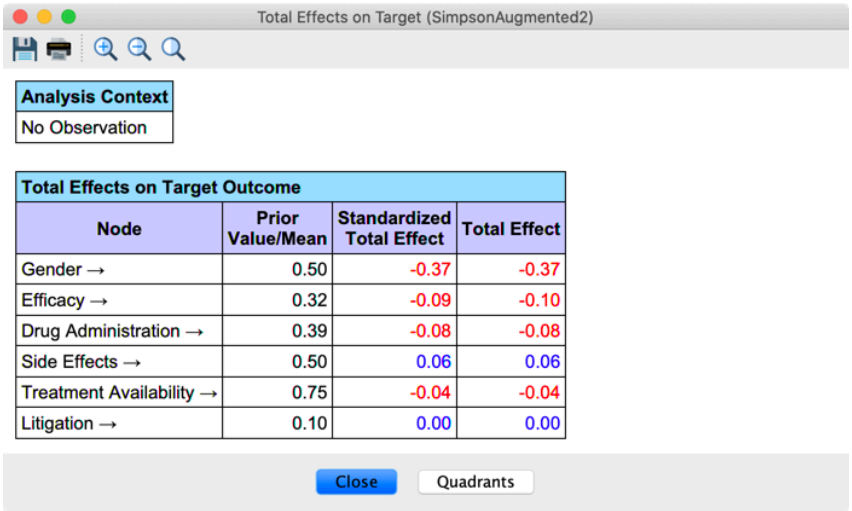
For generating this graph, we can set a number of options:



The x-axis, Variable Delta Means, represents the difference between the Mean Value generated for the analysis (here, Hard Evidence/Intervention on the states of the variable under study) and its Prior Mean Value. The y-axis represents the difference between the PosteriorMean Value of *Outcome* and its PriorMean Value.

> If we do not specifically associate numerical values with symbolic states, BayesiaLab uses the state index. In our example,
>
> - False is 0, and True is 1.
> - Male is 0, and Female is 1.
> - Not Recovered is 0, and Recovered is 1.

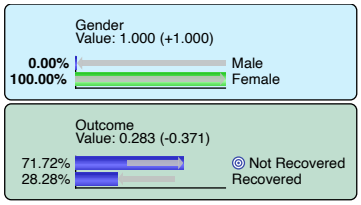
We see that Side Effects is the only variable with a positive causal effect. We also notice that Litigation has no causal effect. Given that all variables are binary, the corresponding curves are linear. Therefore, the curves' derivatives will be perfect summaries of the Total Effect Curves: `Menu > Analysis > Report > Target > Total Effects on Target`:



The Total Effect is the derivative computed at (0, 0) in the previous Target Mean Analysis graph, i.e., the slope of the curve. The Standardized Total Effect is the Total Effect times the ratio between the standard deviation of the variable and the standard deviation of the Target Node.

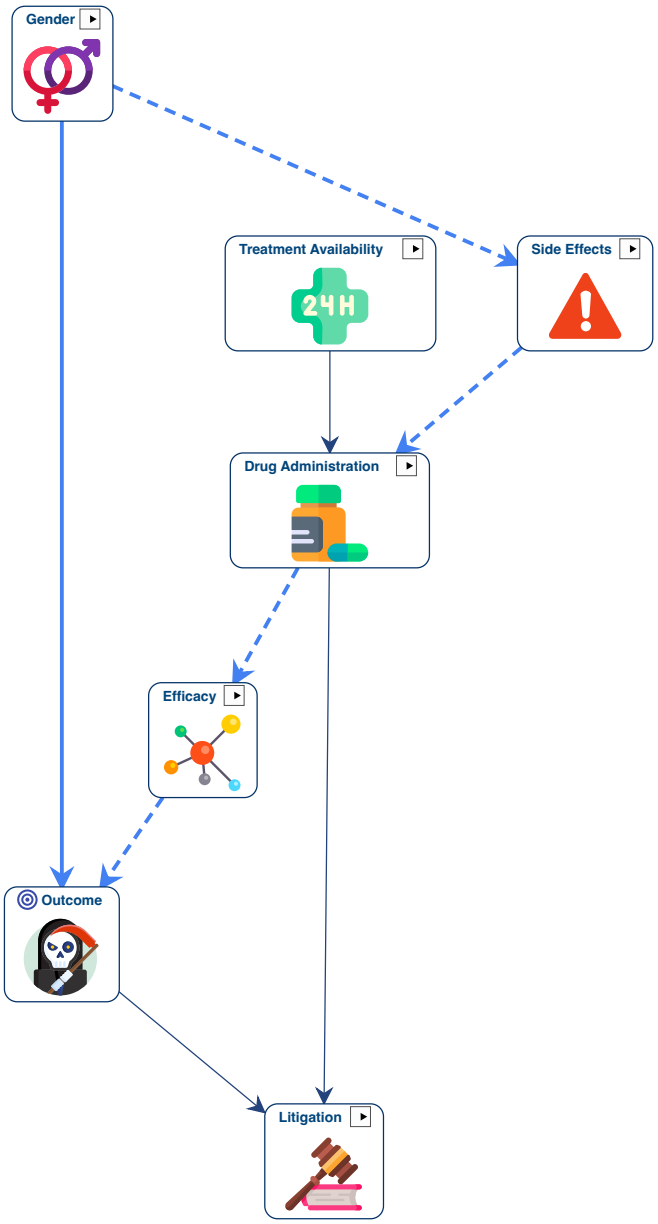
The arrow symbols (→) in the results table indicate that Intervention Mode was active on all nodes, triggering Graph Surgery upon each observation/intervention during the estimation of the effects.

Gender is the variable with the strongest Total Effect. It is negative because of the index values of the states. Females (1) are recovering at a lower rate than Males (0).



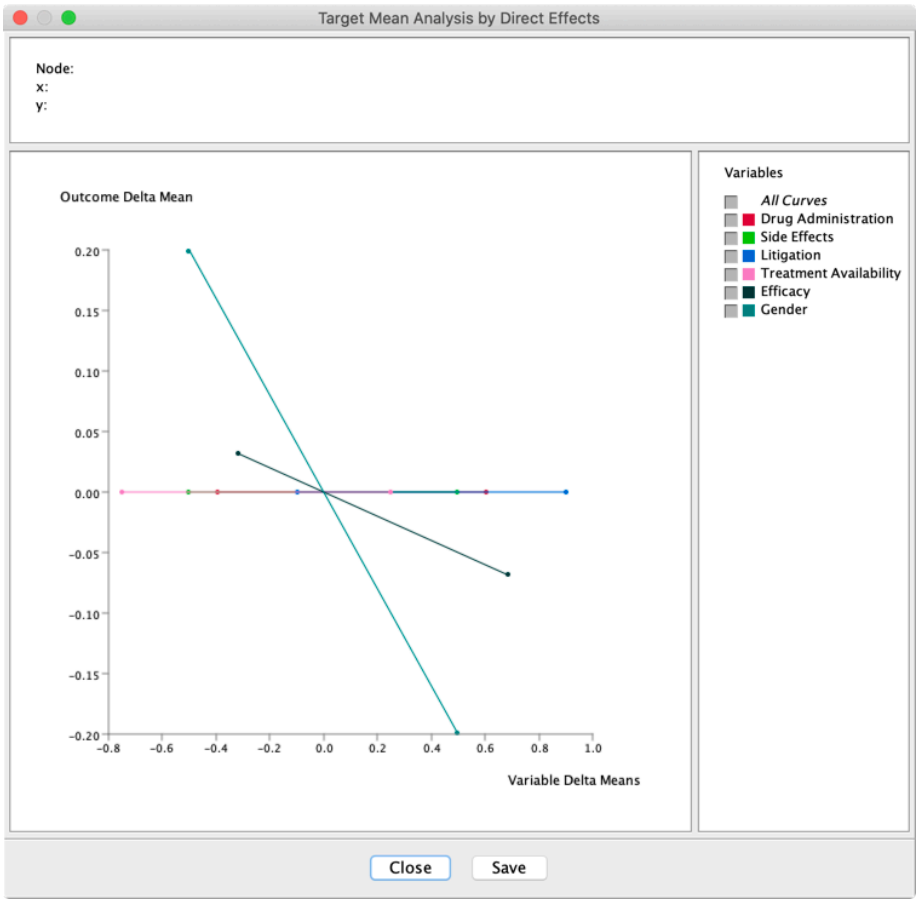
Note that there are two paths from Gender to *Outcome* (paths #1 and #2 illustrated in the previous section), and they are both causal. Gender is indeed a root node, i.e., it has no parents, meaning the Adjustment Criterion is fulfilled by default.

The Total Effect measures the effects of these two causal paths: the direct path (#1) and the indirect path (#2), represented by the dashed blue arcs below.

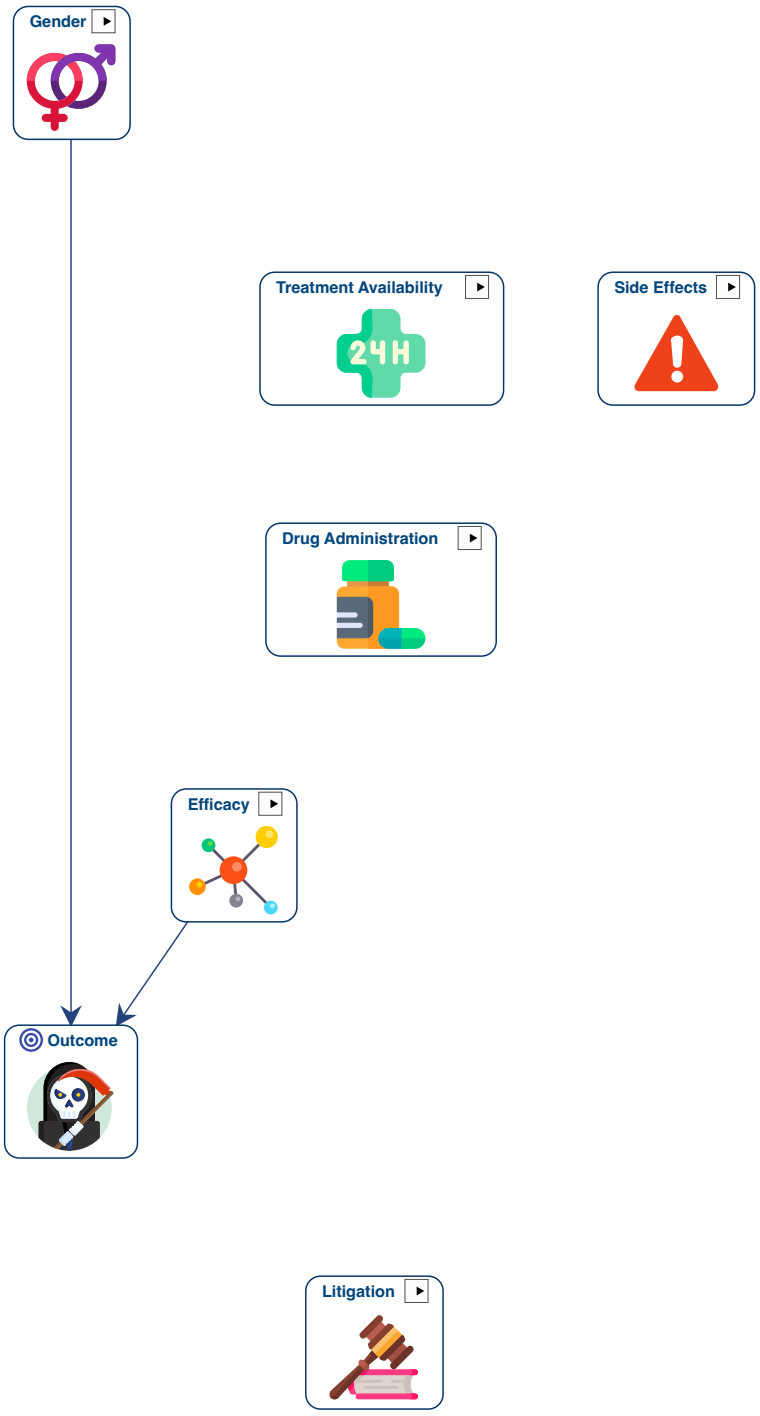


Direct Effects

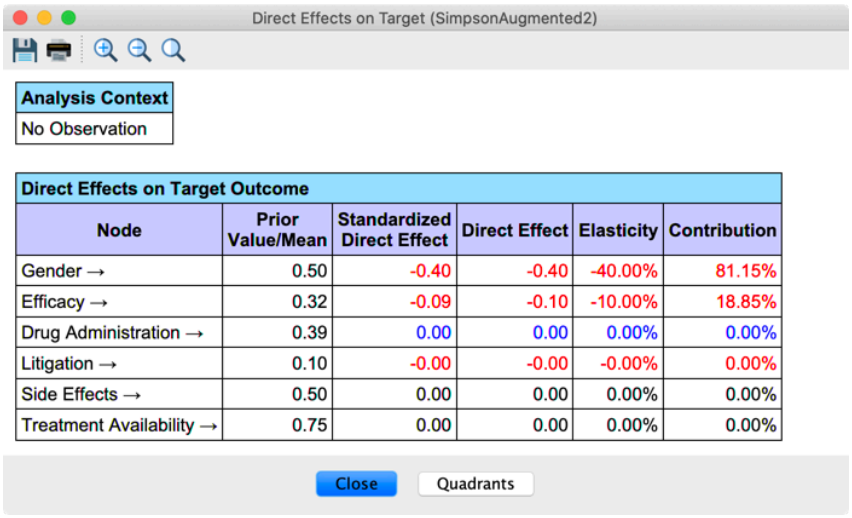
Now suppose we are interested in estimating the effect of the direct paths only. This would require blocking not only the non-causal paths but also the indirect causal paths. This is the role of BayesiaLab's Direct Effect functions. The only difference between Direct and Total Effect functions is that, by default, all other nodes are held constant during the estimation of the variable's Direct Effect. We generate the Direct Effect Curves with `Menu > Analysis > Visual > Target > Target's Posterior > Curves > Direct Effects`, using the same parameters as those previously used for Total Effects:



Given that all nodes are in Intervention Mode, the only variables with Direct Effects are the Parents of *Outcome*. Indeed, intervening on all nodes to hold them constant triggers Graph Surgery and generates the mutilated graph below:



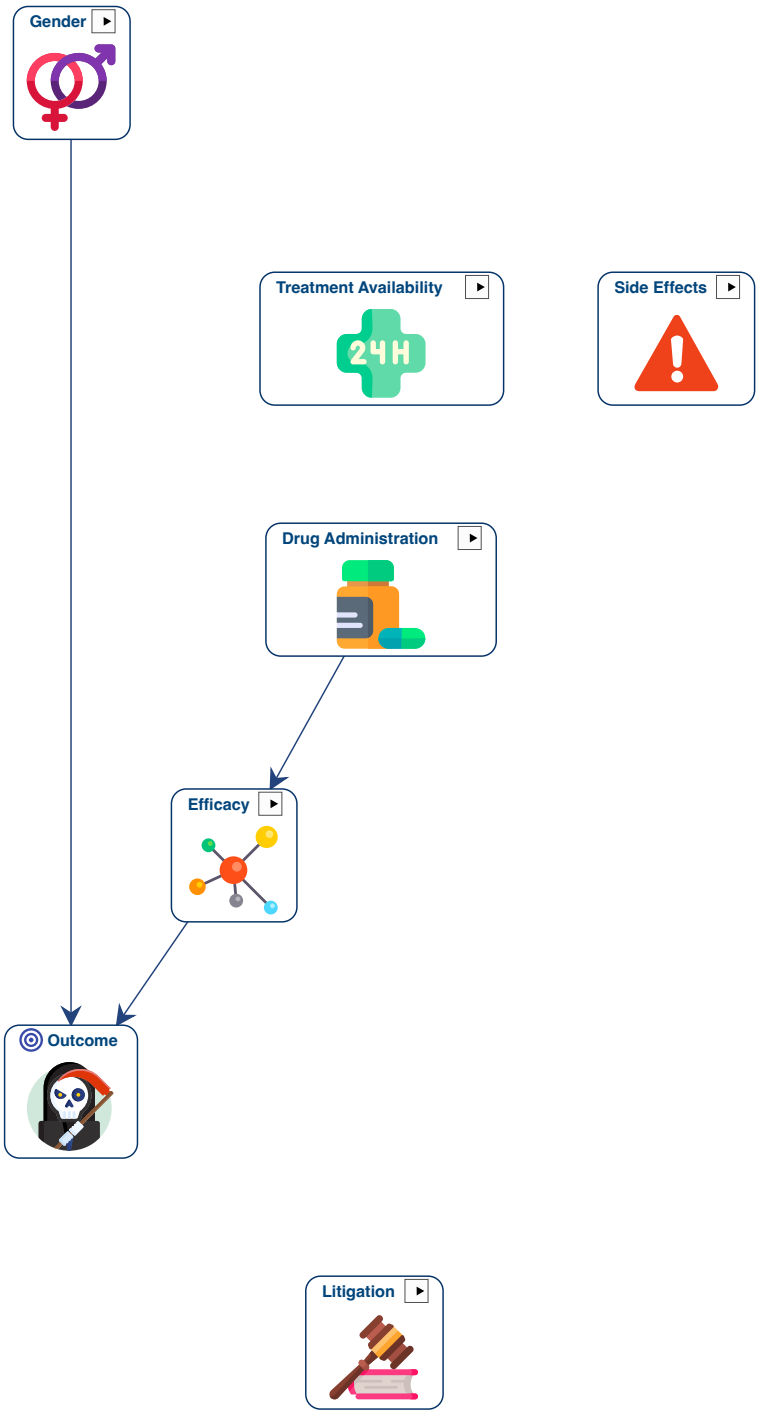
The function `Menu > Analysis > Report > Target > Direct Effects on Target` allows us to compute the Direct Effects, the single-point estimates of these curves:

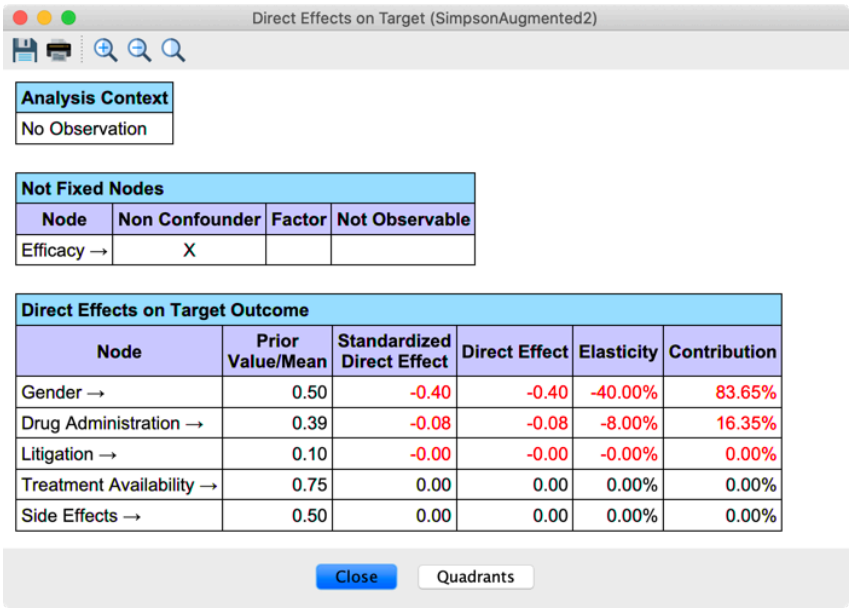


- The Direct Effect is the slope of the Direct Effect Curve between the endpoints of the variable interval.
- The Standardized Direct Effect is the Direct Effect times the ratio between the standard deviation of the variables and the standard deviation of the Target Node.
- The Elasticity is the Direct Effect times the ratio between the range of the variable and the range of the Target Node.
- The Contribution is the Standardized Direct Effect divided by the total sum of Standardized Direct Effects.

Non-Confounders

By default, BayesiaLab's Direct Effect functions measure a variable's effect by holding all other variables constant. However, we can use the predefined class *Non_Confounder* to define the nodes we *do not* want to control. In our example, the main variable of interest, *Drug Administration*, has no direct effect. The post-treatment variable *Efficacy* mediates its causal effect, and the Direct Effect analysis blocks the path. We must therefore use the predefined class Non_Confounder (*Efficacy*'s `Context Menu > Properties > Classes > Add > Predefined Class > Non_Confounders`) to prevent BayesiaLab from holding *Efficacy* constant and allow the estimation of the mediated causal effect. The new mutilated graph below is then used for estimating the Direct Effects:


Gender
Treatment Availability
24H
Side Effects
Drug Administration
Efficacy
Outcome
Litigation



Drug Administration's Direct Effect now equals the Average Causal Effect we manually computed with the Monitors. You can also note that we no longer analyze the effect of the Non-Confounder *Efficacy*.

Likelihood Matching

Now suppose we want to use Likelihood Matching instead of Graph Surgery. We first set back all nodes in Observation Mode via the monitors' Contextual Menus.

Nodes of Interest: Treatments/Drivers

The nodes of interest are the nodes for which we want to estimate the causal effect on the Target Node. We call them Treatments or Drivers.

In the previous section, we assumed that all nodes were of interest and set them in Intervention Mode. With Likelihood Matching, the workflow is less straightforward. For each Driver, we need to analyze the paths to the Target (*cf.* Path Analysis) to define the set of nodes that need to be controlled for to block the non-causal paths and let the causal paths open. Note that these nodes' sets may differ for each Driver, requiring us to perform multiple Total Effect analyses to avoid conflicting adjustments. The first step is then to define our nodes of interest. In the Augmented Simpson Paradox, the main variable of interest is obviously *Drug Administration*, but for illustrative purposes, let us consider Gender as well.

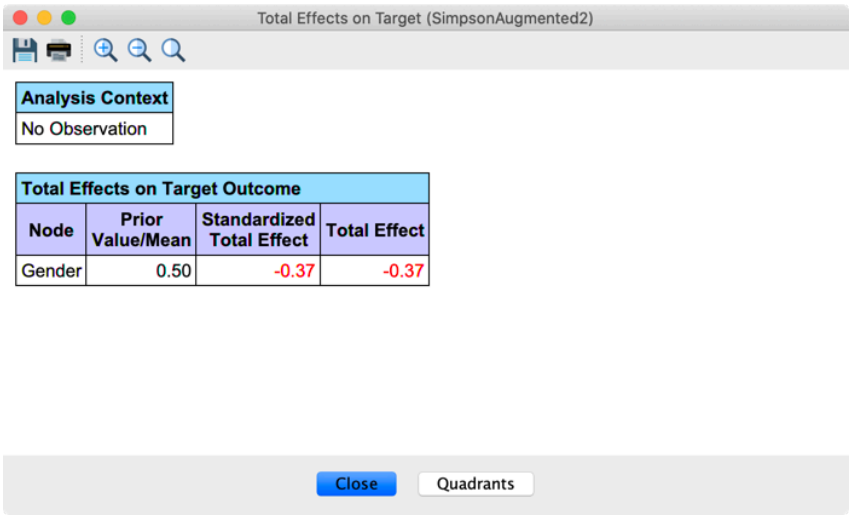
Total Effects

We have seen in the Path Analysis section that there are two paths between Gender and *Outcome*, both causal (#1 and #2). Thus, there is no variable to adjust for to estimate the Total Effect.

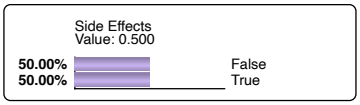
The Path Analysis indicates that there are also two paths between *Drug Administration* and *Outcome*, one causal (#7) and one non-causal (#6): *Drug Administration* ← Side Effects ← Gender → *Outcome*.

So we need to adjust for Side Effects, or for Gender, to block this path. This is in contradiction to the analysis of Gender's effect. We cannot estimate the Total Effects of Gender and *Drug Administration* in the same analysis with Likelihood Matching!

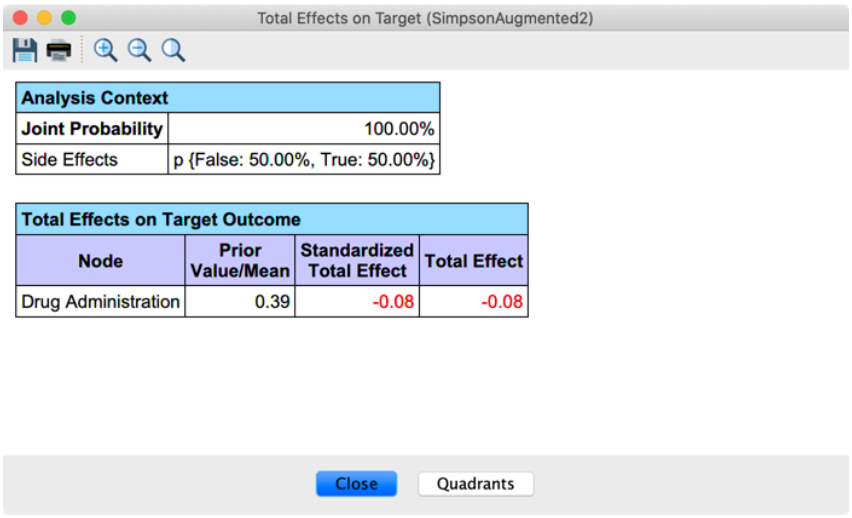
So let us start with Gender. We select the node, go to `Menu > Analysis > Report > Target > Total Effects on Target`, and confirm that we want to perform the analysis on the selected node only:



For *Drug Administration*, let us suppose we choose to adjust for Side Effects. We right-click on its associated Monitor and select `Fix Probabilities` from its Contextual Menu.



Then, we select the node *Drug Administration*, go to `Analysis > Report > Target > Total Effects on Target`, and confirm that we want to perform the analysis on the selected node only:



Direct Effects

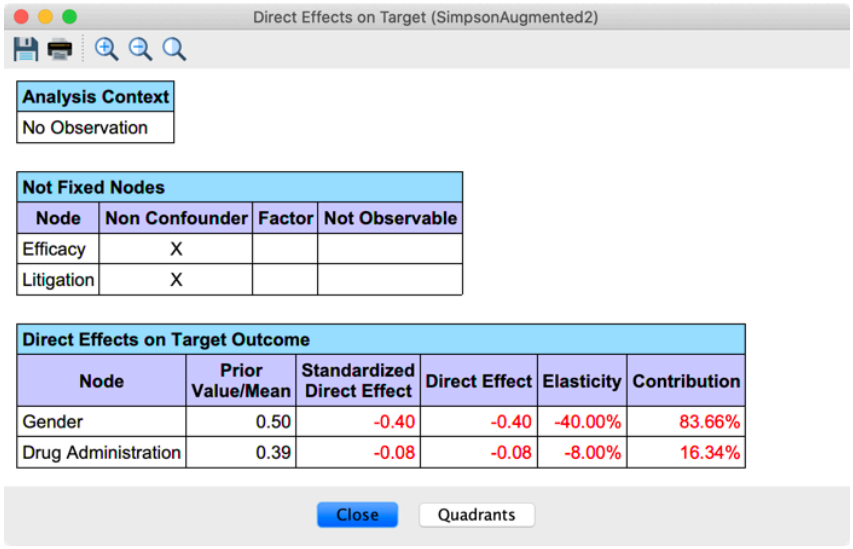
Now let us look at the workflow for estimating Direct Effects with Likelihood Matching, i.e., how to assess the effects of the direct paths only. Remember that, by default, BayesiaLab's Direct Effect functions measure a variable's effect by holding all variables constant except those associated with the predefined class Non_Confounder. Holding a variable constant with Graph Surgery implies the deletion of its entering arcs. Thus, there is no risk of biasing the estimation of Direct Effects. In the Likelihood Matching case, this risk exists because we set evidence on the variable to adjust for it. Indeed, controlling for descendants of the Target Node (e.g., Litigation) automatically biases the estimate.

While we previously added *Efficacy* to the Non_Confounder class to let it mediate the effect of *Drug Administration*, we must also add Litigation to prevent its adjustment.

Notice that there is no conflict in this analysis:

- Gender
  - Controlling for Side Effects, *Drug Administration* allows to cut the indirect causal path (#2);
  - Controlling for *Treatment Availability* has no impact;
  - Not controlling for *Efficacy* has no effect as path #2 is already blocked;
  - Not controlling for Litigation prevents to bias the estimation of the effect;
- *Drug Administration*
  - Controlling for Side Effects, Gender allows to cut the non-causal path (#4);
  - Controlling for *Treatment Availability* has no impact;
  - Not controlling for *Efficacy* allows to let the information flows from *Drug Administration* to *Outcome*;
  - Not controlling for Litigation prevents to bias the estimation of the effect.

We can, therefore, select our two nodes of interest, use `Analysis > Report > Target > Direct Effects` on Target, and confirm that we want to perform the analysis on the selected nodes only:



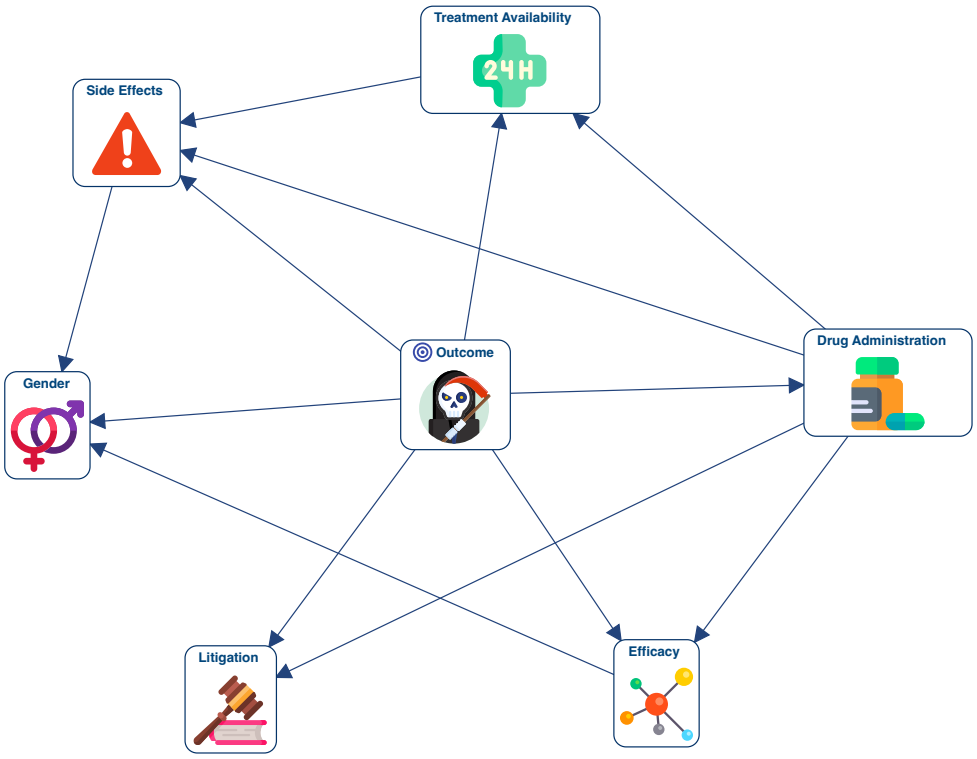
Graph Surgery versus Likelihood Matching

Before concluding this chapter, let us summarize the main characteristics of Graph Surgery and Likelihood Matching:

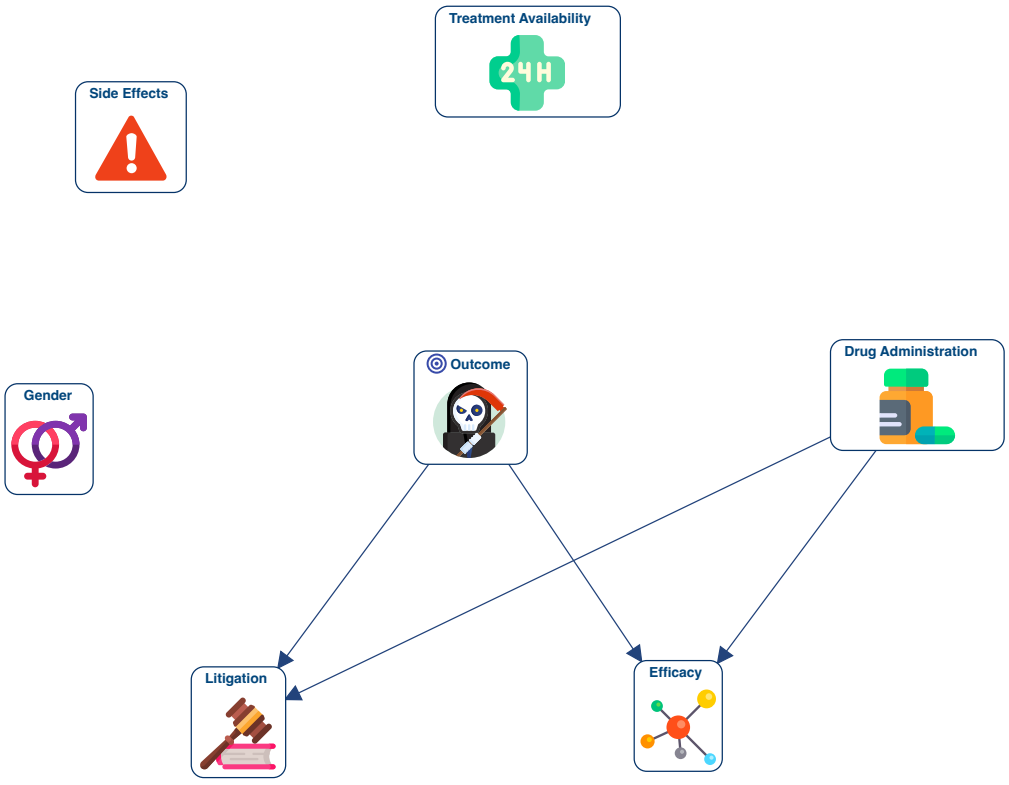
- Graph Surgery
  - requires a fully specified Causal Bayesian Network;
  - uses the mutilated Causal Bayesian Network for causal inference;
- Likelihood Matching
  - requires the causal analysis of the domain to define the variables that need to be adjusted for to block the non-causal paths and let the causal paths open;
  - uses the Bayesian network to carry out probabilistic inference with the adjusted variables. Note that this network does not have to be causal! It just needs to represent the Joint Probability Distribution of the domain.

This last point is especially important. It is indeed sometimes challenging, if not impossible, to design the fully specified Causal Bayesian Network. However, BayesiaLab offers a wide range of machine-learning algorithms that we can use to induce a network that represents the Joint Probability Distribution. Hence, we only need to have a limited amount of causal knowledge to define the variables that have to be adjusted for.

For example, suppose we machine-learned the network below with `Main Menu > Learning > Supervised Learning > Augmented Naive Bayes`:



The main architecture of the network is Naïve, i.e., the Target Node is the parent of all nodes. Therefore, this Bayesian network is clearly not causal. If we were to use Graph Surgery, we would not find any total or direct effects (see the corresponding mutilated graph below when estimating the Direct Effects with *Efficacy* and Litigation defined as Non_Confounder).



However, Likelihood Matching returns the correct estimations for the Total Effects with two separate analyses.

One analysis for Gender, without adjusting for any variables:

Total Effects on Target (SimpsonAugmented2DataANB)

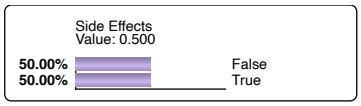
Analysis Context
No Observation

Total Effects on Target Outcome

Node	Prior Value/Mean	Standardized Total Effect	Total Effect	G-test	df	p-value	G-test (Data)	df (Data)	p-value (Data)
Gender	0.50	-0.37	-0.37	70,537.54	1	0.00%	70,535.94	1	0.00%

Close Quadrants

And one analysis for *Drug Administration*, by holding constant Side Effects:



Total Effects on Target (SimpsonAugmented2DataANB)

Analysis Context	
Joint Probability	100.00%
Side Effects	p {False: 49.99%, True: 50.01%}

Total Effects on Target Outcome

Node	Prior Value/Mean	Standardized Total Effect	Total Effect	G-test	df	p-value
Drug Administration	0.39	-0.08	-0.08	3,109.10	1	0.00%

Close Quadrants

As for Direct Effects, the analysis can be carried out for both variables with the current definition of Non_Confounders.

Direct Effects on Target (SimpsonAugmented2DataANB)

Analysis Context
No Observation

Not Fixed Nodes

Node	Non Confounder	Factor	Not Observable
Efficacy	X		
Litigation	X		

Direct Effects on Target Outcome

Node	Prior Value/Mean	Standardized Direct Effect	Direct Effect	Elasticity	Contribution
Gender	0.50	-0.40	-0.40	-39.78%	83.76%
Drug Administration	0.39	-0.08	-0.08	-7.89%	16.24%

Close Quadrants

Conclusion

This chapter highlights how much effort is required to derive causal effect estimates from observational data. Simpson's Paradox illustrates how much can go wrong even in simple circumstances. Given such

potentially serious consequences, it is a must for policy analysts to examine all aspects of causality formally. To paraphrase Judea Pearl, we must not leave causal considerations to the mercy of intuition and good judgment. Fortunately, causality has emerged from its pariah status in recent decades, which has allowed tremendous progress in theoretical research and practical tools: "[...] practical problems relying on causal information that long were regarded as either metaphysical or unmanageable can now be solved using elementary mathematics" (Pearl, 1999).

Example: Simpson's Paradox

Introduction

Simpson's Paradox illustrates the implications of falsely assuming ignorability. This will lead us to abandon the idea of ignorability and, along with it, the potential outcomes framework and replace it with a formal identification and estimation process that relies on graphical models.

We will use an example that appears trivial on the surface but which has produced countless instances of false inference throughout the history of science. Due to its counterintuitive nature, this example has become widely known as Simpson's Paradox (Wall Street Journal, Dec. 2, 2009).

This is an important exercise as it illustrates how an incorrect interpretation of an association can produce bias. The word "bias" may not necessarily strike fear into our hearts. In our common understanding, "bias" implies "inclination" and "tendency," and it is perhaps not a particularly forceful expression. Hence, we may not be overly troubled by a warning about bias. However, Simpson's Paradox shows how bias can lead to catastrophically wrong estimates.

Does the Treatment Kill Patients?

A hypothetical disease equally affects men and women. An observational study finds that a certain treatment is linked to an increase in the recovery rate among all treated patients from 40 to 50%. Based on the study, this new treatment is widely recognized as beneficial and subsequently promoted as a new therapy.

	Patient Recovered	
Treatment	Yes	No
Yes	50%	50%
No	40%	60%

We can imagine a headline along the lines of "New Therapy Increases Recovery Rate by 10%." However, when examining patient records by gender, the recovery rate for male patients–upon treatment–decreases from 70% to 60%; for female patients, the recovery rate declines from 30% to 20%. Men are, therefore, more likely to recover than women, with or without treatment.

		Patient Recovered	
Gender	Treatment	Yes	No
Male	Yes	60%	40%
	No	70%	30%
Female	Yes	20%	80%
	No	30%	70%

So, is this new treatment effective overall or not? This puzzle can be resolved by realizing that, in this observed population, there was an unequal application of the treatment to men and women, i.e., some type of self-selection occurred. More specifically, 75% of male patients and only 25% of female patients received the treatment. Although the reason for this imbalance is irrelevant for inference, one could imagine that the side effects of this treatment are much more severe for females, who thus seek alternative therapies. As a result, there is a greater share of men among treated patients. Given that men have a better a priori recovery prospect with this type of disease, the recovery rate of all treated patients increases. So, what is the true causal effect of this treatment?

Synthetic Data

Our particular manifestation of Simpson's Paradox is not very far-fetched, but it is still fictional. Therefore, we must rely on synthetic data to make this problem domain tangible for our study efforts. We generate 1,200 observations by sampling from the joint probability distribution of the original Data-Generating Process (DGP). Needless to say, for this dataset to be a suitable example for non-experimental observations like we would find under real-world conditions, the true DGP is not known but merely an assumption.

Our synthetic dataset consists of three variables with two discrete states each:

- X (Treatment): Yes (1)/No (0)
- Y (Outcome): Recovered (1)/Not Recovered (0)
- Z (Gender): Male (1)/Female (0)

The following table shows a preview of the first ten rows of the dataset:

Z (Gender)	X (Treatment)	Y (Outcome)
Female (0)	No (0)	Not Recovered (0)
Male (1)	No (0)	Not Recovered (0)
Female (0)	Yes (1)	Recovered (1)
Female (0)	Yes (1)	Not Recovered (0)
Male (1)	No (0)	Recovered (1)
Female (0)	No (0)	Not Recovered (0)
Male (1)	Yes (1)	Not Recovered (0)
Male (1)	Yes (1)	Recovered (1)
Female (0)	Yes (1)	Not Recovered (0)
Female (0)	No (0)	Not Recovered (0)

Data Download

Download: `Simpson.csv`

Sources of Causal Information

Causal Inference by Experiment

Randomized experiments have always been the gold standard for establishing causal effects. For instance, in the drug approval process, controlled experiments are mandatory. Without first having established and quantified the treatment effect, and any associated side effects, no new drug could win approval by the Federal *Drug Administration*.

Causal Inference from Observational Data and Theory

However, in many other domains, experiments are not feasible, be it for ethical, economic, or practical reasons. For example, it is clear that a government could not create two different tax regimes to evaluate their respective impact on economic growth. Neither would it be possible to experiment with two different levels of carbon emissions to measure a warming effect on the global climate.

"So, what does our existing data say?" would be an obvious question from policymakers, especially given today's high expectations concerning Big Data. Indeed, in lieu of experiments, we can attempt to find instances in which the proposed policy already applies (by some assignment mechanism) and compare those to other instances in which the policy does not apply.

However, as we will see in this chapter, performing causal inference on the basis of observational data requires an extensive range of assumptions, which can only come from theory, i.e., domain-specific knowledge. Despite all the wonderful advances in analytics in recent years, data alone, even Big Data, cannot prove the existence of causal effects.

Historical Context

Today, we can openly discuss how to perform causal inference from observational data. For the better part of the 20th century, however, the prevailing opinion had been that speaking of causality without experiments is unscientific. Only towards the end of the century, this opposition had slowly eroded (Rubin 1974, Holland 1986), which subsequently led to numerous research efforts spanning philosophy, statistics, computer science, information theory, etc. The Potential Outcomes Framework has played an important role in this evolution of thought.

Potential Outcomes Framework

Although there is no question about the common-sense meaning of "cause and effect," for formal analysis, we require a precise mathematical definition. In the fields of social science and biostatistics, the potential outcomes framework is a widely accepted formalism for studying causal effects (the potential outcomes framework is also known as the counterfactual model, the Rubin model, or the Neyman-Rubin model). Rubin (1974) defines "causal effect" as follows:

> "Intuitively, the causal effect of one treatment, $T = 1$, over another, $T = 0$, for a particular unit and an interval of time from t_1 to t_2 is the difference between what would have happened at time t_2 if the unit had been exposed to $T = 1$ initiated at t_1 and what would have happened at t_2 if the unit had been exposed to $T = 0$ initiated at t_1: 'If an hour ago I had taken two aspirins instead of just a glass of water, my headache would now be gone,' or because an hour ago I took two aspirins instead of just a glass of water, my headache is now gone.' Our definition of the causal effect of $T = 1$ versus $T = 0$ treatment will reflect this intuitive meaning."

In this quote, we altered the original variable name E to $T = 1$ and C to $T = 0$ in order to be consistent with the nomenclature in the remainder of this chapter. T is commonly used in the literature to denote the treatment condition.

- $Y_{i,1}$ Potential outcome of individual i given treatment $T = 1$ (e.g., taking two Aspirins)
- $Y_{i,0}$ Potential outcome of individual i given treatment $T = 0$ (e.g., drinking a glass of water)

The individual-level causal effect (ICE) is defined as the difference between the individual's two potential outcomes, i.e.,

$ICE = Y_{i,1} - Y_{i,0}$

Given that we cannot rule out differences between individuals (effect heterogeneity), we define the average causal effect (ACE) as the unweighted arithmetic mean of the individual-level causal effects:

$ACE = E[Y_{i,1}] - E[Y_{i,0}]$

$E[\cdot]$ denotes the expected value, i.e., the unweighted arithmetic mean.

The challenge is that $Y_{i,1}$ (treatment) and $Y_{i,0}$ (non-treatment) can never be both observed for the same individual at the same time. We can only observe treatment or non-treatment, but not both.

So, where does this leave us? What we can produce easily is the "naive" estimator of association S between the "treated" and the "untreated" sub-populations:

$S = E[Y \mid T = 1] - E[Y \mid T = 0]$

For notational convenience, we omit the index i because we are now referring to sub-populations and not to an individual.

Because the sub-populations in the treated and untreated groups contain different individuals, S is not necessarily a measure of causation, in contrast to ACE.

The question is, how can we move from what we can measure, i.e., the naive association, to the quantity of interest, i.e., the causal effect? Determining whether we can measure causation from association is known as identification analysis.

We must check whether there were any conditions under which the measure of association, S, equals the measure of causation, ACE. As a matter of fact, this would be the case if the sub-populations were comparable with respect to all confounders, i.e., the factors that could also influence the outcome.

Ignorability

Remarkably, the conditions under which we can measure causal effects from observational data are very similar to those that justify causal inference in randomized experiments. A pure random selection of treated and untreated individuals does indeed remove any potential selection bias and leaves the confounding factor distributions identical in the sub-populations, thus allowing the estimation of the effect of the treatment alone. This condition is known as "ignorability," which can be formally written as:

$(Y_1, Y_0) \perp A$

This means that the potential outcomes, Y_1, and Y_0 must jointly be independent of the treatment assignment, A. This condition of ignorability holds in an ideal experiment. Unfortunately, this condition is very rarely met in observational studies. However, conditional ignorability may hold, which refers to ignorability within subgroups of the domain defined by the values of X (note that X can be a vector).

In words, conditional on variables X, Y_1, and Y_0 are jointly independent of A, the assignment mechanism. If conditional ignorability holds, we can utilize the estimator, $S \mid X$, to estimate the average causal effect, $ACE \mid X$.

$$\begin{aligned} ACE \mid X &= E[Y_1 \mid X] - E[Y_0 \mid X] \\ &= E[Y_1 \mid A, X] - E[Y_0 \mid A, X] \\ &= E[Y \mid T = 1, X] - E[Y \mid T = 0, X] \\ &= S \mid X \end{aligned}$$

How can we select the correct set of variables X among all variables in a system? How do we know that such variables X are observed or even exist in a domain? This is what makes the concept of ignorability highly problematic in practice. Pearl (2009) states:

> The difficulty that most investigators experience in comprehending what "ignorability" means, and what judgment it summons them to exercise, has tempted them to assume that it is automatically satisfied, or at least is likely to be satisfied if one includes in the analysis as many covariates as possible. The prevailing attitude is that adding more covariates can cause no harm (Rosenbaum 2002, p. 76) and can absolve one from thinking about the causal relationships among those covariates, the treatment, the outcome, and, most importantly, the confounders left unmeasured (Rubin 2009).

The absence of hard-and-fast criteria makes ignorability a potentially dangerous concept for practitioners.

Chapter 11: Causality and Optimization

Introduction

> Half the money I spend on advertising is wasted; the trouble is I don't know which half.

Over the last century, various versions of this quote have been attributed to John Wanamaker, Henry Ford, and William Procter, among others. Yet, 100 years after these marketing pioneers, in this day and age of big data and advanced analytics, the quote still rings true among marketing executives. The ideal composition of advertising and marketing efforts remains the industry's Holy Grail. Certainly, there are many advertising agencies and market research firms that promote their proprietary methodology in pursuit of the optimum allocation of marketing resources. Also, there have been decades of research in marketing science on this topic. Yet, despite all commercial and academic efforts, there is a remarkable lack of universally accepted methods for marketing mix modeling and optimization. As a result, the current practice remains "more art than science."

We speculate that the lack of a well-established marketing mix methodology has little to do with the domain itself. Rather, it reflects the fact that marketing is yet another domain that frequently has to rely on non-experimental data for decision support. As such, marketing mix optimization is a rather prototypical problem that mirrors the challenges of many other fields.

What is perhaps unique to marketing is the large number of instruments, i.e., the wide range of advertising channels and promotions, that can be utilized as individual levers in reaching and convincing consumers. Moreover, many marketing instruments can be easily quantified in terms of cost. Hence, the marketing domain lends itself as a teaching example for this chapter.

Marketing Mix Modeling Workflow

Background, Challenges, and Objectives

Background & Challenges

While this chapter's example is inspired by a real business, we are trying to minimize any resemblance to a particular company or industry. We shall refer to our fictional business as ACME Corp. All of its sales and marketing data are synthetically generated. This allows us to illustrate a variety of effects in a single composite example. With the few publicly available datasets of real marketing and sales data, we might not be able to observe the range of characteristics that we wish to present here. Also, for ease of interpretation, we have magnified some effects, resulting in a somewhat idealized consumer response to the marketing actions of ACME Corp. With artificial data, we also have the luxury of plentiful observations, although that is not necessarily unrealistic.

Variables

Our fictional business ACME utilizes a variety of marketing and advertising channels. Throughout this chapter, we will also refer to them as "marketing drivers" or just "drivers."

- *TV Advertising*
- *Internet Advertising*
- *Print Advertising*
- *Direct Marketing*
- *Incentives* (i.e., price discounts)

These variables are all measured on proprietary scales. For instance, *TV Advertising* might be measured in GRP (Gross Rating Points), and Print might be recorded in columns-inches. *Incentives* refer to price promotions and discounts measured in dollars. All these marketing instruments we consider as being under ACME's control, i.e., we can set them to any desired level within an overall budget constraint.

Furthermore, we have a target variable, *Sales*, measured in units sold daily, which we hope to improve by optimizing the mix of marketing instruments.

Beyond the variables that are under ACME's control, there are four calendar variables:

- *Quarter*
- *Weekday*
- *Month*
- *End-of-Month Indicator*

Finally, we measure a number of variables that are beyond ACME's direct control but still have an influence on the business, including:

- *Co-Op Promotions* (promotions sponsored by the vehicle manufacturer)
- *Competitive Incentives* (i.e., price discounts on competitive products)
- *Web Traffic* (organic traffic to ACME's website–not paid-for traffic)
- *Showroom Traffic* (organic visits to ACME's facilities)
- *Test Drives* (organic)

While numerous other variables do certainly exist in this domain, we have no further data available. Later, we will need to formalize our assumptions in this regard.

Objectives

For a comprehensive study of marketing mix modeling and optimization, there are numerous questions we should consider, such as:

- Which form of advertising is the strongest driver of ACME's sales?
- How do competitive incentives affect ACME's sales?
- What is the optimum marketing mix overall, given different levels of marketing budget constraints?
- Are there saturation effects of certain marketing channels?
- Are there counterproductive promotions?
- How can we attribute the observed sales volume to marketing initiatives?
- What would be the baseline sales volume without any advertising?
- Are there synergy effects that make some instruments more important jointly than their individual importance?

Causal Questions

The single most important thing we need to recognize regarding the above questions is that they are all causal questions. This means we are not looking for a prediction of *Sales* based on the observation of marketing variables. Rather, we wish to simulate the manipulation of all marketing variables in such a way that we maximize *Sales*. Thus, we are performing an intervention in our domain, which requires causal inference. This is the reason why we can only introduce the marketing mix question after having established foundational causal concepts in Chapter 10. There, we explained how a causal graph, combined with certain criteria, can tell us precisely what variables we have to adjust to identify a causal effect. We merely had to provide a causal graph, i.e., encode our causal understanding of the domain. With three variables in Simpson's Paradox, there were only 25 possible causal structures. A common-sense understanding of the domain allowed us to quickly identify the only reasonable graph in terms of causal directions. In that case, making assumptions about the full causal structure was straightforward. So, we may now feel well-equipped to answer more complex causal questions, such as the given marketing domain.

From 25 to 1,439,428,141,044,398,334,941,790,719,839,535,103 Graphs

Going from three variables in Simpson's Paradox to 14 variables in this marketing example, it would be reasonable to expect a substantial increase in the number of possible causal networks. As it turns out, given a set of 14 variables, there are now 1.4×1036 possible causal structures as opposed to 25. Clearly, we can no longer rely on our intuition to pick the correct one out of over one undecillion possible graphs. Furthermore, clever algorithms and fast computers cannot help us with this task either. As it stands, causal directions can generally not be discovered through machine learning.

However, without a causal graph, we cannot use the familiar criteria for confounder selection, such as the Adjustment Criterion. And, without the ability to select and control for confounders, we cannot employ the usual estimation methods. A commonly used fall-back position is to simply "control for

all pretreatment covariates" (Rubin, 2009). However, the example in the previous chapter highlighted the risks of doing that. So, it appears that we have already reached a dead end with our example.

For a comprehensive study of marketing mix modeling and optimization, there are numerous questions we should consider, such as:

- Which form of advertising is the strongest driver of ACME's sales?
- How do competitive incentives affect ACME's sales?
- What is the optimum marketing mix overall, and given different levels of marketing budget constraints?
- Are there saturation effects of certain marketing channels?
- Are there counterproductive promotions?
- How can we attribute the observed sales volume to marketing initiatives?
- What would be the baseline sales volume without any advertising?
- Are there synergy effects that make some instruments more important jointly as opposed to their individual importance?

Causal Questions

The single most important thing we need to recognize regarding the above questions is that they are all causal questions. This means we are not looking for a prediction of *Sales* based on the observation of marketing variables. Rather, we wish to simulate the manipulation of all marketing variables in such a way that we maximize *Sales*. Thus, we are performing an intervention in our domain, which requires causal inference. This is the reason why we can only introduce the marketing mix question after having established foundational causal concepts in Chapter 10. There, we explained how a causal graph, combined with certain criteria, can tell us precisely what variables we have to adjust for identifying a causal effect. We merely had to provide a causal graph, i.e., encode our causal understanding of the domain. With three variables in Simpson's Paradox, there were only 25 possible causal structures. A common-sense understanding of the domain allowed us to quickly identify the only reasonable graph in terms of causal directions. In that case, making assumptions about the full causal structure was straightforward. So, we may now feel well-equipped to answer more complex causal questions, such as the given marketing domain.

From 25 to 1,439,428,141,044,398,334,941,790,719,839,535,103 Graphs

Going from three variables in Simpson's Paradox to 14 variables in this marketing example, it would be reasonable to expect a substantial increase in the number of possible causal networks. As it turns out, given a set of 14 variables, there are now 1.4×1036 possible causal structures as opposed to 25. Clearly, we can no longer rely on our intuition to pick the correct one out of over one undecillion possible graphs. Furthermore, clever algorithms and fast computers cannot help us with this task either. As it stands, causal directions can generally not be discovered through machine learning.

However, without a causal graph, we cannot use the familiar criteria for confounder selection, such as the Adjustment Criterion. And, without the ability to select and control for confounders, we cannot employ the usual estimation methods. A commonly used fall-back position is to simply "control for all pretreatment covariates" (Rubin, 2009). However, the example in the previous chapter highlighted the risks of doing that. So, it appears that we have already reached a dead end with our example.

The Disjunctive Cause Criterion

As it turns out, recent research has made significant progress and produced a new criterion for selecting confounders. VanderWeele and Shpitser (2010) have discovered that it is possible to select confounders without knowing the full causal graph:

> We show that irrespective of the true causal structure is and irrespective of whether there are important unobserved variables if there exists some subset of the observed covariates that suffice to control for confounding, then the set obtained by applying our criterion will also constitute a set that suffices.

This is a profound insight, given that not knowing the causal structure between covariates is not at all unique to our example. We speculate that "causal ignorance" is the prevailing condition in most research projects. VanderWeele and Shpitser have shown that confounders can be found differently:

> We propose that control be made for any [pre-treatment] covariate that is either a cause of treatment or of the outcome or both.

In other words, we now need to ask the common-sense question about each covariate, "is it a cause of the treatment or the outcome or both?" If the answer is yes, the variable in question is a confounder, and we must control for it to estimate the causal effect. With this new approach to confounder selection, we do not need to know–or even consider–the relationships between the covariates.

There are a number of caveats, though. Once again, we have to assume that there are no unobserved confounders. Furthermore, we must also assume that there exists a set of variables Z that would meet one of the formal identification criteria. If this assumption holds, the proposed selection criterion will identify the set of variables Z. We must stress that such an assumption cannot be tested. It can only be justified on theoretical grounds. Nevertheless, it is a much weaker assumption than claiming to know the full causal structure.

Confounder Selection in Practice

In the context of this marketing example, we consider *Sales* as the outcome variable. However, unlike a single treatment X in Simpson's Paradox, we now have many potential treatments, i.e., all the marketing variables. Not only do we have to identify Confounders with regard to one treatment/outcome relationship, but we need to do this for all treatment/outcome pairs. For instance, if we were considering *TV Advertising* as treatment, we will need to check all covariates as to whether they are causes of *TV Advertising*, *Sales*, or both.

In this domain, it is fairly easy to judge that all of ACME's advertising efforts (*TV Advertising*, *Internet Advertising*, *Print Advertising*, *Direct Marketing*) and *Incentives* should be seen as causes of *Sales*. We also reason that the calendar variables, Quarter, Weekday, Month, and *End-of-Month Indicator* are also causes of *Sales*. It is common knowledge, for instance, that Saturday is the main car shopping day in the U.S. Also, the fourth quarter marks the start of a new model year and concludes the calendar year, which makes it the peak selling season.

Five variables remain, i.e., *Co-Op Promotions*, *Competitive Incentives*, *Web Traffic*, *Showroom Traffic*, and *Test Drives*. Are they not also causes of *Sales*? Yes, but we argue that they are not pre-treatment

variables. Rather, given our domain knowledge, we believe that these variables "respond" to ACME's marketing and advertising efforts, meaning that they are "downstream" from the original causes. For example, the original cause, *Print Advertising*, drives *Showroom Traffic*, which subsequently leads to *Sales*.

Regarding *Competitive Incentives* being a Non-Confounder, we argue that the competition presumably wants to counteract ACME efforts. If ACME increases incentives as part of a campaign, competitors would presumably follow suit with their own incentives. However, one may object to this line of reasoning and instead suggest that *Competitive Incentives* come first and that it prompts ACME to react. From that viewpoint, *Competitive Incentives* would definitely be pre-treatment. However, if we treated *Competitive Incentives* as Confounder, it would imply that the competition would "hold still" while ACME tries out different marketing spend levels during optimization. Clearly, this would be an unrealistic assumption. Instead, we believe that it is reasonable to think that the competition would react similarly as they have always done historically.

With that, we have specified all explicit assumptions regarding this domain. Furthermore, we have also assumed implicitly that there are no unobserved Confounders, i.e., that no other hidden variables exist that influence our domain. Such a claim is almost as bold as assuming the complete causal structure of this domain. Also, there is no way to test for the existence of hidden Confounders. As outrageous as this may seem, this assumption is made in virtually all models based on observational data, regardless of the modeling technique. The only way we can justify this assumption is on theoretical grounds, i.e., we need to have domain knowledge that allows us to rule out unobserved Confounders.

Estimation Challenges

Now that we have identified the Confounders, we would expect to be able to estimate the causal effects. Theoretically, the Adjustment Formula (i.e., stratification) could serve as our computation method. Why is this not immediately feasible?

The first objection is that our dataset consists of mostly continuous variables rather than discrete states, which was the case in the previous chapter. However, we can overcome this challenge by discretizing all continuous variables, which we have demonstrated repeatedly in this book.

With the discretized dataset of our domain, a new problem looms: assuming that all confounders now have 5 discrete states, we would need to calculate the weighted average of approximately 2 million strata to estimate the effect of one treatment. In other words, we would require the entire joint probability table representing the domain. Even if we could manage the computational task with a computer fast enough and with enough memory to store the joint probability table, we would not have anywhere near enough observations to estimate this table. While we have thousands of daily observations, our joint probability table would consist of millions of rows.

Summary of Challenges

Let us summarize where we stand. We started this chapter with the challenge that we could not encode one true causal graph. Thus, identification using traditional criteria was not possible. The new Disjunctive Cause Criterion by Shpitser and VanderWeele saved us from having to define a full causal graph. Fewer, simpler assumptions will now suffice to select the confounders. But now that we have the Confounders, our straightforward estimation techniques, e.g., stratification, will no longer work. It seems for every step forward, we must take another one back.

Costs and Resources

Function Nodes

The question of budget brings up another issue. Thus far, all variables are shown on their original, proprietary scale without any cost information. For instance, we have not yet defined how much "one unit" of *TV Advertising* costs in dollars. Prior to version 6 of BayesiaLab, the Cost property was available to specify the unit cost for each variable. At the time, one could have specified that 1 GRP costs $1,000. However, real-world applications are not as straightforward as having a fixed price per unit. As is the case with most business transactions, volume discounts may apply that need to be considered when optimizing media spend.

With BayesiaLab 6, we introduced the concept of Function Nodes. They facilitate the computation of scalar values based on the distribution and values of the states in nodes. This is best illustrated in the context of our example. We will now use a Function Node to "translate" the original units of *TV Advertising* into dollar values.

> A Function Node calculates values ad hoc. As such, a Function Node does not exist in the original dataset.

Introducing the Function Node

- In Modeling Mode F4, click on the Function Node Creation Mode by clicking on the corresponding icon on the menu bar.
- Position the node on the Graph Panel. By default, the first Function Node to be introduced has the name F1.
- Go into Arc Creation Mode and draw an arc from TV to F1.

For random nodes, the warning symbol means that the Conditional Probability Table has not been estimated yet. In the context of Function Nodes, it means that an equation has yet to be defined that will determine the value of the Function Node.

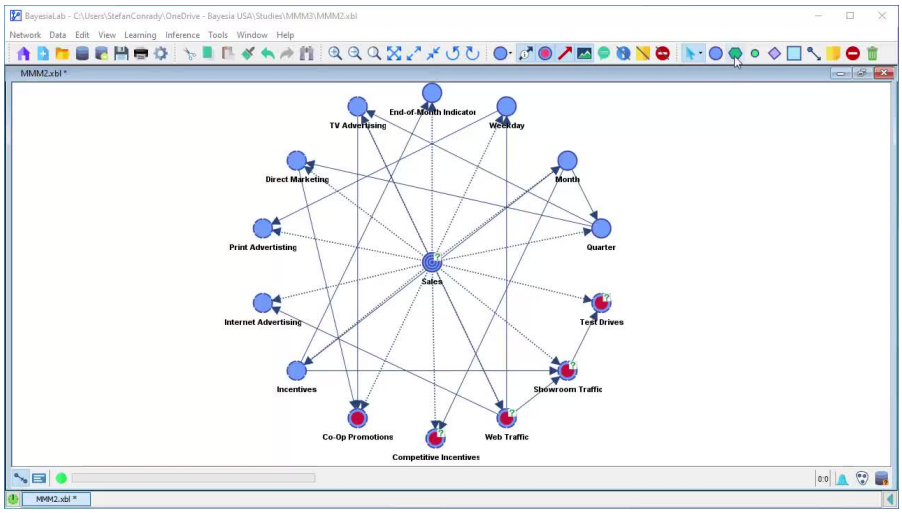
- Open the Function Node F1 by double-clicking on it, which brings up the Node Editor.

Note the *TV Advertising* node listed in the center of the three panels at the bottom of the window. This is where the parent nodes of a Function Node are shown. In our case, *TV Advertising* is currently the only parent node. This means that *TV Advertising* is the only variable that can be included in the Equation Tab in the top panel of the window. Whereas a Function Node, such as F1, represents scalar values, "normal" nodes, such as *TV Advertising*, always represent distributions of states.

This is where the functions in the bottom left panel come into play, in particular the Inference Functions. We can use them to extract a scalar statistic from *TV Advertising*, which F1 will then represent as a scalar value.

Here, as a first step, we want F1 to represent the mean value of the cost of *TV Advertising*:

- Double-click on `Inference Functions` and then double-click on MeanValue(v). This adds the inference function to the Equation Tab. By default, a placeholder variable v is highlighted in the equation
- You can single-click on *TV Advertising* to bring up the domain range of this variable for your information
- Double-click on *TV Advertising* to add ?*TV Advertising*? to the Equation Tab. ?*TV Advertising*? should automatically assume the position of v if that placeholder was still highlighted. The final syntax will appear as ?F1?=MeanValue(?*TV Advertising*?)
- Click `Validate` to check the syntax and have BayesiaLab compute the value of F1.

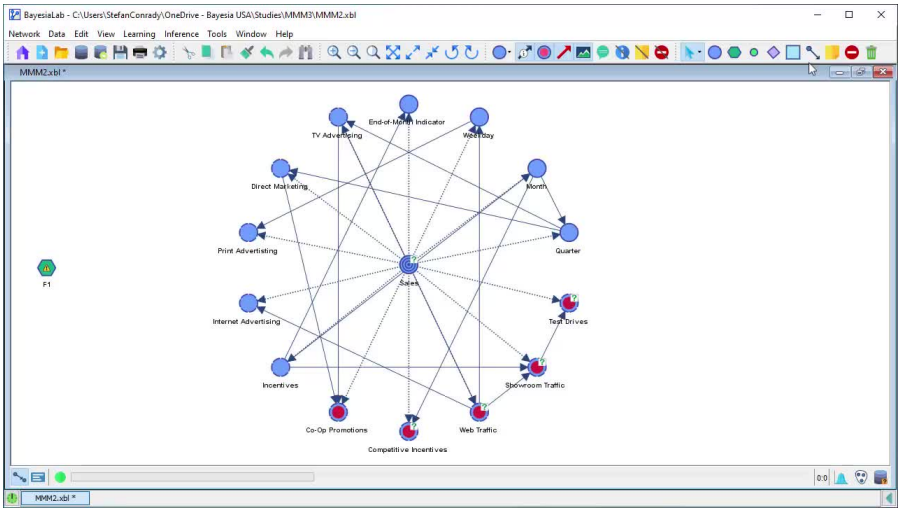


Click to view video

Summing Up the Costs

Where are we going with this? We will now add further parents to F1 so that we can calculate the cumulative cost of all Confounders. We simply draw arcs from all the Confounders to F1.

- Select the `Arc Creation Mode` and draw arcs from all Confounders to F1.
- Hold `L` to remain in the Arc Creation Mode so you can keep adding arcs without having to go back to the toolbar.

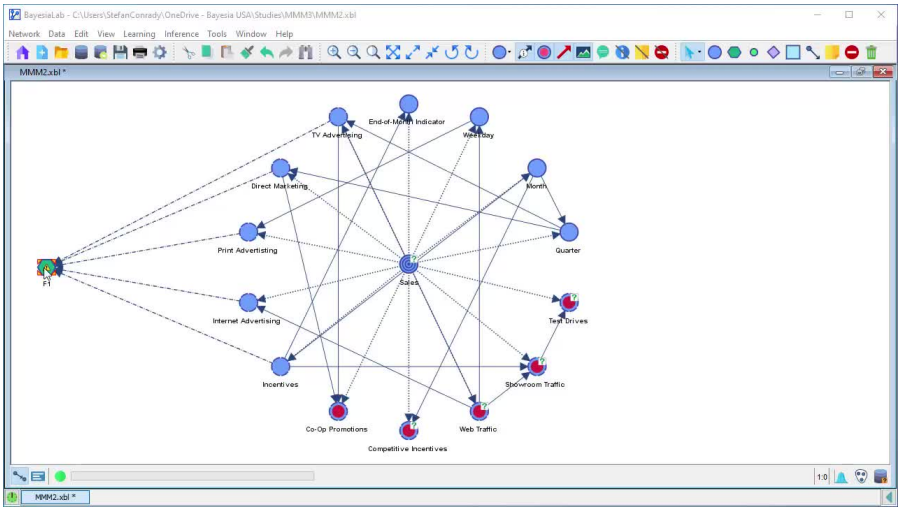


Click to view video

Recall that we did not draw any arcs from Weekday, Month, Quarter, and *End-of-Month Indicator* to F1. The reason is that these calendar-related variables are beyond the control of ACME. Therefore, ACME could not buy more Saturdays or pay for a longer fourth quarter. Also, we will not draw arcs from the Non-Confounders to F1 as ACME cannot directly control them through their budget either.

The Function Node F1 can now serve as a "summary node" for all Confounders. If all Confounders were recorded on the same scale and had the same cost of $1/unit, you could sum up the mean values of all the Confounders:

- Double-click F1 to open the Node Editor.
- Enter this expression into the Equation Tab: `MeanValue(?TV?)+MeanValue(?Print?)+MeanValue(?Online?)+MeanValue(?Incentives?)+MeanValue(?Direct Marketing?)`
- Instead of typing the entire syntax, you can also add the inference function MeanValue() and all listed Confounders by double-clicking on the respective items in the lower panels of the Node Editor.



Click to view video

Needless to say, assuming a cost of $1/unit for each type of advertising is entirely unrealistic. And the purpose of having a Function Node is that we can enter any arbitrary cost function for each advertising channel. A typical example would be entering a quantity discount in the form of an if/then statement:

```
?F1?=IF(MeanValue(?Print Advertising?)>=10, 0.9*MeanValue(?Print Advertising?),MeanValue(?Print Advertising?))
```

This statement would discount the cost of *Print Advertising* by 10% once 10 units are reached. This way, you can define even complex pricing and discount structures. Why is this so important? The optimization algorithm that BayesiaLab employs can take advantage of any such additional nonlinearities.

However, given that our model is based on synthetic data that already features plenty of nonlinearities, it serves no educational purpose to add further artifacts to our problem domain. Hence, we stick with a fictional cost of $1/unit for each advertising channel.

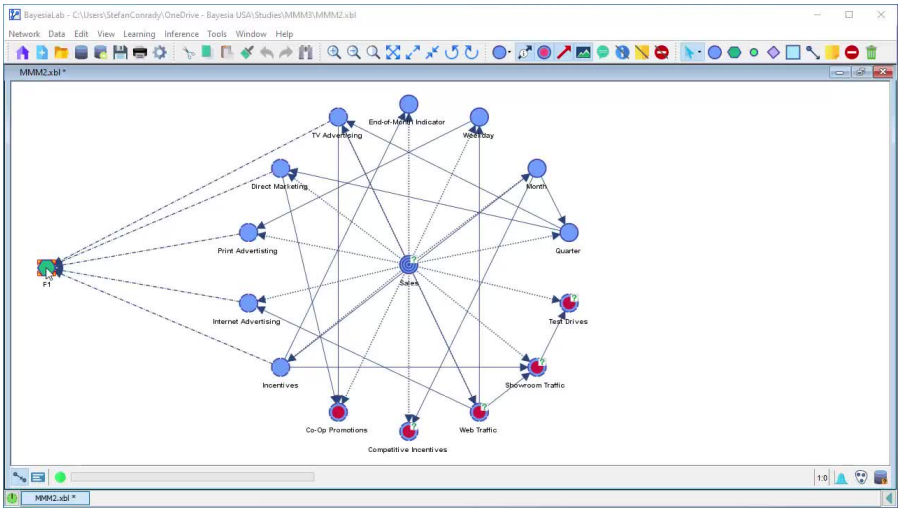
Resources

A Function Node is a highly flexible element in BayesiaLab and can play many different roles. Here, we justified its use by our need to calculate the cumulative cost of all advertising efforts.

However, not only do we need to know the total cost, but we also need to constrain it in the subsequent optimization. If cost were no object, we could simply read the optima from the Target Mean Analysis plot by taking the x-levels that correspond to the maximum y-levels for each driver. Alas, budget constraints do apply in the real world.

In BayesiaLab, we can formalize the "budget" role of F1 by adding it to the pre-defined Class Resource.

- Right-click on F1.
- From the Contextual Menu, select `Properties > Classes > Add`.
- Then, check `Predefined Class` and select Resource from the drop-down menu.
- Click `Yes` to conclude the step.



Click to view video

Note that we could add additional Function Nodes to this Class. This way, the Class Resource can represent the sum of multiple Function Nodes.

Not-Observable Nodes

We previously pointed out that Quarter, Weekday, Month, and *End-of-Month Indicator* neither have any monetary cost. The positive effect of Weekday=Saturday on *Sales* is a "free" benefit to ACME. And the absence of arcs going into F1 already prevents these nodes from being included in the monetary cost summary in F1.

However, as we prepare for optimization, we must also encode formally that these variables cannot be modified or influenced by anyone. In other words, ACME cannot manipulate them. Thus, Quarter, Weekday, Month, and *End-of-Month Indicator* require a special designation so that BayesiaLab *includes* them in the Likelihood Matching of the Confounders but excludes them from active manipulation during optimization.

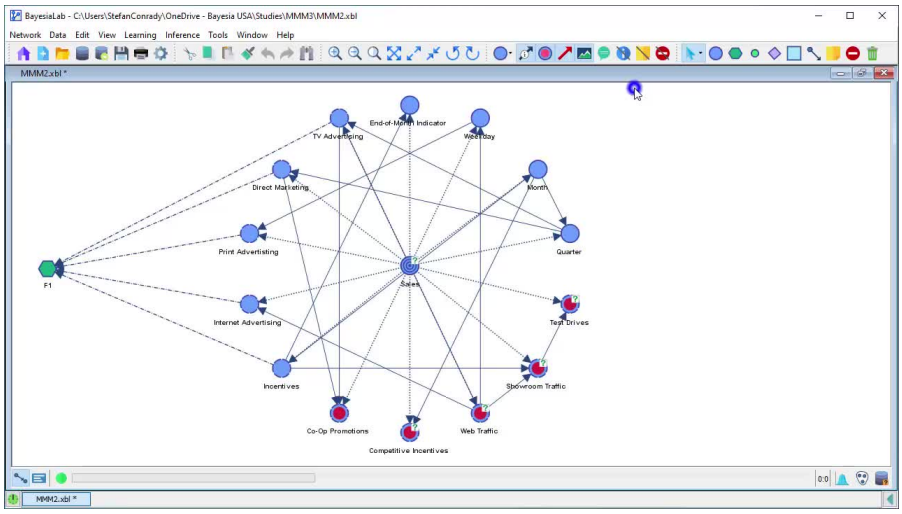
Unfortunately, the BayesiaLab jargon will now become a bit convoluted. The "do-not-manipulate" assignment of variables is done via their Cost attribute. This Cost is not to be confused with the monetary cost in terms of dollars, which is computed by F1. In general, the Cost attribute of a variable quantifies the effort required to observe a variable (see the Diagnosis example in Chapter 6). A special case is Cost=0, which makes a variable Not Observable in BayesiaLab. In the context of the calendar variables in our example, this nomenclature is counterintuitive as we would think that the dates in the calendar are certainly observable. However, it goes beyond the scope of this chapter to present the rationale of this terminology. For the purposes of this example, we set Cost=0 to exclude the calendar variables from being manipulated during the optimization.

Assigning the Observation Cost

- Select the calendar variables and then right-click on any one of them.
- From the Contextual Menu, select `Properties > Cost`
- Uncheck the box Cost or enter 0 in the `Edit Cost` window.

- Click OK

The Not Observable status of the variables is now reflected in the light purple color of the calendar nodes.



Click to view video

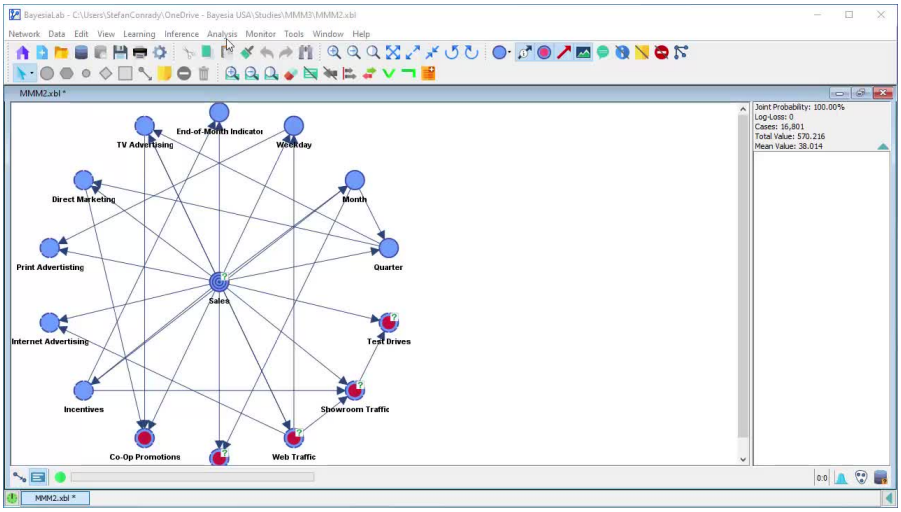
Direct Effects Analysis

Direct Effects on Target Report

We now return to our marketing example for good and utilize Likelihood Matching to estimate the Direct Effect of each driver variable on the Target Node *Sales*.

- From within the F5](/bayesialab/user-guide/main-menu/view/validation-mode), select `Menu > Analysis > Report > Target > Direct Effects on Target`.

This prompts BayesiaLab to estimate the Direct Effects of each driver variable with regard to the Target Node while performing Likelihood Matching on all Confounders.



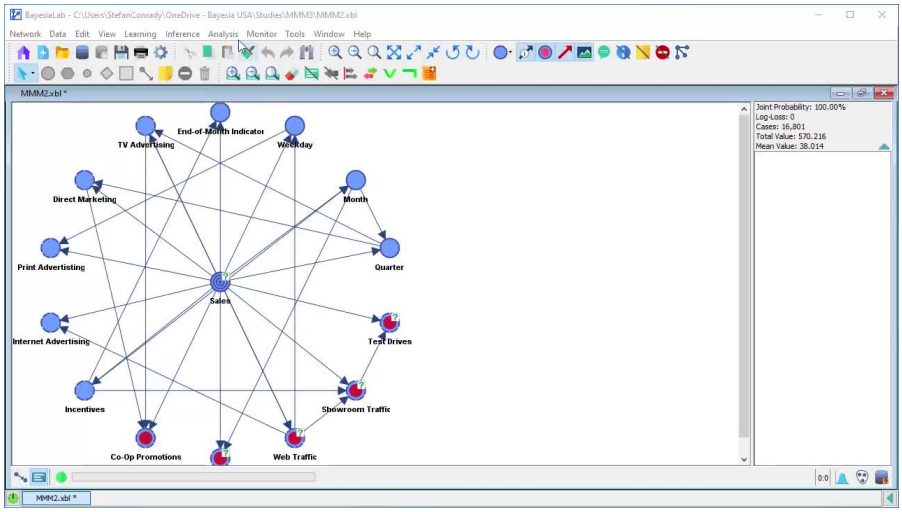
Click to view video

The resulting table resembles the typical output we would obtain from a linear regression analysis with parameter estimates for each covariate. As such, we may be tempted to interpret the Direct Effect as the slope of a response curve. Indeed, BayesiaLab computes the Direct Effect as the derivative of the response curve around the mean of the values of each driver. If each response curve were linear, the Direct Effect would indeed be a meaningful value for characterizing the entire curve. The question is, does this assumption of linearity hold? In Simpson's Paradox, it certainly did. Due to the binary nature of all variables, the example was inherently linear. Hence, computing a single coefficient for the Direct Effect was adequate for describing the causal effect.

Target Mean Analysis

In this marketing mix example, however, we can make no such assumption. Rather than speculating about the nature of the relationships, we let BayesiaLab estimate the response curves, whatever their shapes might be:

- Select `Menu > Analysis > Visual > Target > Target's Posterior > Curves > Direct Effects`.
- Then, from the options, choose:
  - Target: Mean
  - Variables: Mean
  - Use Hard Evidence
- Click `Display Sensitivity Chart`, which generates a plot of *Sales* as a function of each driver variable.



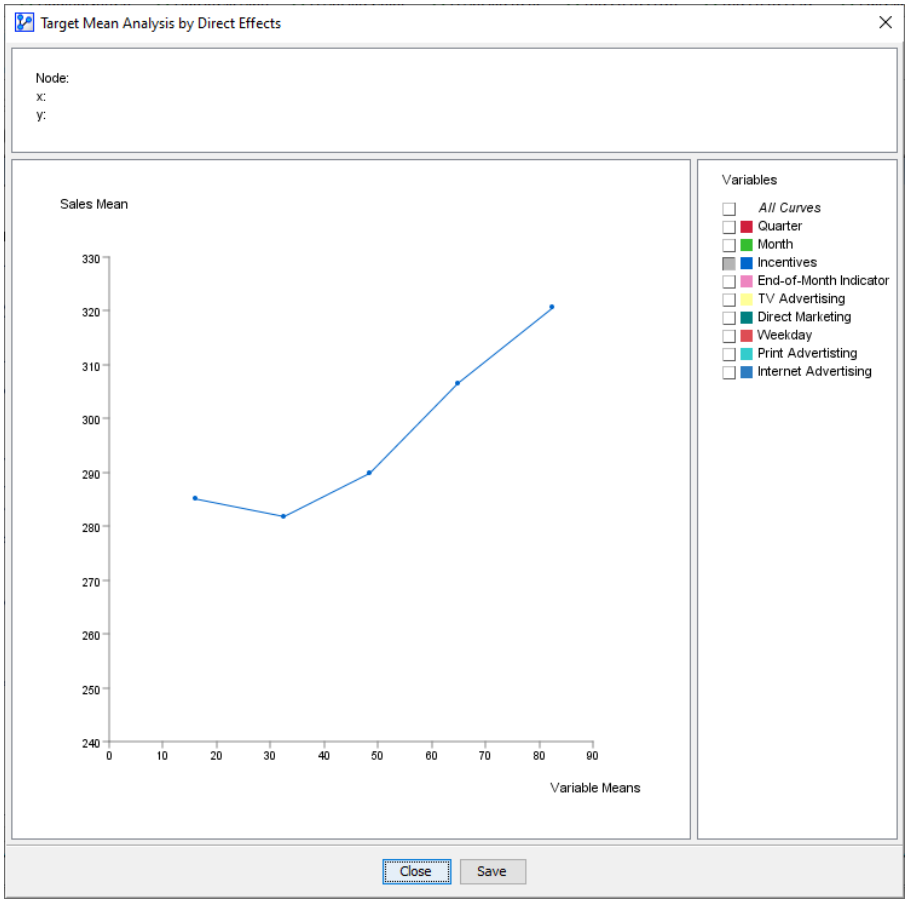
Click to view video

Note that the nodes in the Class Non_Confounder are not included here.

Also, all drivers are represented with their original scales, so Weekday ($Weekday \in \{1, \ldots, 7\}$), Quarter ($Quarter \in \{1, \ldots, 4\}$), and *End-of-Month Indicator* ($End\text{-}of\text{-}Month\ Indicator \in \{0, 1\}$) are all squeezed into the leftmost portion of the plot. Later, we will "decompress" the plot by normalizing the drivers' value range so they all appear on a 1-100 scale.

For now, however, we want to focus on a single driver:

- Remove all curves by clicking the `All Curves` checkbox.
- Select only *Incentives*, which leaves one curve.



The x-values of the points on the curve correspond to the mean values of the discretized states of *Incentives*. Given that we discretized *Incentives* into 5 bins, we have 5 discrete x-values. The y-values are the expected values of the Target Node *Sales* at each corresponding x-value of *Incentives*.

It is important to understand that while the node *Incentives* varies in value, all Confounders are balanced through Likelihood Matching in such a way that *Incentives* is independent of all the Confounders. With that, we can consider setting each value of *Incentives* as a deliberate intervention, and the changes to outcome variable *Sales* are the causal effect of changing *Incentives*. Thus, the curve we see is a causal response curve.

Replicating Target Mean Analysis Step-by-Step

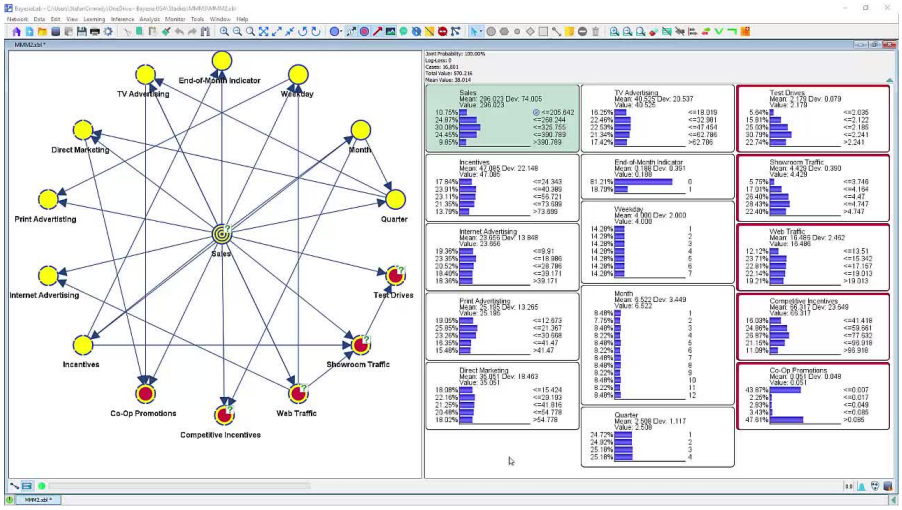
Given the importance of Target Mean Analysis, we now simulate the curve plotting process step by step. We show what is happening in BayesiaLab "behind the scenes" as the curve is plotted using Direct Effects.

- Select the Monitors of all Confounders.
- Apply Fix Probabilities to all Confounders.

This "fixed" status is indicated by purple bars in the Monitors of the Confounders.

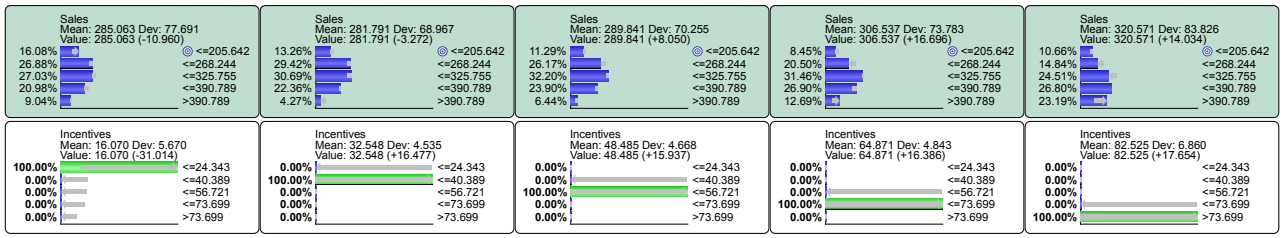
Note that you must not fix the probabilities of the Non-Confounders. Their Monitor bars have to remain blue. *Incentives* and *Sales*, of course, must remain unfixed as well. The former you will manipulate, and the latter's response you want to observe.

- Set *Incentives* to each possible state, from the lowest to the highest. Likelihood Matching maintains the distributions of all the Confounders while *Sales* and the Non-Confounders can respond to the intervention.

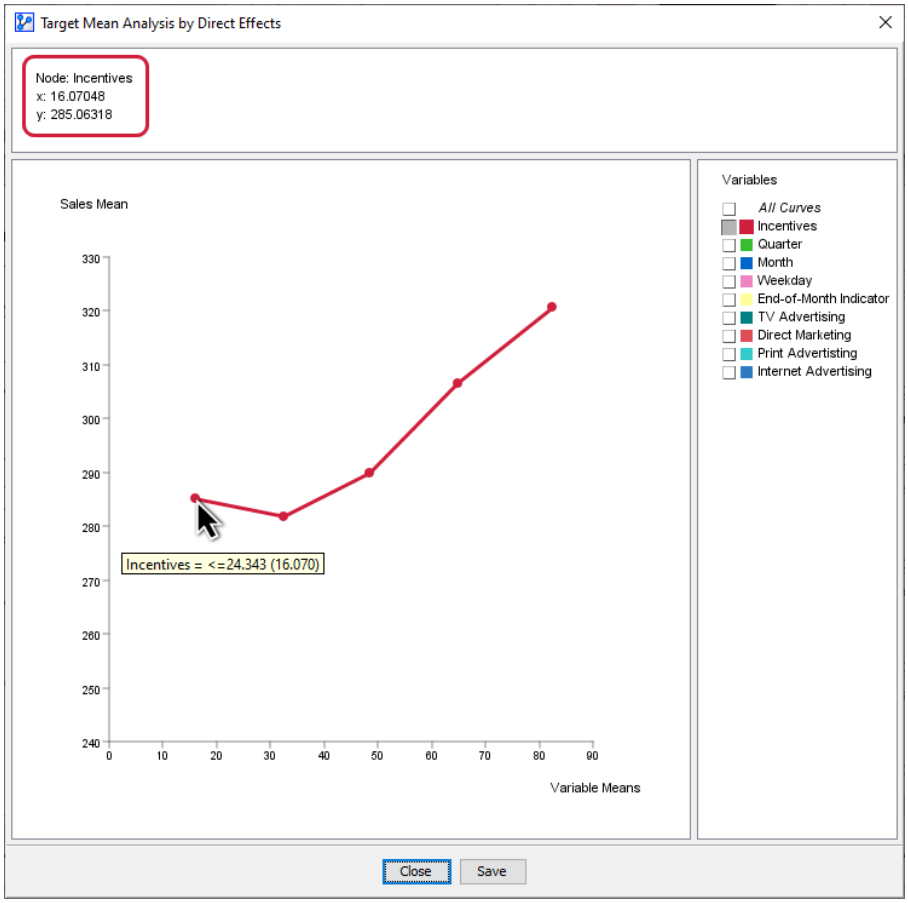


Click to view video

To further illustrate this important process, we have extracted the Monitors for *Incentives* and *Sales* from the Monitor Panel above and lined them up side-by-side:



For instance, given the state *Incentives* `<=24.343`, which has a mean value of 16.070, *Sales* has a mean value of 285.063 (see leftmost panel). So, the mean values of *Incentives* and *Sales* are the x and y coordinates of the first point on the response curve below. The remaining points on the curve are formed in the same way.



Note that this step-by-step approach was only meant to show what BayesiaLab is performing in the background whenever you invoke the Target Mean Analysis plot.

Target Mean Analysis (Direct Effects)

Now that we have explained how the Target Mean Analysis plot is generated, you can let BayesiaLab perform it again automatically for all drivers, similar to what we did in its first run:

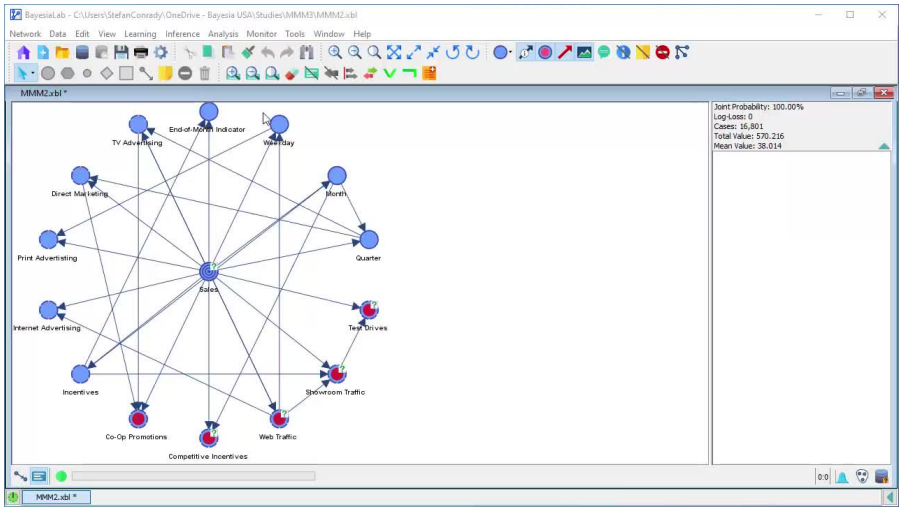
- Select `Menu > Analysis > Visual > Target > Target's Posterior > Curves > Direct Effects`.

At the time, however, it was difficult to interpret and compare the curves as the marketing variables were all recorded on different scales.

- In this run, select `Normalize` in the dialog box, which brings all x-values on a common 0-100 scale.
- Once the plot appears, deselect the calendar-related variables, i.e., Weekday, Quarter, Month, and *End-of-Month Indicator*.

We leave them out for now as they are of lesser interest to us–we cannot modify the calendar after all. Later in our analysis, we will assign a special status to them to formally exclude these variables from being optimized.

This provides an informative picture. We can now characterize the response of *Sales* to the drivers that ACME has under control. More specifically, we observe the exclusive Direct Effect of each driver on *Sales* without confounding effects through the other variables.



Click to view video

Response Curve Patterns

What is perhaps most striking in this plot is that many of the curves appear non-linear. Clearly, any assumption of linearity would not have held. Using the Direct Effects on Target Report, which we used earlier to estimate the slope of these curves, did entirely obscure the dynamics we can observe now.

Furthermore, we can derive several important insights from this plot. For instance, the response curve for *TV Advertising* rises quickly around its middle values, peaks, and then declines. *Direct Marketing* looks like an upside-down U, suggesting that there is a "sweet spot" in terms of marketing exposure. The curve for *Print Advertising* looks S-shaped, while the variable *Incentives* appears to be exponentially linked to *Sales*.

The "wild mix" of response curve patterns highlights the inherent difficulty of marketing mix optimization. While the curves themselves may be individually meaningful to a marketing expert, it is far from obvious how much should be allocated to each marketing channel within the constraints of an overall marketing budget.

Machine Learning

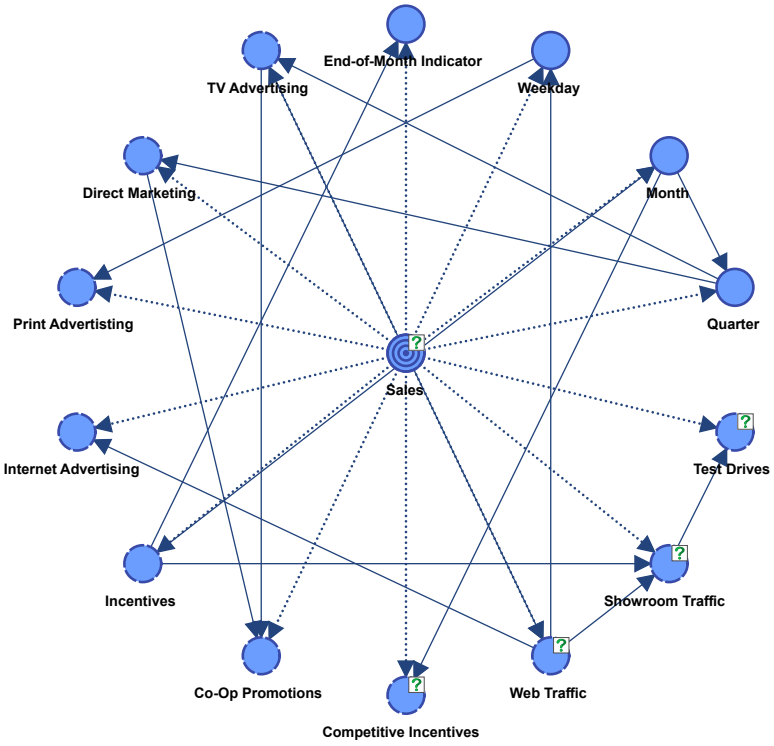
The way out of this predicament is to turn to modeling. Instead of working with an enormously large joint probability table, we will approximate the joint probability distribution of the domain with a model. But does this not take us back to the problem of being unable to machine-learn a causal model? No, because we do not need a causal model. Rather, we merely need a statistical model to approximate the statistical relationships of the variables in our domain. Of course, that task is easy.

In earlier chapters, we have already introduced various machine learning algorithms in BayesiaLab that can generate Bayesian networks from data.

Given that we have a target variable, i.e., *Sales*, Supervised Learning will be the appropriate approach. First, we import the dataset:

Download: `ACME_Data.csv`

We discretize all variables with the R2GenOpt algorithm into five states. Our choice of this discretization algorithm follows the rationale that we presented in earlier chapters. After importing the dataset, we use the Augmented Naive Bayes algorithm to learn a network with *Sales* as the Target Node. The result is shown below.



Here, we will omit a discussion of the network quality. Rather, we proceed with the given network and look at how to use it to understand our problem domain.

As is, this network has the familiar appearance of a predictive model. And we have kept emphasizing that machine learning can only produce predictive models, which, in turn, are only capable of performing observational inference. However, it is causal inference that we require for the purpose of marketing mix optimization. Clearly, a causal interpretation of the arcs in the above network would not make any sense. After all, *Sales* is the outcome, not the cause.

This does not matter because we had no intention of finding a causal model with machine learning. We had already given up on finding the true causal model. The model we obtained is only meant to compactly represent the Joint Probability Distribution (JPD) of all variables in this domain.

Our key to causal inference is that we have the JPD, as represented by the machine-learned Bayesian network, and we know, based on domain knowledge and the Disjunctive Cause Criterion, which variables are Confounders.

One issue remains open, though, and that is what mechanism to use for estimation. Recall matching as an estimation technique and, in particular, Likelihood Matching (see Intuition for Matching). Here, however, we are not dealing with just one covariate Z, but rather with 10, of which 8 are confounders.

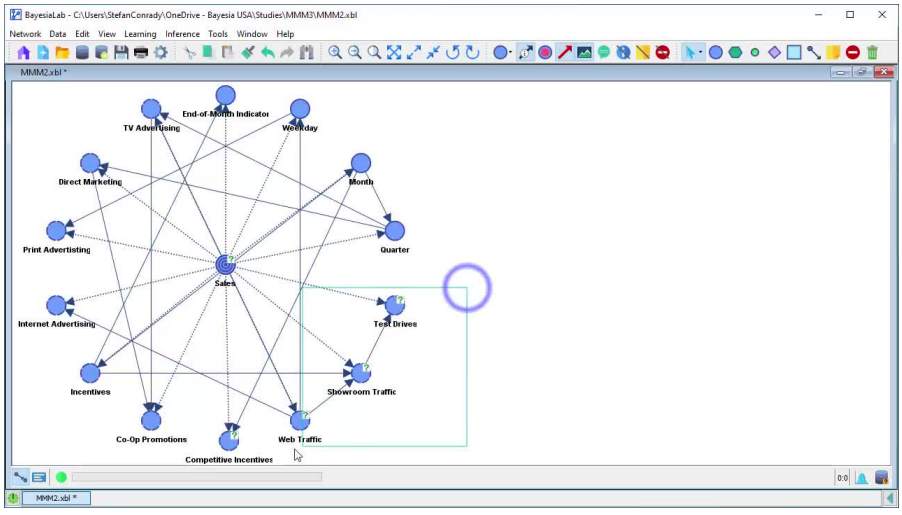
Confounders and Non-Confounders

Before proceeding to estimation in BayesiaLab, we need to formally declare which variables are Confounders. In BayesiaLab, all nodes are considered Confounders by default. Hence, we need to declare the inverse condition, i.e., we must tell BayesiaLab which nodes are *not* Confounders or "Non-Confounders."

Designate Non-Confounders

As per the rationale we laid out earlier, we need to mark the pre-treatment variables Competitive Print, Competitive TV, and Competitive TV and assign them to the predefined Class Non_Confounders.

- Select the nodes to be added to the Class Non_Confounders.
- Right-click on one of them to bring up the Contextual Menu.
- Select `Properties > Classes > Add`.
- Check the radio box for Predefined Class and pick Non_Confounder from the drop-down menu.



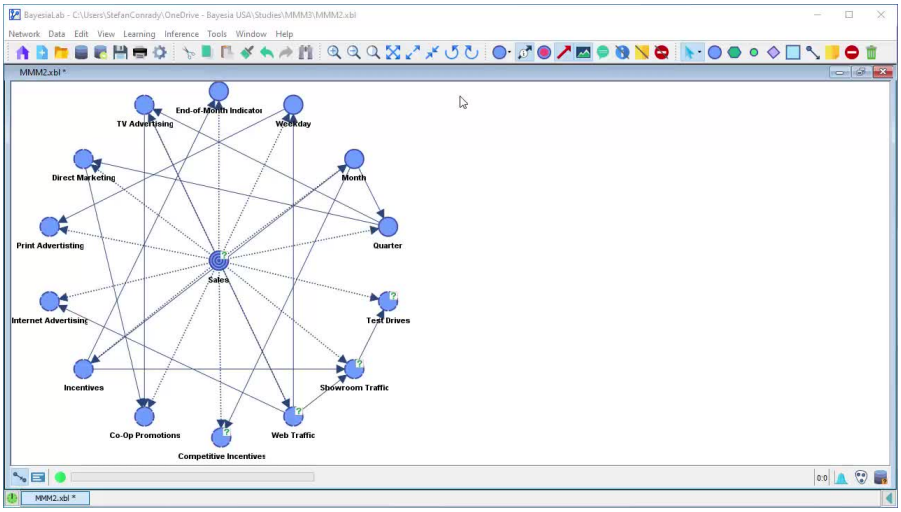
Click to view video

As a result, we now have a clear distinction between Confounders and Non-Confounders and can perform an effect estimation on that basis.

Add Color to Non-Confounders

To highlight this distinction, we assign colors to each Class.

- Right-click on the Graph Panel background to bring up the Contextual Menu.
- Select Edit Classes.
- In the Class Editor, highlight Non_Confounder in the list and click `Associate Colors`.
- From the dialog box, select Associate Default Colors with Classes and click `OK`.



Click to view video

Now, all Non-Confounders are highlighted in red.

Many Confounders, Many Treatments, Many Levels

Under Estimation Challenges, we already pointed out that stratification is no longer feasible as an estimation technique, which leaves us with matching. However, this example exceeds in complexity of Simpson's Paradox in a number of ways. In Simpson's Paradox, we had one Confounder and one treatment variable, which had only one treatment level. Now we have several Confounders and several treatments. And many of the treatment variables have multiple treatment levels. As a result, the task of matching is no longer straightforward. Now, we need to simultaneously balance all Confounders with regard to each treatment variable in such a way that each treatment remains at its marginal probability level. With that in place, we can simulate different treatment levels and observe the corresponding outcomes. What we then observe in the different outcomes is indeed the causal effect. It is easy to imagine that the computational effort for this process is substantial. It goes beyond the scope of this book to explain the details of BayesiaLab's Likelihood Matching algorithm. For our purposes, we only need to know how to invoke the algorithm in BayesiaLab for causal effect estimation. Likelihood Matching is launched whenever we run Direct Effects in BayesiaLab.

Target Optimization

Summary of Node Roles

Target Node:

- *Sales*

Confounders:

- *TV Advertising*
- *Internet Advertising*
- *Print Advertising*
- *Direct Marketing*
- *Incentives* (i.e., price discounts)

Confounders and Not Observable:

- Quarter
- Weekday
- Month
- *End-of-Month Indicator*

Non-Confounders:

- *Co-Op Promotions*
- *Competitive Incentives*
- *Web Traffic*
- *Showroom Traffic*
- *Test Drives*

> As we proceed, we furthermore assume that there are no unobserved Confounders. Such an assumption can only be justified on theoretical grounds. Given that this example represents a fictional domain, there is no purpose in debating the validity of the assumption.

Target Optimization

Now we are ready to set the parameters of the Target Optimization.

- Select `Menu > Analysis > Target Optimization > Genetic`.
- Set `Profile Search Criterion` to `Mean`.
- Set `Criterion Optimization` to `Maximization`.
- Check `Take Into Account` the Resources. This means that for each to-be-tested scenario, BayesiaLab computes the value of the Class Resource, i.e., the value of the Function Node F1. Furthermore, you need to specify a range of acceptable values.
- Check `Target Resources` and set this value to the available budget. By default, the value is set to the current value of F1. In our case, it is the sum of the means of the marginal distributions of the Confounders.
- Do not check `Take Into Account` the Joint Probability, as it does not apply to this example.

It would be applicable to models based on disaggregated data, e.g., with cross-sectional data representing the behavior of individuals.

- Some optimization tasks may require a trade-off between constraints and target achievement. The Weighting option allows us to prioritize specific optimization criteria. Set Resources to 10 to ensure that the search algorithm stays as close as possible to the specified Target Resources.
- Search Method depends on the problem domain. We discussed a similar set of options in the context of Target Dynamic Profile.
- Set Numerical Evidence Proportional to: Mean.
- Set Distribution Estimation Method to Binary. Here, the optimization algorithm needs to modify the mean of each variable by setting Binary evidence (see Chapter 7, Binary Evidence). This is important because we are trying to determine which specific values–not distributions–achieve the maximum level of *Sales*. From a theoretical viewpoint, we could certainly search for the optimal distribution, but that is presumably not practical. Marketing budgets get approved and allocated on the basis of single-point dollar values, not distributions.
- Clicking the `Edit Variations` button brings up the Variation Editor.

We already discussed the Variation Editor in the context of the Target Dynamic Profile. Here, we can further constrain individual variables in addition to the overall constraint determined by the specified Target Resource. A typical reason could be that some marketing expenditures may be locked into long-term contracts, which cannot be changed,

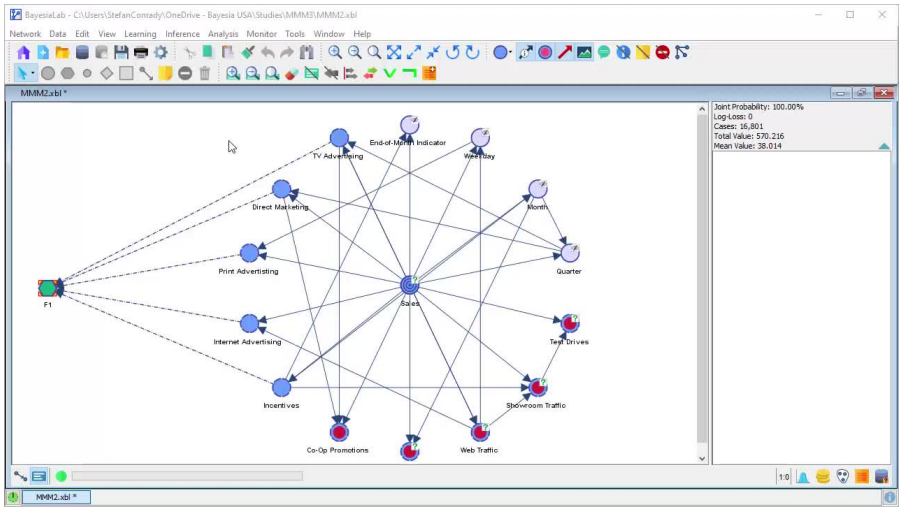
- Set Intermediate Points to 10.

This setting refers to the number of points to test for each variable. To manage a potentially large computing load, it is good practice to start with a smaller number of Intermediate Points, e.g., 10, and later perform a search with a finer grid.

- Check Direct Effects. As we have emphasized throughout this chapter, we are looking for the causal effects of the driver variables. Otherwise, BayesiaLab would perform an optimization based on the Total Effect and produce meaningless results for our purposes.
- Output determines how BayesiaLab stores the solutions found by Target Optimization. In our case, we want to save all new scenarios and overwrite any existing scenarios. Also, as the optimization can carry on for a long time, it is important to know that it can be stopped anytime. In that case, all solutions computed up to that point are saved as Evidence Scenarios.
- Set Return the n Best Solutions to 10
- In terms of the Genetic Settings, we recommend keeping the defaults. Computer scientists will be familiar with these algorithm-related options. For the type of problem at hand, however, little can be gained by changing the defaults.
- By definition, genetic search algorithms continue mutating scenarios endlessly in order to find better solutions. Such an algorithm will never come to a natural conclusion. Hence, the Genetic Stop Settings are a practical way to stop the algorithm when no improvement has been observed after a certain number of iterations.
- Clicking `OK` starts the optimization, and we can monitor the activity in the status line at the bottom of the screen.
- Once the Genetic Stop Criterion is met or when you manually terminate the optimization, BayesiaLab delivers the Optimization Report for the Target Node *Sales*.

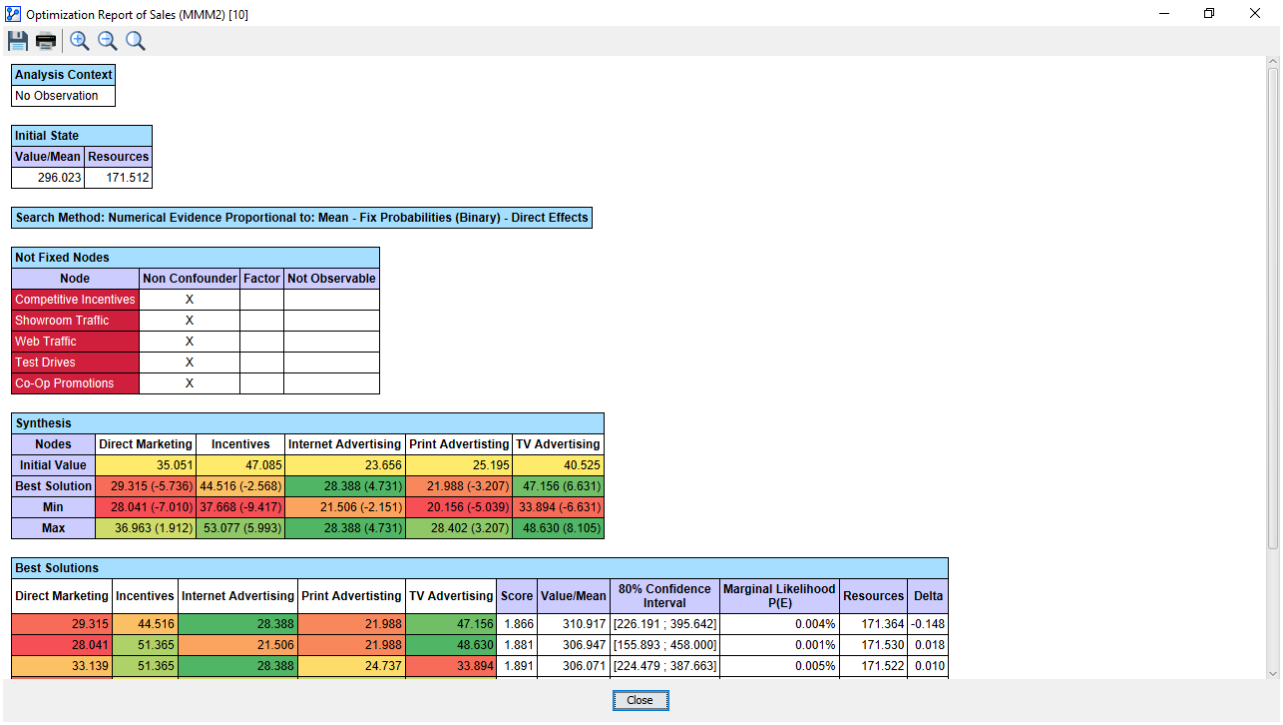
Note that there is no progress bar, as no predetermined endpoint exists for a genetic optimization algorithm. We can, however, monitor the score to see the development of solutions.

Additionally, we need to take note of any warning symbols appearing in the bottom right corner of the main window. As Likelihood Matching is performed repeatedly throughout the optimization, there is the possibility of the Likelihood Matching algorithm–not the Target Optimization algorithm–being unable to converge. A yellow warning triangle icon ⚠ indicates this particular condition. Further details can be retrieved from the Console.



Click to view video

Target Optimization Report



Optimization Report of Sales (MMM2) [10]

Analysis Context

No Observation

Initial State

Value/Mean	Resources
296.023	171.512

Search Method: Numerical Evidence Proportional to: Mean - Fix Probabilities (Binary) - Direct Effects

Not Fixed Nodes

Node	Non Confounder	Factor	Not Observable
Competitive Incentives	X		
Showroom Traffic	X		
Web Traffic	X		
Test Drives	X		
Co-Op Promotions	X		

Synthesis

Nodes	Direct Marketing	Incentives	Internet Advertising	Print Advertising	TV Advertising
Initial Value	35.051	47.085	23.656	25.195	40.525
Best Solution	29.315 (-5.736)	44.516 (-2.568)	28.388 (4.731)	21.988 (-3.207)	47.156 (6.631)
Min	28.041 (-7.010)	37.668 (-9.417)	21.506 (-2.151)	20.156 (-5.039)	33.894 (-6.631)
Max	36.963 (1.912)	53.077 (5.993)	28.388 (4.731)	28.402 (3.207)	48.630 (8.105)

Best Solutions

Direct Marketing	Incentives	Internet Advertising	Print Advertising	TV Advertising	Score	Value/Mean	80% Confidence Interval	Marginal Likelihood P(E)	Resources	Delta
29.315	44.516	28.388	21.988	47.156	1.866	310.917	[226.191 ; 395.642]	0.004%	171.364	-0.148
28.041	51.365	21.506	21.988	48.630	1.881	306.947	[155.893 ; 458.000]	0.001%	171.530	0.018
33.139	51.365	28.388	24.737	33.894	1.891	306.071	[224.479 ; 387.663]	0.005%	171.522	0.010

Close

- Initial States: Value/Mean refers to the mean value of the marginal distribution of the Target Node *Sales*.
- Resources represents the corresponding value of F1.
- Search Methods repeats the options we chose. While this may appear redundant, it is critically important to understand the precise conditions under which the solution was found.
- For the same purpose, the report lists the Not Fixed Nodes, which are the Non-Confounders that we had defined, meaning that they were not included in Likelihood Matching.
- We find the main part of the report in the Synthesis table. Here, we see the Initial State, which lists the mean values of the marginal distributions of the drivers. Best Solution shows the optimal levels of each driver. The values in parentheses indicate the deviation from the Initial State. Thus, we can easily see by what amount the marketing variables need to be changed. For example, the report recommends decreasing the *Direct Marketing* spend by 5.736 units.
- In the Best Solutions table at the bottom of the report, BayesiaLab lists the top scenarios identified plus the corresponding achievement in terms of the Target Node *Sales*. This tells us, for instance, that *Sales* will increase to 310.917 if the marketing variables are set to the specified levels. At the same time, the Resources would amount to 171.364, which is very close to the specified constraint.

If we suspect that we might have missed out on a potentially better solution, we could re-run the optimization using more Intermediate Points or set a different Stop Criterion. The potential extent

of the search is ultimately a function of the available computing resources. However, given the size of the search space and the non-exhaustive nature of genetic search algorithms, we can never be certain to have found the optimum.

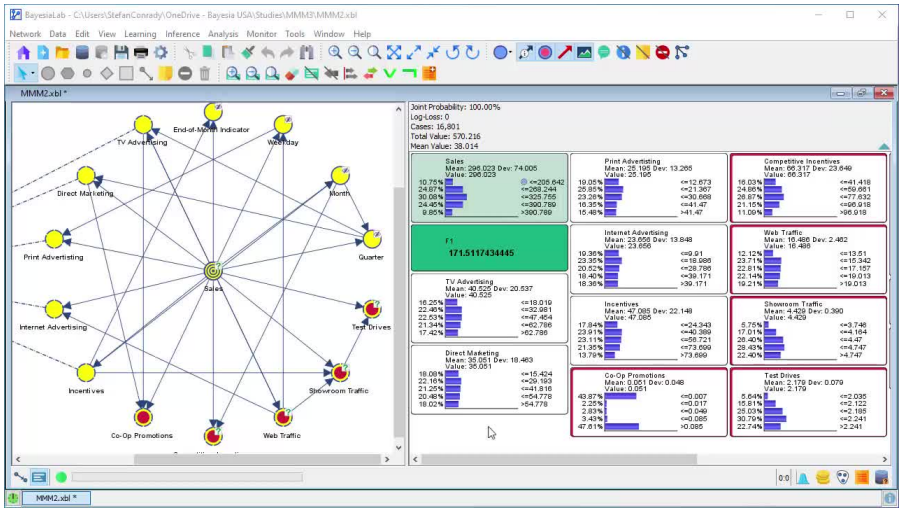
It's Causal!

At this point, we need not shy away from causal language. All our causal assumptions had been explicitly specified earlier, and on that basis, we performed a causal optimization. If the assumptions can be justified, the effect estimation is causal and does not need to be circumscribed with cautious language. Given our input and applying hard-and-fast identification criteria, we are not in a gray area in terms of the causal interpretation of the results. If the results are to be challenged regarding their causal validity, we need to go straight back to the assumptions. The causal claims stand and fall with the assumptions we make, not with the estimation techniques.

Evidence Scenarios

While we may immediately gravitate toward the best solution listed first in the solutions table, we can examine all proposed solutions, which are stored as Evidence Scenarios:

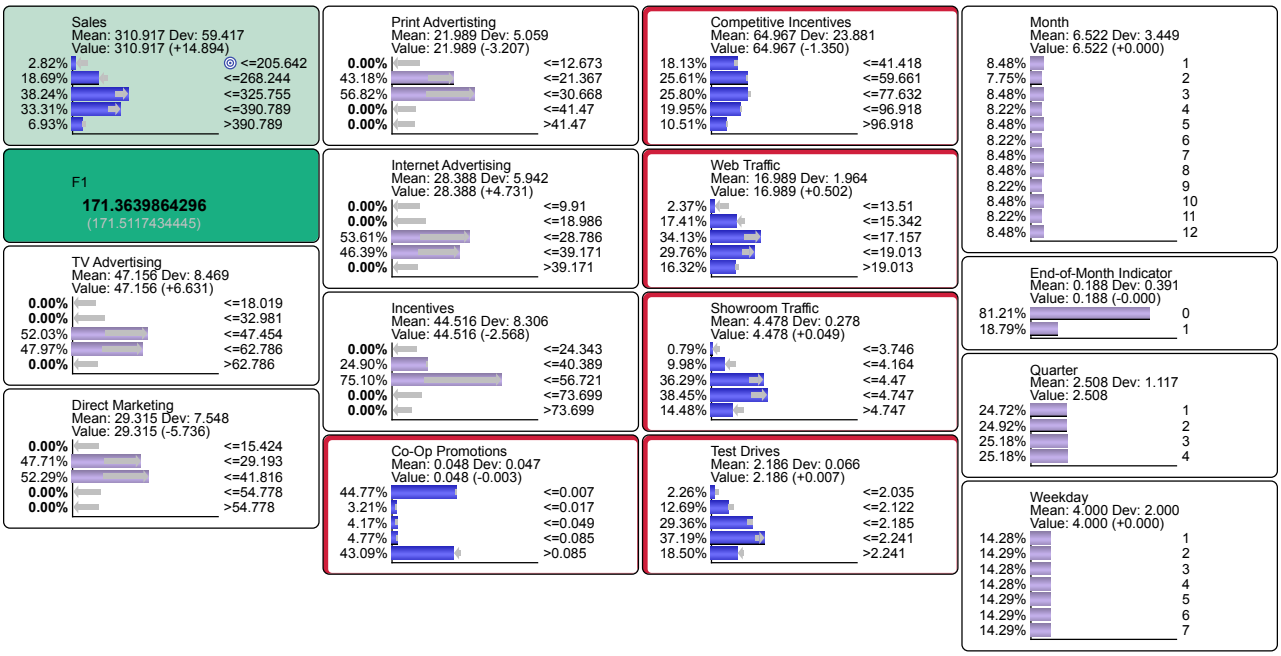
- Right-click on the Evidence Scenario icon .
- Check `Include Not Observable` to see the complete solution, including the values of the Not-Observable nodes.
- Select the Evidence Set with Index 0., which sets all nodes to the values specified under Best Solution in the Optimization Report.



Click to view video

This simulation confirms what we obtained in the Optimization Report, i.e., we can increase *Sales* by almost 15 units (per day) with the same budget. As a by-product of retrieving this Evidence Set, we can observe the corresponding values of the Non-Confounders. Similarly, we can evaluate the remaining scenarios for their plausibility. A seemingly inferior scenario could perhaps be more practical to implement.

For reference, all Monitors corresponding to Evidence Set 0 are shown below:



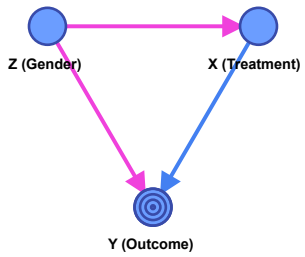
Summary

This chapter presented a comprehensive workflow for optimizing the marketing mix of an organization. The key to this approach was using a machine-learned Bayesian network model in combination with causal assumptions in the form of confounder selection according to the Disjunctive Cause Criterion. BayesiaLab's Likelihood Matching algorithm facilitated the causal effect estimation and the subsequent optimization of marketing drivers.

Total Effect vs Direct Effect

Simpson's Paradox Revisited

Why are we using the term "Direct Effect" instead of "causal effect," which is obviously what we are looking for? It helps to recall the Simpson's Paradox example from the previous chapter. Through path analysis, we were able to distinguish between causal (blue ➡) and non-causal paths (pink ➡). We argued that the arc $X \rightarrow Y$, i.e., the direct causal path, represents the causal effect. By adjusting for Z and thus blocking the non-causal path $X \leftarrow Z \rightarrow Y$, we were able to isolate the "direct effect" of X on Y. However, if had we not adjusted for Z, both the causal and the non-causal path would remain open, and we would obtain the "total effect." This is indeed the nomenclature we follow in BayesiaLab.



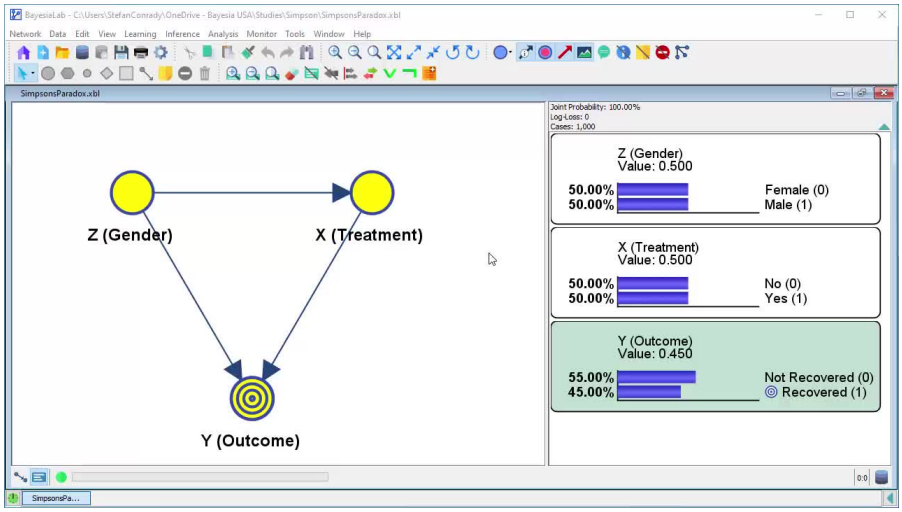
By invoking Direct Effect, BayesiaLab will automatically perform Likelihood Matching on all Confounders and estimate the causal effect.

Direct Effect

To emphasize the distinction between Direct Effect and Total Effect, we look one more time at Simpson's Paradox:

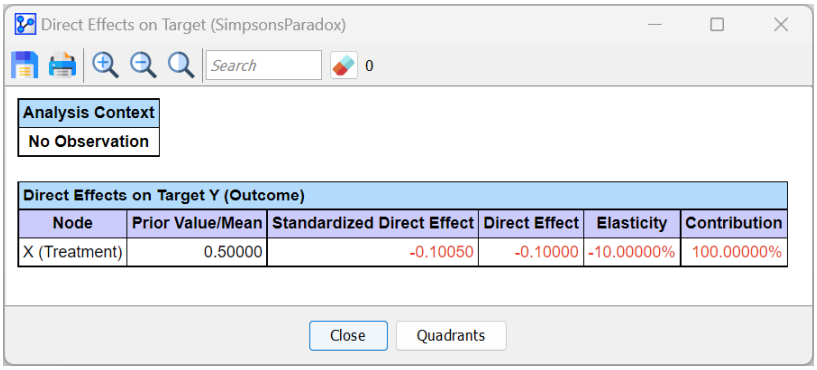
Download: `SimpsonsParadox.xbl`

- Click X and then select `Menu > Analysis > Report > Target Analysis > Direct Effects on Target`.



Click to view video

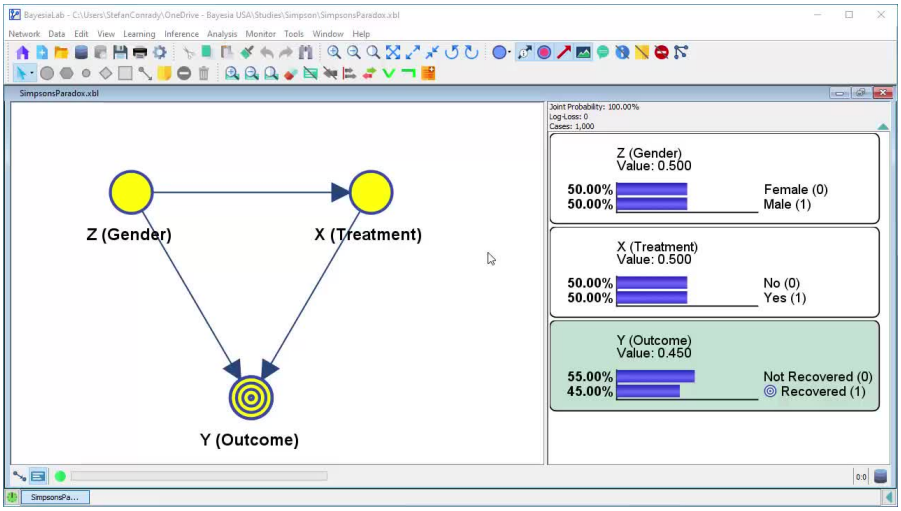
We immediately obtain a report that shows a Direct Effect of -0.1. This value is identical to the Average Treatment Effect we computed in the previous chapter. As expected, adjustment by stratification, Graph Surgery, and Likelihood Matching provides the same effect estimate.



Total Effect

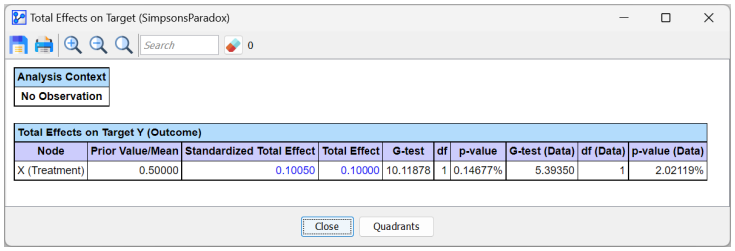
For comparison, we now estimate the Total Effect:

- Select `Menu > Analysis > Report > Target Analysis > Total Effects on Target`.



Click to view video

The resulting report window now shows the Total Effect, which amounts to +0.1.



This result matches the naive estimator, i.e., the effect we observe when considering the whole population, which is clearly not the causal effect. So, why would we need to estimate the Total Effect at all? It would be the only correct estimator for performing observational inference, i.e., prediction. If we were merely observing treated versus not treated patients instead of performing an intervention, the Total Effect provides the expected change of the outcome variable.